CBCS SCHEME

USN | 1 | C | R | 2 | U | C | S | 2 | 5 | 7 |

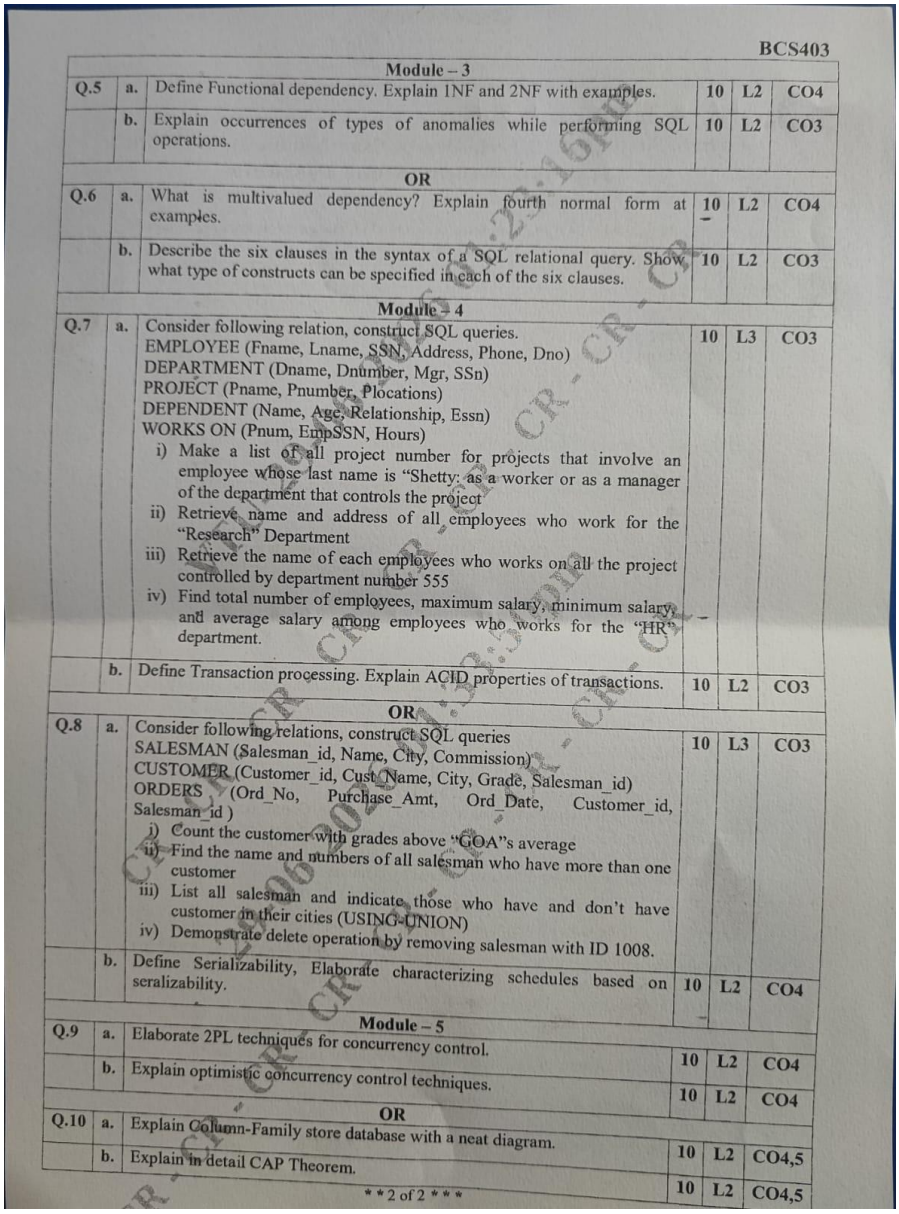
BCS403

Fourth Semester B.E./B.Tech. Degree Examination, June/July 2026
Database Management Systems

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

			M	L	C
		Module – 1			
Q.1	a.	Define DBMS. Explain the main characteristics of the database approach.	5	L2	CO1
	b.	With a neat diagram, explain Three –Schema architecture. Justify the need of mappings among schema levels.	8	L2	CO1
	c.	Write short notes on : i) DB languages ii) DB Interfaces.	7	L1	CO1
		OR			
Q.2	a.	Differentiate between traditional file system Vs database systems.	5	L2	CO1
	b.	With a neat diagram, discuss various DB system environment components.	8	L2	CO1
	c.	Draw an ER diagram, for ONLINE SHOPPING database with atleast five entities. Specify primary key and participation constraints.	7	L3	CO2
		Module – 2			
Q.3	a.	Explain briefly domain, key, integrity and referential integrity constraints with necessary examples.	10	L2	CO1
	b.	Consider following relations and construct Relational Algebra queries for the following : SAILOR (Sid, Sname, Rating, Age) BOAT (Bid, Bname, Color) RESERVE (Sid, Bid, Day) i) Retrieve the color of boats reserved by "Deepak" ii) Retrieve the sailor names who have reserved 'Red' or 'Green' boats iii) Retrieve the SID of sailors with age over 25 who have not reserve a boat iv) Retrieve the total count of boats reserved by each sailor	10	L3	CO2
		OR			
Q.4	a.	Explain ER-to-Relational mapping algorithm with example	10	L2	CO1
	b.	Consider the following relations and construct Relations Algebra Queries. STUDENT(Sid, Sname, Major, GPA) FACULTY (Fid, Fname, Dept, Designation, Salary) COURSE (Cid, Cname, Fid) ENROLL (Cid, Sid, Grade) i) Retrieve the name of facilities who are receiving salary more than 34567.00 ii) List the name of all faculty members who teaches the course "BCS 403". iii) List all the departments having an average salary of above 45678.00 iv) List the names of students enrolled for the course "Mastering SGC".	10	L3	CO2



BCS403

		Module – 3			
Q.5	a.	Define Functional dependency. Explain 1NF and 2NF with examples.	10	L2	CO4
	b.	Explain occurrences of types of anomalies while performing SQL operations.	10	L2	CO3
		OR			
Q.6	a.	What is multivalued dependency? Explain fourth normal form at examples.	10	L2	CO4
	b.	Describe the six clauses in the syntax of a SQL relational query. Show what type of constructs can be specified in each of the six clauses.	10	L2	CO3
		Module – 4			
Q.7	a.	Consider following relation, construct SQL queries. EMPLOYEE (Fname, Lname, SSN, Address, Phone, Dno) DEPARTMENT (Dname, Dnumber, Mgr, SSn) PROJECT (Pname, Pnumber, Plocations) DEPENDENT (Name, Age, Relationship, Essn) WORKS ON (Pnum, EmpSSN, Hours) i) Make a list of all project number for projects that involve an employee whose last name is "Shetty: as a worker or as a manager of the department that controls the project ii) Retrieve name and address of all employees who work for the "Research" Department iii) Retrieve the name of each employees who works on all the project controlled by department number 555 iv) Find total number of employees, maximum salary, minimum salary, and average salary among employees who works for the "HR" department.	10	L3	CO3
	b.	Define Transaction processing. Explain ACID properties of transactions.	10	L2	CO3
		OR			
Q.8	a.	Consider following relations, construct SQL queries SALESMAN (Salesman_id, Name, City, Commission) CUSTOMER (Customer_id, Cust_Name, City, Grade, Salesman_id) ORDERS (Ord_No, Purchase_Amt, Ord_Date, Customer_id, Salesman_id) i) Count the customer with grades above "GOA"s average ii) Find the name and numbers of all salesman who have more than one customer iii) List all salesman and indicate those who have and don't have customer in their cities (USING-UNION) iv) Demonstrate delete operation by removing salesman with ID 1008.	10	L3	CO3
	b.	Define Serializability, Elaborate characterizing schedules based on seralizability.	10	L2	CO4
		Module – 5			
Q.9	a.	Elaborate 2PL techniques for concurrency control.	10	L2	CO4
	b.	Explain optimistic concurrency control techniques.	10	L2	CO4
		OR			
Q.10	a.	Explain Column-Family store database with a neat diagram.	10	L2	CO4,5
	b.	Explain in detail CAP Theorem.	10	L2	CO4,5

2 of 2 *

Module 1

Q.1. a. Define DBMS. Explain the main characteristics of the database approach (5M)

A database is a collection of logically related data that is organized and stored in such a way that it can be easily accessed, managed, and updated. It represents some aspect of the real world and is designed to serve the information needs of an organization or application.

1. Self-Describing Nature of a Database System

A database system contains not only the actual data but also information about the structure and constraints of the data. This information, called **metadata**, is stored in the **system catalog or data dictionary**.

The metadata describes:

- Names of relations and attributes.
- Data types of attributes.
- Constraints and relationships among data.

Thus, the database is self-describing in nature.

2. Program–Data Independence

In the database approach, application programs are independent of the physical storage details of data. Changes made in the database structure or storage organization do not require modifications to the application programs.

This characteristic provides:

- **Logical data independence**
- **Physical data independence**

Hence, changes in one level do not affect the other levels.

3. Support for Multiple Views of Data

Different users have different requirements. A DBMS allows multiple users to view the same database in different ways according to their needs.

For example:

- A student may view marks and attendance details.
- A faculty member may view student and course information.
- The administrator may access the entire database.

Thus, multiple views provide simplicity and security.

4. Data Sharing and Multiuser Transaction Processing

A database can be shared simultaneously by many users and applications. The DBMS provides mechanisms for controlling concurrent access to maintain consistency.

It supports:

- Concurrent transactions.
- Data sharing among multiple users.
- Recovery from failures.
- Maintenance of database consistency.

Hence, several users can access and update the database without causing conflicts.

5. Controlled Redundancy

The database approach minimizes unnecessary duplication of data. Since the same data is shared by multiple users, redundancy is reduced considerably.

Advantages include:

- Reduced storage space.
- Elimination of inconsistencies.
- Improved data integrity.

6. Data Integrity and Consistency

The DBMS enforces integrity constraints to ensure that the stored data remains accurate and consistent.

Examples include:

- Entity integrity.
- Referential integrity.
- Domain constraints.

Thus, the correctness of data is maintained.

7. Data Security

A database system provides mechanisms to protect data from unauthorized access.

Security features include:

- User authentication.
- Authorization and access privileges.
- Password protection.

- Encryption techniques.

Therefore, only authorized users can access sensitive information.

8. Backup and Recovery

DBMS provides facilities for backing up the database and recovering it in case of hardware failures, software failures, or power outages.

Recovery mechanisms help restore the database to a consistent state and prevent loss of data.

9. Transaction Processing and Concurrency Control

A DBMS supports transaction management based on the ACID properties. It allows multiple transactions to execute concurrently while maintaining consistency and isolation.

This feature ensures:

- Reliable execution of transactions.
- Prevention of anomalies.
- Correctness of concurrent operations.

10. Enforcement of Standards

The database approach allows standards for naming conventions, storage formats, and data representation to be enforced uniformly throughout the organization.

This promotes:

- Better data administration.
- Easier maintenance.
- Improved interoperability.

Q.2.b. With a neat diagram, explain Three-Schema architecture. Justify the need of mappings among (8M)

Three-schema architecture is a framework that helps achieve database system properties of data independence and data abstraction using three levels of data description.

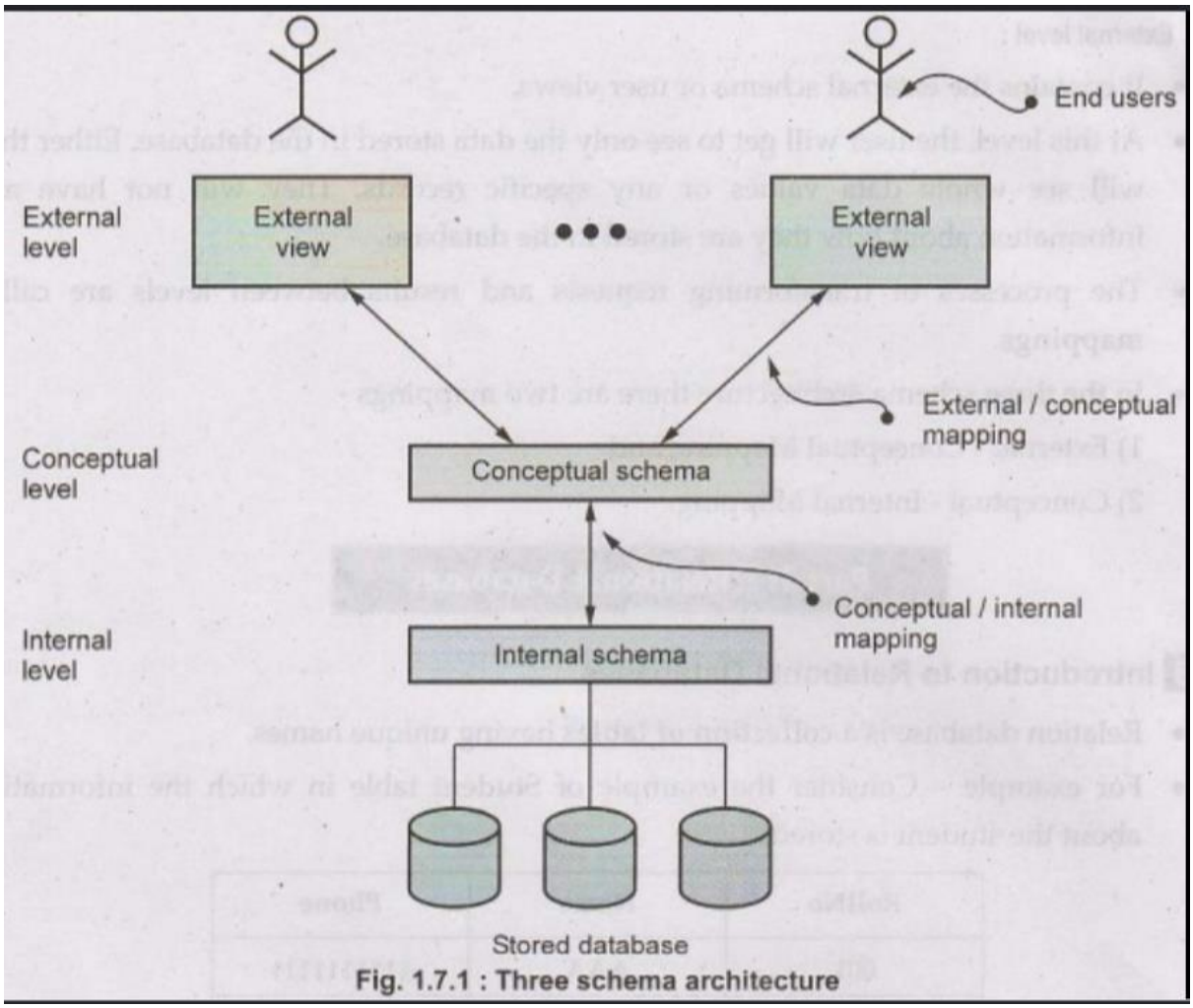


Fig. 1.7.1 : Three schema architecture

1. The internal level has an internal schema, -describes the physical storage structure of the database. -uses a physical data model and describes the complete details of data storage and access paths for the database.

2. The conceptual level has a conceptual schema, -describes the structure of the whole database for a community of users -hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints -representational data model is used to describe the conceptual schema when a database system is implemented -This implementation conceptual schema is often based on a conceptual schema design in a high-level data model.

3. The external or view level includes a number of external schemas or user views, -describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. -As in the previous level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level data model.

The DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database. If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view. The processes of transforming requests and results between levels are calledmappings. These mappings may be time-consuming, so some DBMSs—especially those that are meant to support small databases—do not support external views. Even in such systems, however, a certain amount of mapping is necessary to transform requests between the conceptual and internal levels.

Q.2.c. Write short notes on: i) DB languages ii) DB Interfaces (7M)

(i) Database Languages (DB Languages)

Database Languages are specialized languages provided by a DBMS to define, manipulate, control, and manage data in a database. They enable users and administrators to create database objects, retrieve and update data, and control access to the database.

Types of Database Languages

1. Data Definition Language (DDL)

DDL is used to define and modify the structure (schema) of the database.

Functions:

- Create tables
- Modify table structure
- Delete database objects

Commands:

- CREATE
- ALTER
- DROP
- TRUNCATE
- RENAME

Example:

```
CREATE TABLE STUDENT(
USN INT,
Name VARCHAR(30),
Branch VARCHAR(10)
);
```

2. Data Manipulation Language (DML)

DML is used to retrieve, insert, modify, and delete data stored in database tables.

Commands:

- INSERT
- UPDATE
- DELETE
- SELECT

Example:

```
INSERT INTO STUDENT
VALUES(101,'Ravi','CSE');
```

3. Data Control Language (DCL)

DCL is used to control user access and provide database security.

Commands:

- GRANT
- REVOKE

Example:

```
GRANT SELECT
ON STUDENT
TO USER1;
```

4. Transaction Control Language (TCL)

TCL manages database transactions and ensures data consistency.

Commands:

- `COMMIT`
- `ROLLBACK`
- `SAVEPOINT`

(ii) Database Interfaces (DB Interfaces)

A **Database Interface** is the medium through which users interact with the DBMS. Different interfaces are provided based on the user's knowledge and application requirements.

Types of Database Interfaces

1. Command-Line Interface (CLI)

Users enter SQL commands directly through a command prompt.

Example:

```
SELECT * FROM STUDENT;
```

Suitable for database administrators and experienced users.

2. Menu-Based Interface

Users select operations from menus instead of typing commands.

Example:

- ATM machines
- Library management systems

Easy for beginners.

3. Forms-Based Interface

Users enter data through forms, and the DBMS generates SQL statements internally.

Example:

- Student admission form
- Employee registration form

Commonly used in business applications.

4. Graphical User Interface (GUI)

Provides windows, buttons, icons, and menus for interacting with the database.

Examples:

- Oracle SQL Developer
- MySQL Workbench

Easy to learn and use.

5. Natural Language Interface

Allows users to interact with the database using natural language.

Example:

```
Show all students in CSE department.
```

The system converts the request into an SQL query.

6. Web-Based Interface

Users access databases through web browsers using web applications.

Examples:

- Online banking
- E-commerce websites
- Student portals

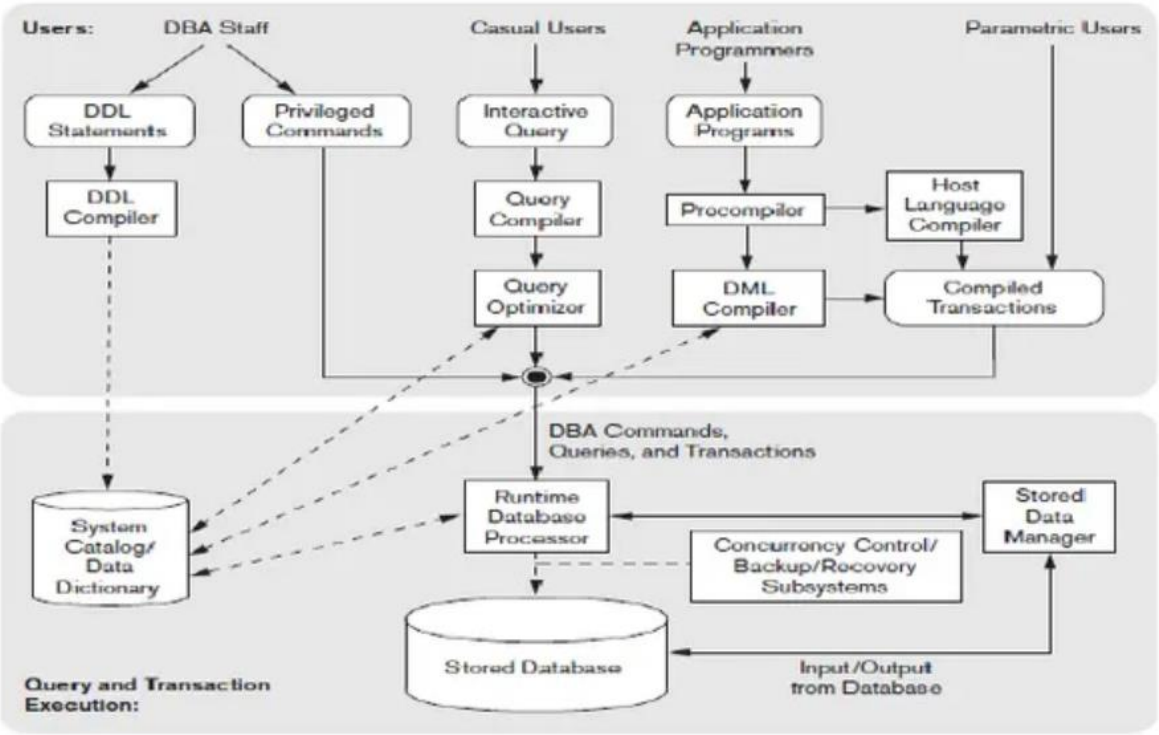
Q.2.a. Differentiate between traditional file system vs database systems (5M)

A **Traditional File System** stores data in separate files managed by the operating system, whereas a **Database Management System (DBMS)** stores and manages data in a centralized database with better security, consistency, and efficient data access.

Traditional File System	Database System (DBMS)
Data is stored in separate files.	Data is stored in a centralized database.
High data redundancy due to duplicate data.	Data redundancy is minimized through normalization.
Data inconsistency is common because the same data may exist in multiple files.	Ensures data consistency by maintaining a single copy of data.
Data sharing is difficult.	Supports easy and controlled data sharing among multiple users.
Provides limited security.	Provides strong security using authentication, authorization, and access control.
No concurrency control for multiple users.	Supports concurrent access using concurrency control techniques.

Backup and recovery are difficult and mostly manual.	Provides automatic backup and recovery mechanisms.
Data independence is not supported.	Supports logical and physical data independence.
Relationships among data are difficult to maintain.	Relationships are maintained using keys and integrity constraints.
Suitable for small applications.	Suitable for large, complex, and multi-user applications.

Q.2.b. With a neat diagram, discuss various DB system environment components (8M)



A **database** is a collection of logically related data that is organized and stored so that it can be easily accessed, managed, and updated. It represents some aspect of the real world and is designed to serve the information needs of an organization.

Example: A student database containing USN, Name, Branch, and Marks.

Component Modules of DBMS

A DBMS consists of several modules that work together to process queries, manage data, and maintain database consistency.

1. Query Processor

The query processor translates user queries into low-level instructions that the DBMS can understand and execute.

Components

- **DDL Interpreter:** Processes schema definition commands such as CREATE, ALTER, and DROP.
- **DML Compiler:** Converts SQL queries into low-level instructions.
- **Query Optimizer:** Selects the most efficient method for executing a query.
- **Query Evaluation Engine:** Executes the query and produces results.

2. Storage Manager

The storage manager acts as an interface between the query processor and the stored database. It manages storage, retrieval, and updating of data.

Components

a) Authorization and Integrity Manager

- Checks user access privileges.
- Enforces integrity constraints.

b) Transaction Manager

- Ensures correct execution of transactions.
- Maintains ACID properties.
- Handles concurrency control and recovery.

c) Buffer Manager

- Transfers data between main memory and disk.
- Improves performance by reducing disk accesses.

d) File Manager

- Manages allocation of disk space.
- Organizes database files on secondary storage.

3. Database Files

Database files contain the actual data stored in the database.

Example

- Student records
- Employee details
- Project information

4. Data Dictionary (DBMS Catalog)

The data dictionary stores metadata about the database such as:

- Table names
- Attribute names
- Data types
- Constraints
- User privileges

It helps the DBMS process queries efficiently.

Interaction of Components

1. Users submit SQL queries through application programs.
2. The **Query Processor** interprets and optimizes the query.
3. The **Query Evaluation Engine** sends requests to the Storage Manager.
4. The **Storage Manager** coordinates with:
    - Authorization & Integrity Manager
    - Transaction Manager
    - Buffer Manager
    - File Manager
5. Data is retrieved from database files.
6. The processed result is returned to the user.

Q.2.c. Draw an ER diagram, for ONLINE SHOPPING database with at least five entities. Specify primary key and participation constraints. (7M)

Entities and Attributes for the E-commerce Website
Entities and Attributes are defined below:

1. Product:

- **P-ID(Primary Key):** Unique identifier for each product.
- **Name:** Name of the product.
- **Price:** Price of the product.
- **Description:** Description of the product.

2. Pro-category:

- **Category - ID(Primary Key):** Unique identifier for each category.
- **Name:** Name of the category.

3. Order:

- **Order - No(Primary Key):** Unique identifier for each order.
- **Order - Amount:**
- **Order - Date:**

4. User:

- **User - ID(Primary Key):** Unique identifier for each user or customer.
- **Name:** Name of the user.
- **Email:** Email of the user.

5. Address:

- **Address - ID(Primary Key):** Unique identifier for each address
- **Country:** Country of the user.
- **State:** State of the user.
- **City:** City of the user.
- **Pin - code:** Pin code of the user.

6. Payment:

- **Payment - ID(Primary Key):** Unique identifier for each payment.
- **Method:** Payment method like UPI or Credit Card etc.
- **Amount:** Total amount paid by user.

7. Tracking Detail:

- **Tracking - ID(Primary Key):** Unique identifier for each tracking detail.
- **Status:** Tracking status like on the way or delivered etc.
- **Order - No(Foreign Key):** Reference to the order which need to be tracked.

8. Cart:

- **Cart - ID(Primary Key):** Unique identifier for each cart.
- **User - ID(Foreign Key):** Reference to the user.

Relationships Between These Entities

1. Product - Prod-Category Relationship:

- One product can belong to only one category.
- One category can have multiple products.
- So this is a **Many-to-one** relationship, showing that many products can belong to a single category.

2. Order-User Relationship:

- One user can place multiple orders.
- Each order is placed by exactly one user.
- This is a **one-to-many** relationship, showing that a user can place multiple orders, but each order is placed by exactly one user.

3. User-Address Relationship:

- One user can have multiple addresses.
- Each address belongs to exactly one user.
- This is a **one-to-many** relationship, indicating that a user can have multiple addresses associated with their account.

4. Tracking Detail - Order Relationship:

- One order can have multiple tracking details.
- Each tracking detail corresponds to exactly one order.
- This is **Many-to-one** relationships showing that an order can have one or multiple associated tracking details.

5. Product - Cart Relationship:

- One product can be added to multiple carts.
- Each cart can contain multiple products.
- This is **many-to-many** relationship.

6. User-Payment Relationship:

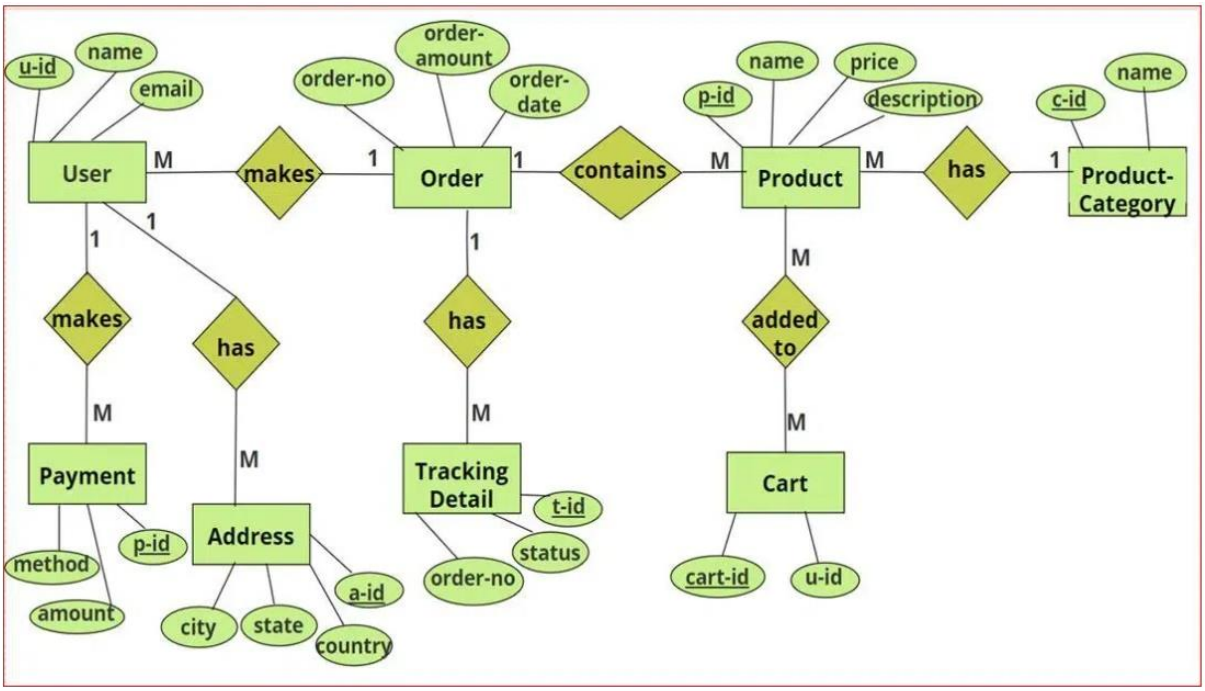
- One user can make multiple payments.
- Each payment is made by exactly one user.

- This is **one-to-many** relationship because each user can make multiple payments, and each payment is made by a single user.

7. Order-Product Relationship:

- One order can contains multiple products.
- Many products are get ordered in each order.
- So this is **one-to-many** relationship we can order multiple products on each order.

Representation of ER Diagram
Below is the ER diagram of large scale E-commerce platforms which meets all our requirements:



Module - 2

Q.3.a. Explain briefly domain, key, integrity and referential integrity constraints with necessary examples (10M)

1. Domain Constraint

A **Domain Constraint** specifies that the value of an attribute must belong to a predefined **domain**, i.e., a valid set of values, data type, range, or format. It ensures that only meaningful and valid values are stored in the database.

Example

Consider the relation:

STUDENT(USN, Name, Age, Branch)

USN	Name	Age	Branch
101	Ravi	20	CSE
102	Priya	19	ISE

Here:

- **Age** must be between **18 and 30**.
- **Branch** can be only **CSE, ISE, ECE, AIML**, etc.

Invalid entries such as:

Age = -5
Branch = 123

are rejected because they violate the domain constraint.

Importance

- Ensures only valid values are stored.
- Prevents invalid or meaningless data.
- Improves data accuracy and reliability.

2. Key Constraint

A **Key Constraint** ensures that every tuple (row) in a relation is **uniquely identifiable**. No two tuples can have the same value for a candidate key or primary key.

Example

Consider:

STUDENT

USN (Primary Key)	Name	Branch
101	Ravi	CSE
102	Priya	ISE

Here, **USN** is the **Primary Key**.

The following insertion is **not allowed**:

USN	Name	Branch
101	Arun	ECE

because **USN = 101** already exists.

Importance

- Eliminates duplicate records.
- Uniquely identifies each tuple.
- Helps establish relationships between tables.

3. Entity Integrity Constraint

The **Entity Integrity Constraint** states that the **Primary Key of a relation cannot be NULL** because every tuple must be uniquely identified.

If a primary key is NULL, the database cannot distinguish one record from another.

Example

STUDENT

USN (Primary Key)	Name
101	Ravi
102	Priya

Valid.

The following tuple is **invalid**:

USN	Name
NULL	Arun

because the primary key cannot be NULL.

Importance

- Ensures every record has a unique identity.
- Prevents unidentified tuples.
- Maintains data integrity.

4. Referential Integrity Constraint

The **Referential Integrity Constraint** ensures that a **Foreign Key** value in one relation either matches an existing **Primary Key** value in the referenced relation or is **NULL** (if permitted). It maintains consistency between related tables.

Example

DEPARTMENT

Dept_ID (Primary Key)	Dept_Name
D1	CSE
D2	ISE

STUDENT

USN	Name	Dept_ID (Foreign Key)
101	Ravi	D1
102	Priya	D2

Valid, because **D1** and **D2** exist in the DEPARTMENT table.

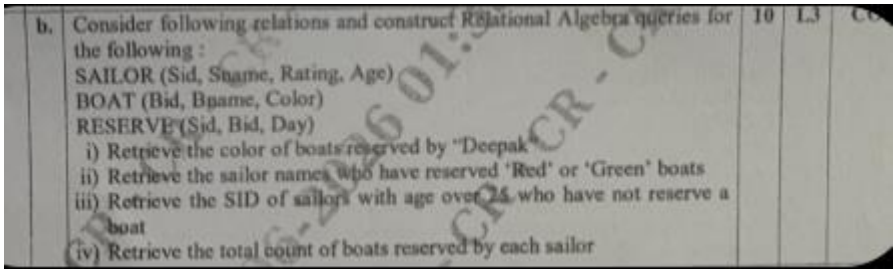
The following tuple is **invalid**:

USN	Name	Dept_ID
103	Arun	D5

because **D5** does not exist in the DEPARTMENT table.

Importance

- Maintains consistency between related tables.
- Prevents invalid foreign key values.
- Avoids orphan records.
- Preserves relationships among tables.



b. Consider following relations and construct Relational Algebra queries for the following : | 10 | L3 | CO
SAILOR (Sid, Sname, Rating, Age)
BOAT (Bid, Bname, Color)
RESERVE (Sid, Bid, Day)
i) Retrieve the color of boats reserved by "Deepak"
ii) Retrieve the sailor names who have reserved 'Red' or 'Green' boats
iii) Retrieve the SID of sailors with age over 25 who have not reserve a boat
iv) Retrieve the total count of boats reserved by each sailor

(i)

$$\pi_{Color}\left((\sigma_{Sname='Deepak'}(SAILOR)) \bowtie RESERVE \bowtie BOAT\right)$$

(ii)

$$\pi_{Sname}\left(SAILOR \bowtie RESERVE \bowtie \sigma_{Color='Red' \vee Color='Green'}(BOAT)\right)$$

(iii)

$$\pi_{Sid}(\sigma_{Age>25}(SAILOR)) - \pi_{Sid}(RESERVE)$$

(iv)

$$\gamma_{Sid,\ COUNT(Bid)\rightarrow TotalBoats}(RESERVE)$$

Q.4.a. Explain ER-to-Relational mapping algorithm with example (10M)

The **ER-to-Relational Mapping Algorithm** is a systematic procedure used to convert an **Entity-Relationship (ER) model** into a set of relational schemas (tables). Each entity, relationship, and attribute in the ER diagram is transformed into one or more relations while preserving the semantics and constraints of the original ER model.

Steps of ER-to-Relational Mapping Algorithm

Step 1: Mapping of Regular (Strong) Entity Types

For each strong entity type in the ER model, create a relation containing all simple attributes of that entity. Choose the primary key of the entity as the primary key of the relation.

Example

Entity:

EMPLOYEE(Emp_ID, Name, Address, Salary)

Relation:

```
EMPLOYEE(Emp_ID, Name, Address, Salary)
```

Primary Key:

```
Emp_ID
```

Step 2: Mapping of Weak Entity Types

For each weak entity, create a relation containing its attributes along with the primary key of the owner entity. The combination of the owner's key and the partial key forms the primary key of the weak entity relation.

Example

Weak Entity:

DEPENDENT(Dependent_Name, Age, Relationship)

Owner Entity:

EMPLOYEE(Emp_ID)

Relation:

```
DEPENDENT(Emp_ID, Dependent_Name, Age, Relationship)
```

Primary Key:

```
(Emp_ID, Dependent_Name)
```

Foreign Key:

```
Emp_ID references EMPLOYEE(Emp_ID)
```

Step 3: Mapping of Binary 1:1 Relationship Types

For a one-to-one relationship, include the primary key of one entity as a foreign key in the relation corresponding to the other entity. Any attributes of the relationship are also included.

Example

Relationship:

MANAGES between EMPLOYEE and DEPARTMENT

Relations:

```
EMPLOYEE(Emp_ID, Name, Salary)
```

```
DEPARTMENT(Dept_ID, Dept_Name, Manager_ID)
```

Here,

```
Manager_ID
```

is a foreign key referencing

```
EMPLOYEE(Emp_ID)
```

Step 4: Mapping of Binary 1:N Relationship Types

For a one-to-many relationship, add the primary key of the entity on the "one" side as a foreign key in the relation corresponding to the entity on the "many" side.

Example

Relationship:

WORKS_FOR between DEPARTMENT and EMPLOYEE

(One department has many employees)

Relations:

```
DEPARTMENT(Dept_ID, Dept_Name)

EMPLOYEE(Emp_ID, Name, Salary, Dept_ID)
```

Foreign Key:

```
Dept_ID references DEPARTMENT(Dept_ID)
```

Step 5: Mapping of Binary M:N Relationship Types

For a many-to-many relationship, create a new relation containing the primary keys of both participating entities. The combination of these keys forms the primary key of the new relation. Relationship attributes are also included.

Example

Relationship:

WORKS_ON between EMPLOYEE and PROJECT.

Attribute:

Hours

Relations:

```
EMPLOYEE(Emp_ID, Name)

PROJECT(Project_ID, Project_Name)

WORKS_ON(Emp_ID, Project_ID, Hours)
```

Primary Key:

```
(Emp_ID, Project_ID)
```

Foreign Keys:

```
Emp_ID references EMPLOYEE(Emp_ID)

Project_ID references PROJECT(Project_ID)
```

Step 6: Mapping of Multivalued Attributes

For every multivalued attribute, create a separate relation. Include the primary key of the original entity and the multivalued attribute.

Example

Suppose EMPLOYEE has a multivalued attribute:

Phone_No

Relation:

```
EMPLOYEE(Emp_ID, Name)

EMP_PHONE(Emp_ID, Phone_No)
```

Primary Key:

```
(Emp_ID, Phone_No)
```

Step 7: Mapping of N-ary Relationship Types

For an n-ary relationship, create a separate relation containing the primary keys of all participating entities. Include any attributes associated with the relationship.

Example

Relationship:

SUPPLY between

- SUPPLIER
- PART
- PROJECT

Relations:

```
SUPPLIER(Supplier_ID, Name)

PART(Part_ID, Part_Name)

PROJECT(Project_ID, Project_Name)

SUPPLY(Supplier_ID, Part_ID, Project_ID, Quantity)
```

Primary Key:

```
(Supplier_ID, Part_ID, Project_ID)
```

b.	Consider the following relations and construct Relations Algebra Queries. STUDENT(Sid, Sname, Major, GPA) FACULTY (Fid, Fname, Dept, Designation, Salary) COURSE (Cid, Cname, Fid) ENROLL (Cid, Sid, Grade) i) Retrieve the name of facilities who are receiving salary more than 34567.00 ii) List the name of all faculty members who teaches the course "BCS 403". iii) List all the departments having an average salary of above 45678.00 iv) List the names of students enrolled for the course "Mastering SGC".	10	L3	CO2

(i)

$$\pi_{Fname}(\sigma_{Salary>34567}(FACULTY))$$

(ii)

$$\pi_{Fname}(FACULTY \bowtie \sigma_{Cname='BCS\ 403'}(COURSE))$$

(iii)

$$\sigma_{AvgSalary>45678}(\gamma_{Dept,\ AVG(Salary)\rightarrow AvgSalary}(FACULTY))$$

(iv)

$$\pi_{Sname}(STUDENT \bowtie ENROLL \bowtie \sigma_{Cname='Mastering\ SGC'}(COURSE))$$

Module-3

Q.5.a. Define Functional dependency. Explain 1NF and 2NF with example (10M)

A **Functional Dependency (FD)** is a constraint between two sets of attributes in a relation. An attribute **Y** is said to be functionally dependent on attribute **X** if each value of **X** is associated with exactly one value of **Y**.

It is represented as:

X→Y

which means **X determines Y**.

Example

Consider the relation:

STUDENT(USN, Name, Branch)

USN	Name	Branch
101	Ravi	CSE
102	Priya	ISE
103	Arun	ECE

Here,

USN → Name, Branch

because each **USN** uniquely identifies one student's **Name** and **Branch**.

Need for Functional Dependency

- Identifies relationships among attributes.
- Helps determine candidate keys.
- Forms the basis for normalization.
- Reduces redundancy.
- Eliminates update anomalies.
- Improves database consistency.

First Normal Form (1NF)

A relation is said to be in **First Normal Form (1NF)** if all its attributes contain only atomic (indivisible) values and each attribute contains a single value. In other words, repeating groups and multivalued attributes are not permitted in a relation satisfying 1NF.

The primary objective of 1NF is to eliminate repeating groups and ensure that every field contains only one value.

Example

Consider the relation:

STUDENT

SID	Name	Subjects
1	Ravi	DBMS, OS
2	Priya	CN

3	Arjun	DBMS, CN

The attribute **Subjects** contains multiple values and hence the relation is not in 1NF.

To convert it into 1NF, each subject is stored separately.

STUDENT

SID	Name	Subject
1	Ravi	DBMS
1	Ravi	OS
2	Priya	CN
3	Arjun	DBMS
3	Arjun	CN

Now each attribute contains only atomic values. Therefore, the relation is in First Normal Form.

Characteristics of 1NF

- Eliminates repeating groups.
- Ensures atomicity of attribute values.
- Removes multivalued attributes.
- Forms the basis for higher normal forms.

Second Normal Form (2NF)

A relation is said to be in **Second Normal Form (2NF)** if it is already in 1NF and every non-prime attribute is fully functionally dependent on the entire primary key. That is, no non-key attribute should depend on only a part of a composite key.

The main objective of 2NF is to eliminate partial dependencies.

Example

Consider the relation:

ENROLLMENT

SID	CourseID	StudentName	CourseName
101	C1	Ravi	DBMS
101	C2	Ravi	OS
102	C1	Priya	DBMS

The composite primary key is:

$$(SID, CourseID)$$

Functional dependencies are:

$$SID \rightarrow StudentName$$

$$CourseID \rightarrow CourseName$$

Since StudentName depends only on SID and CourseName depends only on CourseID, partial dependencies exist. Therefore, the relation is not in 2NF.

Decomposition

STUDENT

SID	StudentName
101	Ravi
102	Priya

COURSE

CourseID	CourseName
C1	DBMS
C2	OS

ENROLLMENT

SID	CourseID
101	C1
101	C2
102	C1

Now all non-key attributes depend on the whole primary key, and hence the relations are in Second Normal Form.

Characteristics of 2NF

- Relation must already be in 1NF.
- Eliminates partial functional dependencies.
- Removes redundant storage of data.
- Reduces update anomalies.

Q.5.b. Explain occurrences of types of anomalies while performing SQL operations (10M)

An **anomaly** is an undesirable situation that occurs in a database due to poor database design and data redundancy. Anomalies lead to **data inconsistency, duplication, and difficulty in maintaining the database**. They commonly occur while performing **INSERT, UPDATE, and DELETE** operations on database tables.

The three major types of anomalies are:

1. Insertion Anomaly
2. Update (Modification) Anomaly
3. Deletion Anomaly

1. Insertion Anomaly

An **Insertion Anomaly** occurs when it is impossible to insert data into a table without inserting some unnecessary or unrelated information. This happens because multiple facts are stored in the same relation.

Example

Consider the relation:

Student_ID	Student_Name	Course	Faculty
101	Ravi	DBMS	Dr. Rao
102	Priya	Python	Dr. Kumar

Suppose a new course **AI** has been introduced, but no student has enrolled in it yet.

It is not possible to insert:

Course	Faculty
AI	Dr. Sharma

because the table also requires **Student_ID** and **Student_Name**.

Occurrence

Occurs during the **INSERT** operation.

Problems

- Cannot insert independent information.
- Forces insertion of NULL or duplicate values.
- Reduces database flexibility.

2. Update (Modification) Anomaly

An **Update Anomaly** occurs when the same information is stored in multiple rows and must be updated everywhere. If some rows are updated and others are not, the database becomes inconsistent.

Example

Student_ID	Student_Name	Course	Faculty
101	Ravi	DBMS	Dr. Rao
102	Priya	DBMS	Dr. Rao
103	Arun	DBMS	Dr. Rao

Suppose **Dr. Rao** is replaced by **Dr. Mehta**.

Every row containing **Dr. Rao** must be updated.

If only two rows are updated:

Student_ID	Faculty
101	Dr. Mehta
102	Dr. Mehta
103	Dr. Rao

The database now contains inconsistent information.

Occurrence

Occurs during the **UPDATE** operation.

Problems

- Data inconsistency.
- Increased maintenance effort.
- Same information must be updated multiple times.

3. Deletion Anomaly

A **Deletion Anomaly** occurs when deleting a record unintentionally removes other valuable information from the database.

Example

Student_ID	Student_Name	Course	Faculty
101	Ravi	DBMS	Dr. Rao
102	Priya	Python	Dr. Kumar

Suppose Ravi is the only student enrolled in **DBMS**.

If Ravi's record is deleted:

```
DELETE FROM STUDENT
WHERE Student_ID = 101;
```

the information that **Dr. Rao teaches DBMS** is also lost.

Occurrence

Occurs during the **DELETE** operation.

Problems

- Loss of important information.
- Accidental deletion of related data.
- Difficulty in recovering deleted information.

Anomalies can be minimized by applying **Normalization**.

- **1NF** removes repeating groups and multivalued attributes.
- **2NF** removes partial dependencies.
- **3NF** removes transitive dependencies.

Normalization reduces redundancy and ensures data integrity.

Q.6.a. What is multivalued dependency? Explain fourth normal form at examples (10M)

A **Multivalued Dependency** exists when, for a single value of an attribute, there are **multiple independent values** of another attribute.

It is represented as:

$A \twoheadrightarrow B$

which is read as:

"A multidetermines B."

This means that for each value of **A**, there can be several values of **B**, and these values are independent of other attributes.

Example of Multivalued Dependency

Consider the relation:

STUDENT(Student_ID, Hobby, Language)

Student_ID	Hobby	Language
101	Cricket	English
101	Cricket	Hindi
101	Music	English
101	Music	Hindi

Here,

- Student **101** has two hobbies: **Cricket** and **Music**.
- Student **101** knows two languages: **English** and **Hindi**.

The hobbies and languages are **independent** of each other.

Therefore,

Student_ID ↠ Hobby

and

Student_ID ↠ Language

This creates unnecessary combinations of hobbies and languages, leading to redundancy.

Problems Caused by Multivalued Dependency

- Data redundancy.
- Wastage of storage space.
- Insertion anomalies.
- Deletion anomalies.
- Update anomalies.
- Difficult database maintenance.

Fourth Normal Form (4NF)

A relation is said to be in **Fourth Normal Form (4NF)** if:

1. It is already in **Boyce-Codd Normal Form (BCNF)**.
2. It contains **no non-trivial multivalued dependencies**.
3. Every multivalued dependency is based on a **candidate key**.

In simple terms, **4NF removes independent multivalued facts from a relation.**

Example (Relation Not in 4NF)

Consider the relation:

STUDENT(Student_ID, Hobby, Language)

Student_ID	Hobby	Language
101	Cricket	English
101	Cricket	Hindi
101	Music	English
101	Music	Hindi

Here,

Student_ID ↠ Hobby
Student_ID ↠ Language

Since both multivalued dependencies exist independently, the relation is **not in 4NF**.

Conversion into 4NF

Decompose the relation into two relations.

STUDENT_HOBBY

Student_ID	Hobby
101	Cricket
101	Music

STUDENT_LANGUAGE

Student_ID	Language
101	English
101	Hindi

Now,

- There are no unnecessary combinations.
- Data redundancy is removed.
- Each relation stores only one multivalued fact.

Hence, both relations are in **Fourth Normal Form (4NF).**

Characteristics of 4NF

- Must satisfy BCNF.
- Removes multivalued dependencies.
- Eliminates unnecessary repetition of data.
- Improves consistency.
- Reduces insertion, deletion, and update anomalies.

Advantages of 4NF

- Eliminates redundancy caused by multivalued dependencies.
- Prevents update anomalies.
- Simplifies database maintenance.
- Improves storage efficiency.
- Maintains data integrity.

Q.6.b. Describe the six clauses in the syntax of a SQL relational query. Show what type of constructs can be specified in each of the six clauses (10M)

1. SELECT Clause

The **SELECT** clause specifies the attributes (columns) to be displayed in the query result. It determines **what data** should be retrieved from the database.

Constructs Specified

- Column names
- Arithmetic expressions
- Aggregate functions
- DISTINCT keyword
- Aliases (AS)

Syntax

```
SELECT column1, column2
FROM table_name;
```

Example

```
SELECT Name, Salary
FROM EMPLOYEE;
```

This query displays the **Name** and **Salary** of employees.

2. FROM Clause

The **FROM** clause specifies the table or tables from which data is to be retrieved.

It can also specify joins between multiple tables.

Constructs Specified

- Single table
- Multiple tables
- JOIN operations
- Table aliases

Syntax

```
FROM table_name
```

Example

```
SELECT *
FROM EMPLOYEE;
```

3. WHERE Clause

The **WHERE** clause specifies conditions to filter rows before they are displayed. Only records satisfying the condition are selected.

Constructs Specified

- Comparison operators (=, >, <, >=, <=, <>)
- Logical operators (AND, OR, NOT)
- BETWEEN
- IN

- LIKE
- IS NULL
- Subqueries

Syntax

```
WHERE condition
```

Example

```
SELECT Name
FROM EMPLOYEE
WHERE Salary > 50000;
```

This query displays employees whose salary is greater than ₹50,000.

4. GROUP BY Clause

The **GROUP BY** clause groups rows having the same values in specified columns. It is mainly used with aggregate functions.

Constructs Specified

- Grouping attributes
- Aggregate functions
  - COUNT()
  - SUM()
  - AVG()
  - MAX()
  - MIN()

Syntax

```
GROUP BY column_name;
```

Example

```
SELECT DeptNo, AVG(Salary)
FROM EMPLOYEE
GROUP BY DeptNo;
```

This query calculates the average salary of employees in each department.

5. HAVING Clause

The **HAVING** clause specifies conditions on groups created by the GROUP BY clause. It filters groups after aggregation.

Constructs Specified

- Aggregate conditions
- COUNT()
- SUM()

- AVG()
- MAX()
- MIN()

Syntax

```
HAVING condition
```

Example

```
SELECT DeptNo, COUNT(*)
FROM EMPLOYEE
GROUP BY DeptNo
HAVING COUNT(*) > 5;
```

This query displays departments having more than five employees.

6. ORDER BY Clause

The **ORDER BY** clause sorts the query result in ascending or descending order.

By default, sorting is in **ascending (ASC)** order.

Constructs Specified

- ASC (Ascending)
- DESC (Descending)
- Multiple columns

Syntax

```
ORDER BY column_name ASC|DESC;
```

Example

```
SELECT Name, Salary
FROM EMPLOYEE
ORDER BY Salary DESC;
```

This query displays employees in descending order of salary.

Q.7	a.	Consider following relation, construct SQL queries. EMPLOYEE (Fname, Lname, SSN, Address, Phone, Dno) DEPARTMENT (Dname, Dnumber, Mgr, SSn) PROJECT (Pname, Pnumber, Plocations) DEPENDENT (Name, Age, Relationship, Essn) WORKS ON (Pnum, EmpSSN, Hours) i) Make a list of all project number for projects that involve an employee whose last name is "Shetty: as a worker or as a manager of the department that controls the project ii) Retrieve name and address of all employees who work for the "Research" Department iii) Retrieve the name of each employees who works on all the project controlled by department number 555 iv) Find total number of employees, maximum salary, minimum salary, and average salary among employees who works for the "HR" department.	10	L3	CO3

(i)

```
SELECT P.Pnumber
FROM PROJECT P, DEPARTMENT D, EMPLOYEE E
WHERE P.Dnum = D.Dnumber
AND D.Mgr = E.SSN
AND E.Lname = 'Shetty'

UNION

SELECT W.Pnum
FROM WORKS_ON W, EMPLOYEE E
WHERE W.EmpSSN = E.SSN
AND E.Lname = 'Shetty';
```

(ii)

```
SELECT E.Fname, E.Lname, E.Address
FROM EMPLOYEE E, DEPARTMENT D
WHERE E.Dno = D.Dnumber
AND D.Dname = 'Research';
```

(iii)

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE E
WHERE NOT EXISTS
(
    SELECT P.Pnumber
    FROM PROJECT P
    WHERE P.Dnum = 555
    AND NOT EXISTS
    (
        SELECT *
        FROM WORKS_ON W
        WHERE W.EmpSSN = E.SSN
        AND W.Pnum = P.Pnumber
    )
);
```

(iv)

```
SELECT COUNT(*) AS Total_Employees,
       MAX(Salary) AS Maximum_Salary,
       MIN(Salary) AS Minimum_Salary,
       AVG(Salary) AS Average_Salary
FROM EMPLOYEE E, DEPARTMENT D
WHERE E.Dno = D.Dnumber
AND D.Dname = 'HR';
```

Q.7.b. Define Transaction processing. Explain ACID properties of transactions(10M)

Transaction Processing is the process of executing a set of database operations as a single logical unit of work while maintaining the correctness and consistency of the database.

A transaction has two possible outcomes:

- **COMMIT** – All operations are completed successfully, and the changes are permanently saved.
- **ROLLBACK (ABORT)** – If an error occurs, all changes made by the transaction are undone, restoring the database to its previous consistent state.

Example

Consider a bank transaction that transfers ₹5,000 from Account A to Account B.

```
Read(A)
A = A - 5000
Write(A)

Read(B)
B = B + 5000
Write(B)

COMMIT
```

If the system fails after deducting money from Account A but before adding it to Account B, the transaction is **rolled back**, ensuring that no partial update occurs.

ACID Properties of Transactions

The correctness of transaction processing is ensured by the **ACID properties**.

ACID stands for:

- **A – Atomicity**
- **C – Consistency**
- **I – Isolation**
- **D – Durability**

1. Atomicity

Atomicity means that a transaction is treated as a single indivisible unit. Either **all operations are executed successfully**, or **none of them are executed**.

Example

During a bank transfer:

- Debit ₹5,000 from Account A.
- Credit ₹5,000 to Account B.

If the credit operation fails after the debit operation, the debit is also cancelled using **ROLLBACK**.

Importance

- Prevents partial updates.
- Maintains database consistency.
- Ensures "all-or-nothing" execution.

2. Consistency

Consistency ensures that a transaction transforms the database from **one valid state to another valid state**, without violating integrity constraints.

Example

Suppose the total balance in two accounts is ₹30,000.

Before transaction:

```
A = ₹20,000
B = ₹10,000
Total = ₹30,000
```

After transferring ₹5,000:

```
A = ₹15,000
B = ₹15,000
Total = ₹30,000
```

The total balance remains unchanged, so the database remains consistent.

Importance

- Preserves data integrity.
- Enforces database constraints.
- Prevents invalid data.

3. Isolation

Isolation ensures that concurrent transactions do not interfere with each other. Each transaction executes as if it is the only transaction running in the system.

Example

Transaction T1 updates an employee's salary.

At the same time, Transaction T2 reads the salary.

T2 should not read the updated value until T1 commits.

Importance

- Prevents dirty reads.
- Prevents lost updates.
- Prevents unrepeatable reads.
- Ensures serializable execution.

4. Durability

Durability ensures that once a transaction is **committed**, its changes become permanent and are not lost even if the system crashes or there is a power failure.

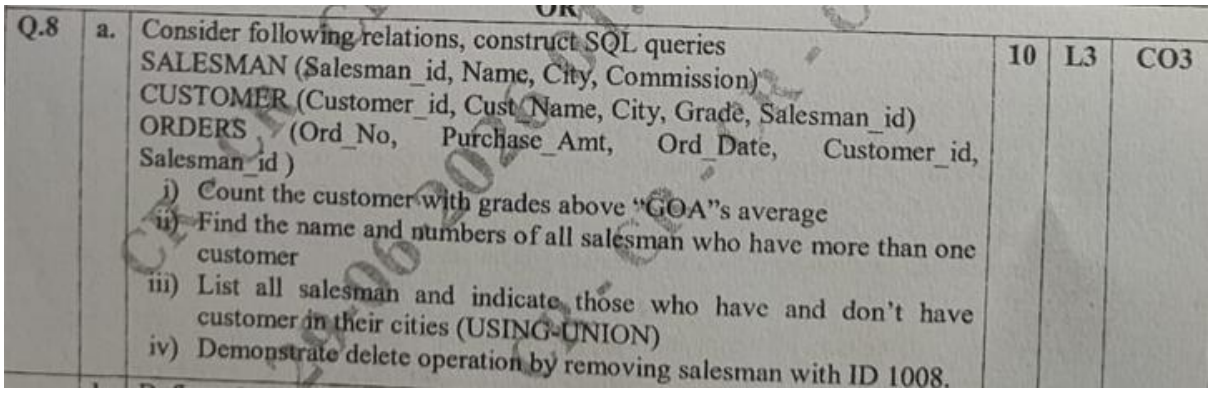
Example

A customer successfully transfers money, and the transaction is committed.

Even if the system crashes immediately afterward, the updated account balances remain stored in the database.

Importance

- Prevents data loss after commit.
- Ensures permanent storage of committed transactions.
- Supports recovery after system failure.



Q.8	a.	OR Consider following relations, construct SQL queries SALESMAN (Salesman_id, Name, City, Commission) CUSTOMER (Customer_id, Cust_Name, City, Grade, Salesman_id) ORDERS (Ord_No, Purchase_Amt, Ord_Date, Customer_id, Salesman_id) i) Count the customer with grades above "GOA"s average ii) Find the name and numbers of all salesman who have more than one customer iii) List all salesman and indicate those who have and don't have customer in their cities (USING-UNION) iv) Demonstrate delete operation by removing salesman with ID 1008.	10	L3	CO3

(i)

```
SELECT COUNT(*)
FROM CUSTOMER
WHERE Grade >
(
    SELECT AVG(Grade)
    FROM CUSTOMER
```

```
    WHERE City = 'GOA'
);
```

(ii)

```
SELECT S.Salesman_id, S.Name
FROM SALESMAN S, CUSTOMER C
WHERE S.Salesman_id = C.Salesman_id
GROUP BY S.Salesman_id, S.Name
HAVING COUNT(C.Customer_id) > 1;
```

(iii)

```
SELECT S.Name, 'Has Customer in City' AS Status
FROM SALESMAN S, CUSTOMER C
WHERE S.City = C.City

UNION

SELECT S.Name, 'No Customer in City' AS Status
FROM SALESMAN S
WHERE S.City NOT IN
(
    SELECT City
    FROM CUSTOMER
);
```

(iv)

```
DELETE FROM SALESMAN
WHERE Salesman_id = 1008;
```

Q.8.b. Define Serializability, Elaborate characterizing schedules based on serializability (10M)

A **schedule** is the chronological sequence in which the operations (Read, Write, Commit, Abort) of two or more transactions are executed while preserving the order of operations within each transaction.

Types of Schedules

- **Serial Schedule**
- **Non-Serial Schedule**

Example of a Serial Schedule

```
T1: Read(A)
    Write(A)
    Commit

T2: Read(B)
    Write(B)
    Commit
```

Here, T1 completes before T2 starts.

Serializability

Serializability is a property of a schedule that ensures the concurrent execution of transactions produces the same result as some serial execution of those transactions.

It is the primary correctness criterion for concurrent transactions.

Need for Serializability

- Maintains database consistency.
- Prevents concurrency problems.
- Ensures correct execution of transactions.
- Improves database reliability.
- Supports concurrent transaction processing.

Characterizing Schedules Based on Serializability

Schedules are mainly classified into two types based on serializability:

1. Conflict Serializability
2. View Serializability

1. Conflict Serializability

A schedule is **Conflict Serializable** if it can be transformed into a serial schedule by swapping **non-conflicting operations** without changing the result.

Two operations are said to **conflict** if:

- They belong to different transactions.
- They access the same data item.
- At least one of them is a write operation.

Conflicting Operations

- Read(X) – Write(X)
- Write(X) – Read(X)
- Write(X) – Write(X)

Non-Conflicting Operations

- Read(X) – Read(X)
- Operations on different data items.

Example

Transactions:

```
T1                T2

Read(A)
Write(A)

                  Read(B)
                  Write(B)
```

Since T1 and T2 operate on different data items, their operations can be interchanged.

Hence, the schedule is **Conflict Serializable**.

Precedence (Serialization) Graph

A **precedence graph** is used to test conflict serializability.

- Each transaction is represented as a node.
- An edge **Ti → Tj** indicates that Ti must execute before Tj because of conflicting operations.

Rule

- **No cycle** → Conflict Serializable.
- **Cycle present** → Not Conflict Serializable.

Example:

```
T1  --------->  T2
```

No cycle ⇒ Serializable.

2. View Serializability

A schedule is **View Serializable** if it produces the same final result as a serial schedule, even if it is not conflict serializable.

Two schedules are view equivalent if:

- They read the same initial values.
- Every read operation reads the same value in both schedules.
- The final write on every data item is performed by the same transaction.

Example

```
T1                T2

Write(A)
```

```
                    Write(A)
```

Although the writes may conflict, if the final value of **A** is the same as in some serial schedule, the schedule is **View Serializable**.

Module - 5

Q.9.a. Elaborate 2PL techniques for concurrency control (10M)

Two-Phase Locking (2PL) is a lock-based concurrency control protocol used to ensure the serializability and consistency of transactions in a database system. In this protocol, a transaction must acquire appropriate locks before accessing data items and release locks according to specific rules.

The name **Two-Phase Locking** arises because the execution of every transaction is divided into two distinct phases:

1. **Growing Phase**
2. **Shrinking Phase**

By following these two phases, transactions can execute concurrently without violating database consistency.

Need for Two-Phase Locking

Two-phase locking is used to:

- Ensure serializability of concurrent transactions.
- Maintain database consistency.
- Prevent lost updates and dirty reads.
- Coordinate simultaneous access to shared data.
- Enforce isolation among transactions.

Phases of Two-Phase Locking

1. Growing Phase

Definition

The growing phase is the phase during which a transaction acquires locks on data items but is not allowed to release any lock.

In this phase:

- New locks can be acquired.
- No lock can be released.

The transaction continues to obtain all required locks until it reaches a point called the **lock point**.

Example

Transaction T1:

```
Lock-X(A)
Read(A)
Write(A)

Lock-X(B)
Read(B)
Write(B)
```

During this period, T1 acquires locks on A and B but does not release any lock.

Characteristics

- Only lock acquisition is allowed.
- Lock release is prohibited.
- Ends when the transaction obtains its final lock.

2. Shrinking Phase

Definition

The shrinking phase begins after the transaction releases its first lock. During this phase, the transaction releases locks but is not allowed to acquire any new lock.

In this phase:

- Locks are released.
- No new lock can be acquired.

Example

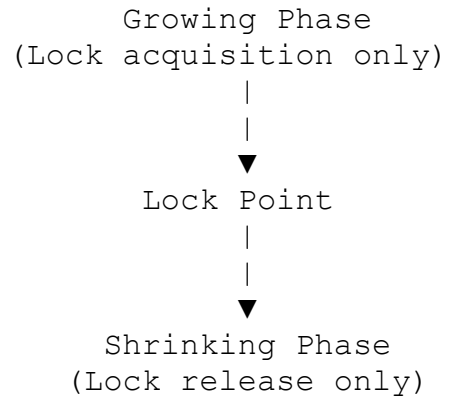
```
Unlock(A)
Unlock(B)
```

Once a transaction starts releasing locks, it enters the shrinking phase and cannot request additional locks.

Characteristics

- Only lock release is allowed.
- Acquisition of new locks is prohibited.
- Continues until all locks are released.

Diagram of Two-Phase Locking



Example of Two-Phase Locking

Consider transaction T1:

```
Lock-X(A)
Read(A)
Write(A)

Lock-X(B)
Read(B)
Write(B)

Unlock(B)
Unlock(A)
```

Here,

- Acquiring locks on A and B represents the **Growing Phase**.
- Releasing locks on B and A represents the **Shrinking Phase**.

Hence, the transaction follows the Two-Phase Locking protocol.

Types of Two-Phase Locking

1. Basic Two-Phase Locking

In Basic 2PL:

- Transactions follow growing and shrinking phases.
- Locks may be released before the transaction commits.
- Guarantees conflict serializability.

Disadvantage

- Deadlocks may occur.

2. Conservative (Static) Two-Phase Locking

In Conservative 2PL:

- A transaction acquires all required locks before execution begins.
- If all locks are unavailable, the transaction waits.

Advantages

- Eliminates deadlocks.

Disadvantages

- Reduced concurrency.
- Requires prior knowledge of all required data items.

3. Strict Two-Phase Locking

In Strict 2PL:

- Exclusive locks are held until the transaction commits or aborts.
- Locks are released only after completion of the transaction.

Advantages

- Prevents cascading rollbacks.
- Ensures recoverable schedules.
- Widely used in commercial DBMSs.

Disadvantages

- Deadlocks are still possible.

4. Rigorous Two-Phase Locking

In Rigorous 2PL:

- Both shared and exclusive locks are retained until the transaction commits or aborts.

Advantages

- Produces strict and serializable schedules.
- Simplifies recovery mechanisms.

Disadvantages

- Reduces concurrency.

Advantages of Two-Phase Locking

- Ensures serializability.
- Preserves database consistency.
- Supports concurrent execution of transactions.
- Prevents anomalies such as lost updates and dirty reads.
- Simple and widely implemented.

Disadvantages of Two-Phase Locking

- May lead to deadlocks.
- Transactions may experience waiting delays.
- Lock management introduces overhead.
- Reduced concurrency in strict forms of 2PL.

Q.9.b. Explain optimistic concurrency control techniques (10M)

Optimistic Concurrency Control (OCC) is a concurrency control technique in which transactions execute freely without locking data items. At the end of execution, a validation test checks whether the transaction can be safely committed. If validation fails, the transaction is aborted and restarted.

Working of Optimistic Concurrency Control

OCC divides the execution of a transaction into **three phases**:

1. Read Phase

- The transaction reads the required data items from the database.
- All updates are made on **local copies** (private workspace).
- The actual database is **not modified** during this phase.
- No locks are acquired.

Example

Transaction **T1** reads account balance and updates it in its local memory without changing the database.

2. Validation Phase

- Before committing, the transaction is checked for conflicts with other concurrent transactions.
- The DBMS verifies whether the transaction can be serialized without violating consistency.
- If validation succeeds, the transaction proceeds to the write phase.
- Otherwise, the transaction is rolled back and restarted.

Validation Checks

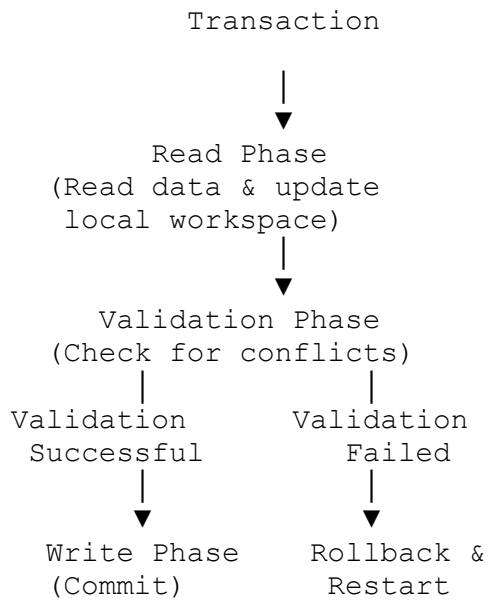
A transaction **Tj** is allowed to commit if one of the following conditions is satisfied for every earlier transaction **Ti**:

- **Ti completes before Tj starts its validation phase.**
- **Ti finishes its write phase before Tj begins its write phase**, and the read set of **Tj** does not intersect with the write set of **Ti**.

3. Write Phase

- If validation is successful, all updates from the local workspace are copied to the database.
- The transaction is committed permanently.
- If validation fails, all local changes are discarded, and the transaction is restarted.

Diagram of OCC



Example

Consider two transactions:

T1

```
Read(A)
A = A + 100
```

T2

```
Read(A)
A = A - 50
```

- Both transactions execute simultaneously without locking **A**.
- During validation:
  - If no conflict exists, both transactions commit.
  - If a conflict is detected (both modifying **A**), one transaction is aborted and restarted.

Advantages of Optimistic Concurrency Control

- No locking overhead.
- Eliminates deadlocks.
- High concurrency among transactions.
- Better performance when conflicts are rare.
- Suitable for read-intensive applications.
- Transactions do not wait for locks.

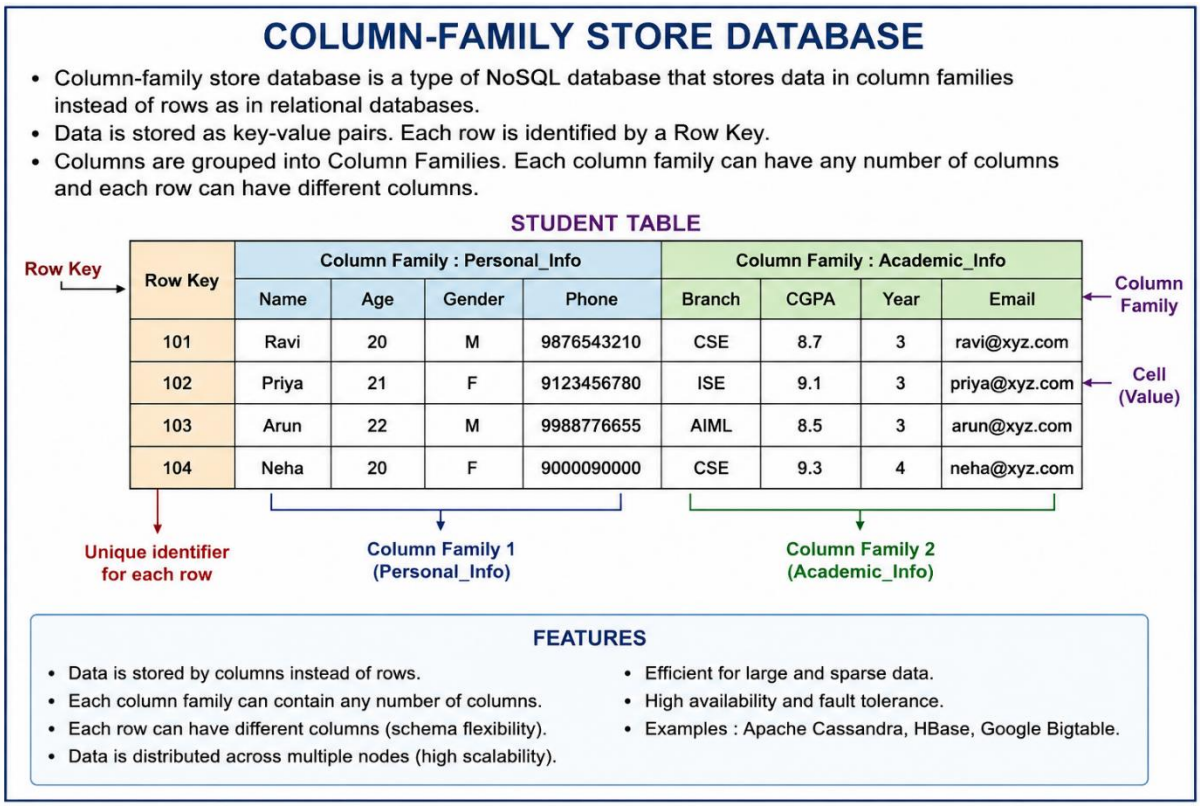
Disadvantages of Optimistic Concurrency Control

- Transactions may be rolled back frequently if conflicts are common.
- Validation phase increases processing overhead.
- Not suitable for systems with frequent updates.
- Wastes computation when aborted transactions are restarted.

Applications

- Online reservation systems.
- Web applications with many read operations.
- Banking systems with low transaction conflicts.
- Distributed databases.
- Decision support and reporting systems.

Q.10.a. Explain Column-Family store database with a neat diagram(10M)



The diagram consists of the following components:

1. Row Key

- Every row is identified by a unique **Row Key**.
- It acts as the primary identifier for retrieving records quickly.

Example:

```
101
102
103
104
```

2. Column Family

A **Column Family** is a logical grouping of related columns.

In the diagram:

- **Personal_Info**
  - Name
  - Age
  - Gender
  - Phone
- **Academic_Info**
  - Branch
  - CGPA
  - Year
  - Email

Each column family stores similar types of information together.

3. Columns

Each column stores one attribute and its value.

For example,

Under **Personal_Info**:

- Name = Ravi
- Age = 20
- Gender = M
- Phone = 9876543210

Under **Academic_Info**:

- Branch = CSE
- CGPA = 8.7
- Year = 3

4. Cell (Value)

The intersection of a **row** and a **column** is called a **cell**, which stores the actual data value.

Example:

```
Row Key = 101
Column = CGPA
Value = 8.7
```

Working of Column-Family Database

1. Each record is identified using a **Row Key**.
2. Related attributes are grouped into **Column Families**.
3. Each row can contain different columns depending on the available data.
4. Data is distributed across multiple servers for scalability.
5. Only the required column family is accessed during query execution, improving performance.

Q.10.b. Explain in detail CAP Theorem(10M)

The **CAP Theorem**, also known as **Brewer's Theorem**, states that a distributed database system can provide at most two out of the following three properties simultaneously:

1. **Consistency (C)**
2. **Availability (A)**
3. **Partition Tolerance (P)**

Thus, it is impossible for a distributed system to guarantee all three properties at the same time. When a network partition occurs, the system must choose between consistency and availability.

Components of CAP Theorem

1. Consistency (C)

Consistency ensures that all nodes in the distributed system see the same data at the same time. Whenever data is updated, all users receive the most recent value.

Example:
If a user updates his account balance, every node in the system should immediately reflect the updated balance.

2. Availability (A)

Availability guarantees that every request receives a response, even if some nodes in the system have failed. The system remains operational and accessible to users.

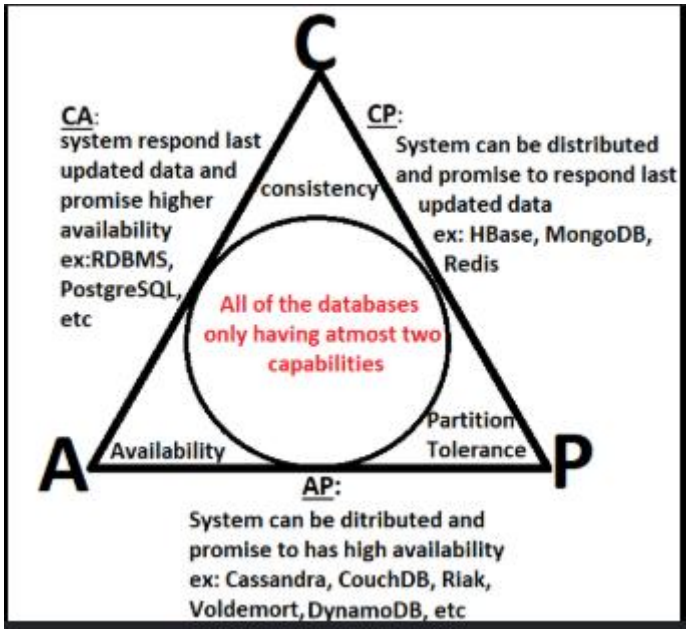
Example:
A web application should continue serving user requests even if one of its servers becomes unavailable.

3. Partition Tolerance (P)

Partition tolerance means that the system continues to function despite communication failures or network partitions between nodes.

Example:
If communication between two data centers is interrupted, the database should continue operating without complete failure.

Diagram of CAP Theorem



Possible Combinations

1. CP Systems (Consistency + Partition Tolerance)

- Maintain data consistency.
- Continue functioning during network partitions.
- May sacrifice availability.

Example: HBase, MongoDB (in certain configurations).

2. AP Systems (Availability + Partition Tolerance)

- Always provide responses to users.
- Continue operating during partitions.
- May temporarily sacrifice consistency.

Example: Cassandra, DynamoDB.

3. CA Systems (Consistency + Availability)

- Provide consistent and available data.
- Cannot tolerate network partitions.
- Suitable mainly for centralized databases.

Example: Traditional Relational Database Management Systems.