CBCS SCHEME

USN | | | | | | | | | |

BCS755A

Seventh Semester B.E./B.Tech. Degree Examination, Dec.2025/Jan.2026
Introduction to DBMS

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.

		Module – 1	M	L	C
Q.1	a.	Define Database. List and briefly explain the advantages of using DBMS approach.	10	L2	CO1
	b.	Define DBMS. Discuss the characteristics of the database approach.	5	L2	CO1
	c.	With neat diagram, explain the three schema architecture.	5	L2	CO1
		OR			
Q.2	a.	Explain the component module of DBMS and their interactions with the help of neat diagram.	10	L2	CO1
	b.	Define the following terms: i) Weak entity ii) DBMS catalog iii) Value sets iv) Cardinality ratio v) Degree of a relationship	10	L2	CO1
		Module – 2			
Q.3	a.	Build an ER diagram of company system taking into account at least four entities. Indicate all keys, constraints and assumptions that are made.	10	L3	CO2
	b.	With an example, explain the steps of ER to relational mapping algorithm.	10	L2	CO2
		OR			
Q.4	a.	Build an ER diagram for university database by considering atleast 5 entities.	10	L3	CO2
	b.	Illustrate specialization and generalization with example.	10	L2	CO2

		Module – 3			
Q.5	**a.**	Explain about relational model constraints.	**10**	**L2**	**CO3**
	b.	By refereeing the following database schema: EMPLOYEE (Fname, Minit, Lname, SSN, Bdate, Address, Sex, Salary, Sup-SSN, Dno) Department (Dname, Dnumber, Mgr-SSN, Mgr-start-date) Dept-Locations (Dnumber, Dlocation) Project (Pname, Pnumber, Plocation, Dnum) Works – on (Essn, Pno, Hours) Dependent (Essn, Dependent-name, sex, Bdate, Relationship) Show the relational algebra expressions for the following queries. i) Retrieve the name and address of all employees who work for the 'Research' department. ii) Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' either as a worker or as a manager of the department that controls the project. iii) List the names of managers who have at least one dependent. iv) Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary. v) For each project, retrieve the project number, the project name, and the number of employees who work on that project.	**10**	**L3**	**CO3**
		OR			
Q.6	**a.**	Explain how the basic operations deal with constraints violations with example.	**10**	**L2**	**CO3**
	b.	Explain Division operation with example.	**10**	**L2**	**CO3**
		Module – 4			
Q.7	**a.**	Explain INSERT, DELETE, UPDATE statements in SQL taking suitable examples.	**10**	**L2**	**CO3**
	b.	What is Normalization? Explain 1NF, 2NF and 3NF with examples.	**10**	**L2**	**CO4**

		OR			
Q.8	a.	List and explain informal design guidelines for relation schemas.	10	L2	CO4
	b.	By refereeing the following database schema: Employee (Fname, Minit, Lname, SSN, Bdate, Address, Sex, Salary, Sup-SSN, Dno) Department (Dname, Dnumber, Mgr-SSN, Mgr-start-date) Dept-Locations (Dnumber, Dlocations) Project (Pname, Pnumber, Plocation, Dnum) Works-on (Essn, Pno, Hours) Dependent (Essn, Dependent-Name, Sex, Bdate, Relationship) Write queries in SQL i) Retrieve all the employee names who are working for department number 5. ii) Retrieve all the projects which are controlled by department number 4 iii) Retrieve the names of employees who have no dependents iv) Retrieve the employee name who is working on all the projects in which 'John Smith' works on. v) Retrieve all the project numbers along with number of employee working on each project.	10	L3	CO3
		Module – 5			
Q.9	a.	How are triggers and assertions defined in SQL? Explain.	10	L2	CO3
	b.	Discuss the Two-Phase locking techniques for concurrency control.	10	L2	CO5
		OR			
Q.10	a.	Explain views in SQL with example.	10	L2	CO3
	b.	Explain validation concurrency control techniques in databases.	10	L2	CO5

* * * * *

		Module – 1	M	L	C
Q.1	a.	Define Database. List and briefly explain the advantages of using DBMS approach.	10	L2	CO1
	b.	Define DBMS. Discuss the characteristics of the database approach.	5	L2	CO1
	c.	With neat diagram, explain the three schema architecture.	5	L2	CO1

Q.1 a. Define Database. List and briefly explain the advantages of using DBMS approach (10M)

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.

ADVANTAGES OF USING DBMS:

i. Controlling Redundancy

Redundancy refers to the unnecessary duplication of the same data at multiple locations in a database. The database approach aims to **control and minimize redundancy** by integrating related data into a single database and allowing different users and applications to share the same data.

In traditional file processing systems, the same information may be stored repeatedly in different files. Such duplication leads to several problems and inconsistencies. Therefore, controlling redundancy is one of the important characteristics of the database approach.

ii. Restricting Unauthorized Access

• A DBMS should provide a security and authorization subsystem • most users will not be authorized to access all information in the database

iii. Providing Persistent Storage for Program Objects

• Databases can be used to provide persistent storage for program objects and

data structures

iv. Providing Storage Structures and Search Techniques for Efficient Query Processing

• the DBMS must provide specialized data structures and search techniques to speed up disk search for the desired records – index

• The DBMS often has a buffering or caching module that maintains parts of the database in main memory buffers

• The query processing and optimization module of the DBMS is responsible for choosing an efficient query execution plan for each query based on the existing storage structures.

v. Providing Backup and Recovery

responsible for recovery the database is restored to the state it was in

before the transaction started executing

vi. Providing Multiple User Interfaces

• a DBMS should provide a variety of user interfaces o query languages for casual users, programming language interfaces for application programmers, forms and command codes for parametric users, and menu-driven interfaces and natural language interfaces for standalone users

vii. Representing Complex Relationships among Data

• A DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, and to retrieve and update related data easily and efficiently

viii. Enforcing Integrity Constraints

• The simplest type of integrity constraint involves specifying a data type for each data item

• referential integrity constraint: specifying that a record in one file

must be related to records in other files

• Primary key constraint: uniqueness on data item values.

ix. Permitting inferencing and Actions Using Rules: A trigger is a

form of a rule activated by updates to the table, which results in

performing some additional operations to some other tables,

sending messages, and so on.

x. Additional Implications of Using the Database Approach

Potential for Enforcing Standards: The DBA can enforce standards in a

centralized database environment o Reduced Application Development

Time: Development time using a DBMS is estimated to be one-sixth to

one-fourth of that for a traditional file system o Flexibility: Modern DBMSs

allow certain types of evolutionary changes to the structure of the

database without affecting the stored data and the existing application programs o Availability of Up-to-Date Information: As soon as one user's update is applied to the database, all other users can immediately see this update o Economies of Scale: reduce the wasteful overlap activities thereby reducing the overall costs of operation and management.

Q.2.b. Define DBMS. Discuss the characteristics of the database approach (5M)

A database management system (DBMS) is a collection of programs enabling users to create and maintain a database. More specifically, a DBMS is a general purpose software system facilitating each of the following (with respect to a database):

● definition: specifying data types (and other constraints to which the data must conform) and data organization

● construction: the process of storing the data on some medium (e.g., magnetic disk) that is controlled by the DBMS

● manipulation: querying, updating, report generation

● sharing: allowing multiple users and programs to access the database "simultaneously"

● system protection: preventing database from becoming corrupted when hardware or software failures occur

● security protection: preventing unauthorized or malicious access to database.

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.1.c with neat diagram, explain the three-schema architecture (5M)

5M: Definition and diagram (2M), Each schema description (3 marks - 1 mark each)

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

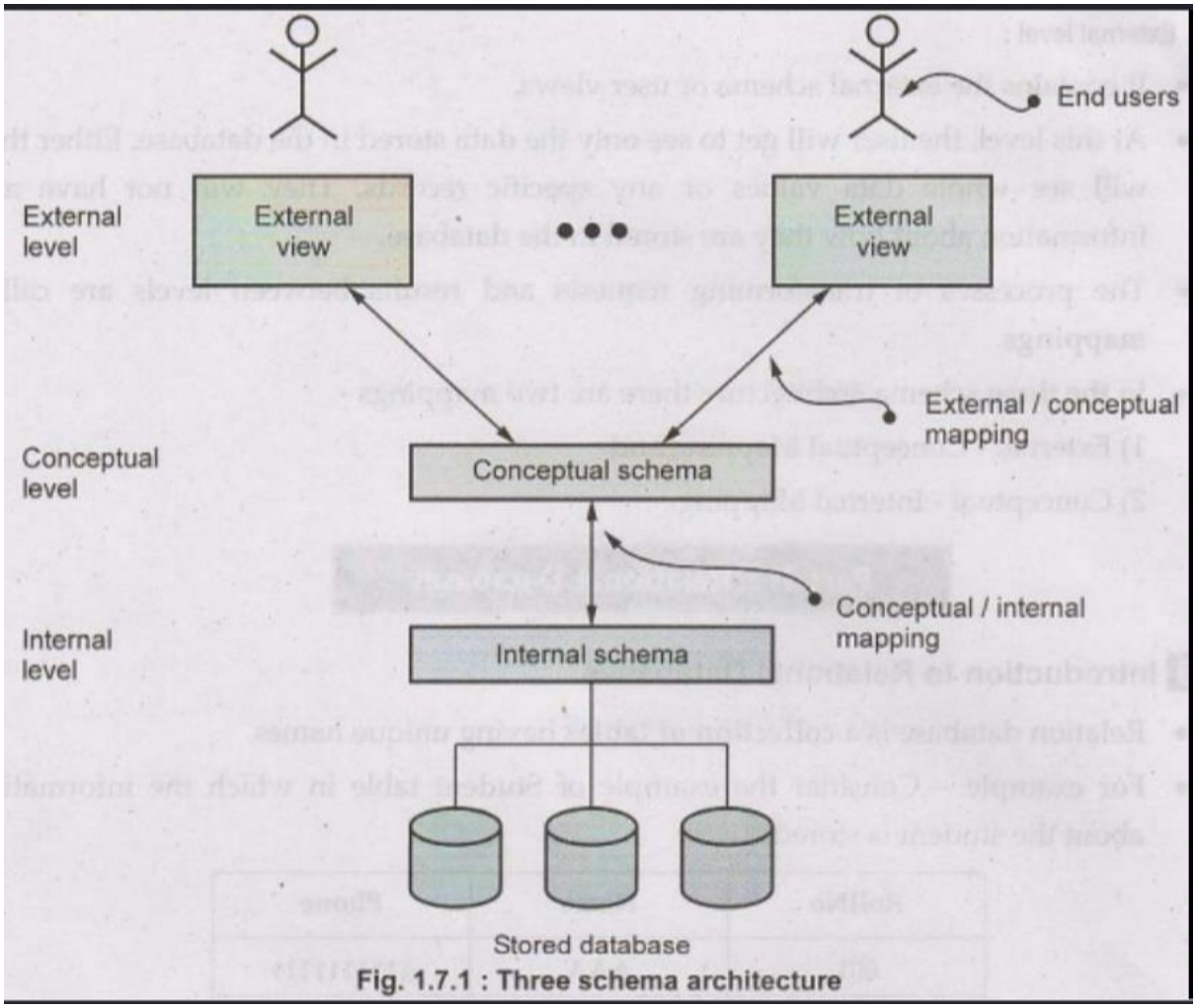


Fig. 1.7.1 : Three schema architecture

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.

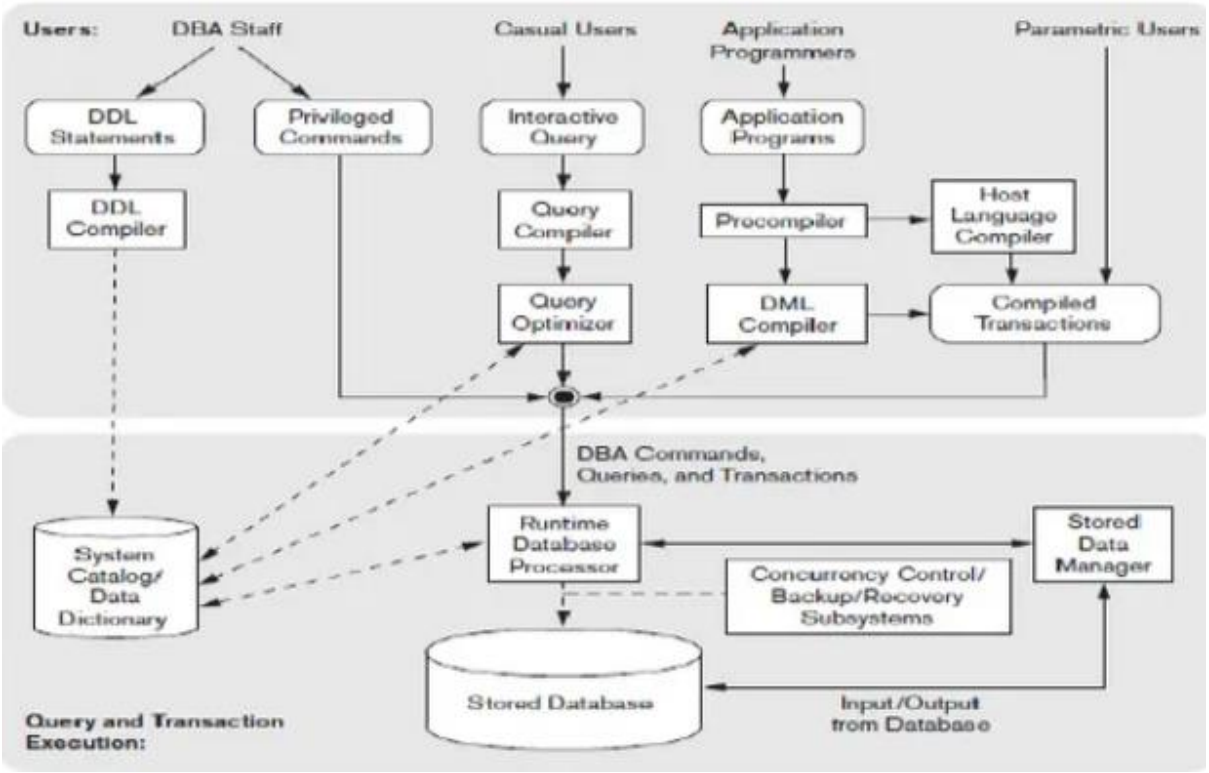
2. The conceptual level has a conceptual schema, -describes the structure of the whole database for a community of users -hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints -representational data model 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	**a.**	Explain the component module of DBMS and their interactions with the help of neat diagram.	**10**	**L2**	**CO1**
	b.	Define the following terms: i) Weak entity ii) DBMS catalog iii) Value sets iv) Cardinality ratio v) Degree of a relationship	**10**	**L2**	**CO1**

Q.2.a Explain the component module of DBMS and their interactions with the help of neat diagram (10M)



A DBMS is a complex software system • discuss the types of software components that constitute a DBMS and the types of computer system software with which the DBMS interacts • two parts: o The top part: various users and their interfaces o The lower part: storage of data and processing of transactions • Access to the disk is controlled primarily by the operating system (OS) • buffer management module to schedule disk read/write, because this has a considerable effect on performance • A higher-level stored data manager module: controls access to DBMS information that is stored on disk, whether it is part of the database or the catalog • DDL Statement: used by the DBA for defining the database and tuning it • DDL Compiler: processes schema definitions and stores descriptions of the schemas in the DBMS catalog.

Privileged commands: The commands which are exclusively used by the DBA • Interactive query: The interface by which the casual users and persons with occasional need for information from the database interact • Query compiler: Queries are parsed and validated for correctness of the query syntax • Query optimizer: Responsible for optimising the query and its execution.

Pre-compiler: Extracts DML commands from an application program written in a host programming language • DML compiler: Compilation of pre-compiled DMLC commands into object code for database access

Host language compiler: The rest of the program is sent to the host language compiler. • Compiled transactions: Canned transactions are executed repeatedly by parametric users, who simply supply the parameters to the transactions.

2. b. Define the following terms: Weak entity, DBMS catalog, Value sets, Cardinality ratio, Degree of a relationship. (10 Marks)

1. Weak Entity

A **weak entity** is an entity type that does not possess a primary key of its own and whose existence depends on another entity called the **owner entity** or **strong entity**. A weak entity is identified by combining the primary key of the owner entity with its own partial key.

The relationship between a weak entity and its owner entity is known as an **identifying relationship**. A weak entity cannot exist independently without the corresponding strong entity.

Example

Consider the entities **Employee** and **Dependent**. A dependent cannot be uniquely identified without the employee to whom it belongs. Hence, **Dependent** is a weak entity.

2. DBMS Catalog

A **DBMS catalog**, also known as the **data dictionary** or **system catalog**, is a repository maintained by the DBMS that stores information about the database structure and its objects. The information stored in the catalog is called **metadata**, which means "data about data."

The DBMS catalog contains information such as:

- Names of relations (tables)
- Attribute names and data types
- Constraints and keys
- Indexes
- User privileges
- Views and schemas

The query processor and other DBMS components use the catalog to process and optimize queries efficiently.

3. Value Sets

A **value set** is the collection of all possible values that an attribute can assume. It specifies the permissible domain of values for an attribute and helps maintain data integrity.

The value set defines:

- Data type of the attribute
- Range of valid values
- Format and size restrictions

Restricting an attribute to a predefined set of values prevents invalid data from being entered into the database.

Example

For an attribute **Gender**, the value set may be:

{Male, Female, Other}

Similarly, for the attribute **Age**, the value set may consist of integer values between 0 and 120.

4. Cardinality Ratio

The **cardinality ratio** specifies the number of entities to which another entity can be associated through a relationship. It describes the mapping constraints between entity sets and indicates the maximum number of relationship instances in which an entity can participate.

The four cardinality ratios are:

(i) One-to-One (1:1)

Each entity in one set is associated with at most one entity in another set.

Example: Person – Passport

(ii) One-to-Many (1:N)

One entity in the first set can be associated with many entities in the second set.

Example: Department – Employee

(iii) Many-to-One (N:1)

Many entities in one set are associated with one entity in another set.

Example: Employee – Department

(iv) Many-to-Many (M:N)

Many entities in one entity set are associated with many entities in another entity set.

Example: Student – Course

Cardinality ratio helps in designing the database and defining relationship constraints.

5. Degree of a Relationship

The **degree of a relationship** indicates the number of entity types participating in a relationship type. It represents how many entities are involved in establishing the relationship.

Depending on the number of participating entity sets, relationships are classified as:

(i) Unary Relationship (Degree 1)

A relationship involving only one entity type.

Example: Employee supervises Employee.

(ii) Binary Relationship (Degree 2)

A relationship involving two entity types.

Example: Student enrolls in Course.

(iii) Ternary Relationship (Degree 3)

A relationship involving three entity types.

Example: Supplier supplies Part to Project.

(iv) N-ary Relationship (Degree n)

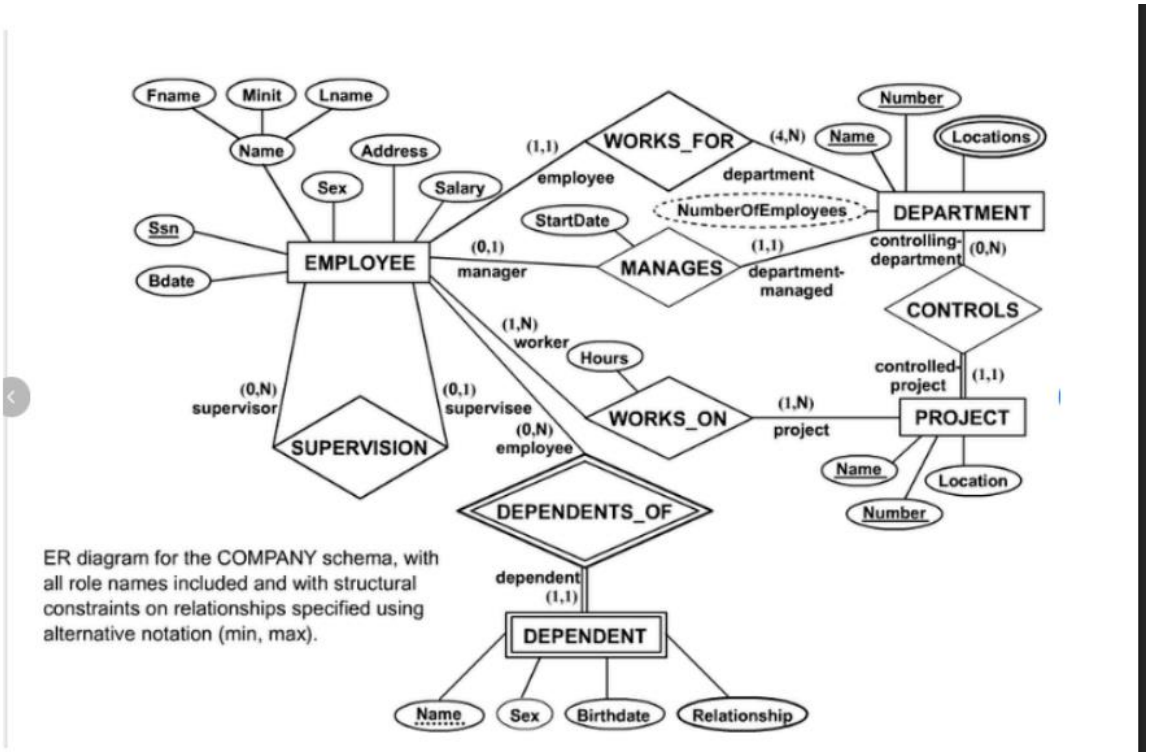
A relationship involving more than three entity types.

The degree of a relationship is an important concept in the Entity-Relationship (ER) model because it helps represent associations among multiple entity sets.

Q.3	a.	Build an ER diagram of company system taking into account at least four entities. Indicate all keys, constraints and assumptions that are made.	10	L3	CO2
	b.	With an example, explain the steps of ER to relational mapping algorithm.	10	L2	CO2

3. a. Build an ER diagram of the company system taking into account at least four entities. Indicate all keys, constraints and assumptions that are made. (10 Marks)

An **Entity-Relationship (ER) diagram** is a conceptual model used to represent entities, attributes, relationships, and constraints in a database system. A company database stores information about employees, departments, projects, and dependents along with the relationships among them.



ER diagram for the COMPANY schema, with all role names included and with structural constraints on relationships specified using alternative notation (min, max).

Entities and Their Attributes

1. EMPLOYEE

Stores information about employees working in the company.

Attributes

- **Emp_ID** (Primary Key)
- Name
- Address
- Salary
- Gender
- Date_of_Birth

2. DEPARTMENT

Stores details of various departments in the company.

Attributes

- **Dept_ID** (Primary Key)
- Dept_Name
- Location

3. PROJECT

Stores information about projects undertaken by the company.

Attributes

- **Project_ID** (Primary Key)
- Project_Name
- Budget
- Location

4. DEPENDENT

Stores information about dependents of employees.

Attributes

- **Dependent_Name** (Partial Key)
- Age
- Relationship

DEPENDENT is a **weak entity** because its existence depends on EMPLOYEE.

Relationships and Constraints

(i) WORKS_FOR

Relationship between EMPLOYEE and DEPARTMENT.

- An employee works for one department.
- A department can have many employees.

Cardinality Ratio

N:1N:1N:1

(ii) MANAGES

Relationship between EMPLOYEE and DEPARTMENT.

- One employee manages one department.
- Each department has one manager.

Cardinality Ratio

1:11:11:1

(iii) WORKS_ON

Relationship between EMPLOYEE and PROJECT.

- One employee may work on many projects.
- One project may involve many employees.

Cardinality Ratio

M:NM:NM:N

Attribute:

- Hours

(iv) HAS_DEPENDENT

Relationship between EMPLOYEE and DEPENDENT.

- One employee may have several dependents.
- Each dependent belongs to exactly one employee.

Cardinality Ratio

1:N1:N1:N

Identifying relationship since DEPENDENT is a weak entity.

3. b. With an example, explain the steps of ER to relational mapping algorithm. (10 Marks)

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)
```

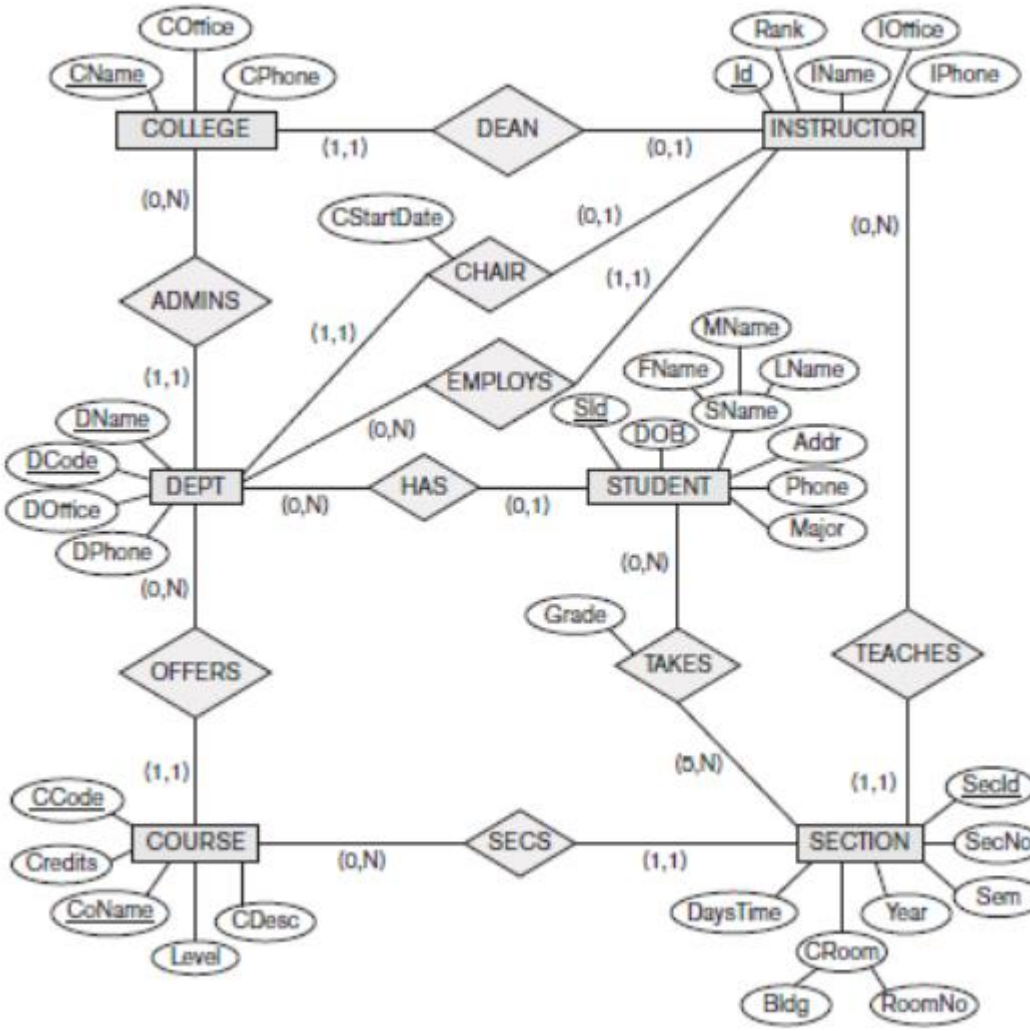
Primary Key:

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

OR

Q.4	**a.**	Build an ER diagram for university database by considering atleast 5 entities.	**10**	**L3**	**CO2**
	b.	Illustrate specialization and generalization with example.	**10**	**L2**	**CO2**

4. a. Build an ER diagram for university database by considering at least 5 entities. (10 Marks)



4. b. Illustrate specialization and generalization with examples. (10 Marks)

Specialization is a top-down approach in which a higher-level entity set is divided into lower-level entity sets based on distinguishing characteristics.

Example Consider an EMPLOYEE entity. Employees can be classified into TEACHING and NON-TEACHING staff.

● EMPLOYEE (EmpID, Name, Salary)

● TEACHING (Subject, Qualification)

● NON-TEACHING (Role, Experience) Here, TEACHING and NON-TEACHING are specialized entities derived from EMPLOYEE.

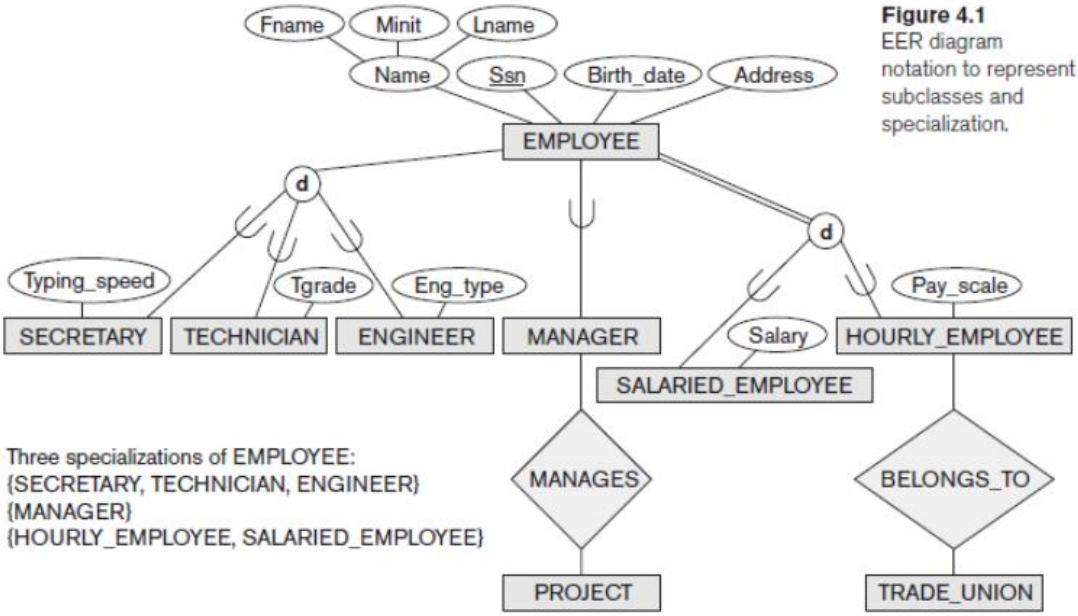


Figure 4.1
EER diagram notation to represent subclasses and specialization.

Generalization

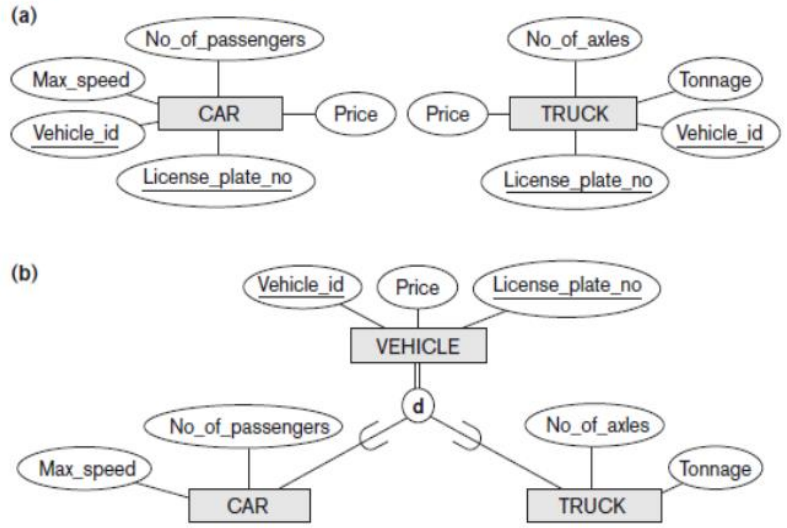
Generalization is a bottom-up approach where multiple lower-level entities are combined into a higher-level entity by identifying common features.

Example Consider STUDENT and FACULTY entities.

- STUDENT (ID, Name, Course)
- FACULTY (ID, Name, Designation)

Common attributes are combined into a generalized entity **PERSON**.

Figure 4.3
Generalization. (a) Two entity types, CAR and TRUCK.
(b) Generalizing CAR and TRUCK into the superclass VEHICLE.



Module – 3					
Q.5	a.	Explain about relational model constraints.	10	L2	CO3
	b.	By refereeing the following database schema: EMPLOYEE (Fname, Minit, Lname, SSN, Bdate, Address, Sex, Salary, Sup-SSN, Dno) Department (Dname, Dnumber, Mgr-SSN, Mgr-start-date) Dept-Locations (Dnumber, Dlocation) Project (Pname, Pnumber, Plocation, Dnum) Works – on (Essn, Pno, Hours) Dependent (Essn, Dependent-name, sex, Bdate, Relationship) Show the relational algebra expressions for the following queries. i) Retrieve the name and address of all employees who work for the 'Research' department. ii) Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' either as a worker or as a manager of the department that controls the project. iii) List the names of managers who have at least one dependent. iv) Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary. v) For each project, retrieve the project number, the project name, and the number of employees who work on that project.	10	L3	CO3

Q 5. a. Explain about relational model constraints. (10M)

The **relational model** represents data in the form of relations (tables). To maintain the **accuracy, consistency, and integrity** of the data stored in these relations, certain restrictions called **relational model constraints** are imposed. These constraints prevent invalid data from being entered into the database and ensure that the database remains in a consistent state.

The main types of relational model constraints are:

1. Domain Constraints
2. Key Constraints
3. Entity Integrity Constraint
4. Referential Integrity Constraint

1. Domain Constraints

A **domain constraint** specifies that the value of each attribute must be an atomic value and must belong to a predefined domain or set of permissible values. The domain determines the data type, format, and range of values that an attribute can take.

Domain constraints ensure that only valid values are stored in the database.

Example

Consider the relation:

`STUDENT(USN, Name, Age)`

Here,

- Age should be an integer.
- Age should lie between 18 and 30.

Valid tuples:

USN	Name	Age
101	Ravi	20
102	Priya	21

Invalid tuple:

USN	Name	Age
103	Arjun	'ABC'

Since Age must be numeric, the above value violates the domain constraint.

Importance

- Maintains data accuracy.
- Prevents invalid values.
- Ensures attribute consistency.

2. Key Constraints

A **key constraint** states that no two tuples in a relation can have the same value for a key attribute. A key uniquely identifies each tuple in a relation.

Thus, duplicate values are not permitted for candidate keys.

Example

Consider:

`EMPLOYEE(Emp_ID, Name, Salary)`

where

`Emp_ID`

is the primary key.

Valid relation:

Emp_ID	Name	Salary
E101	Ravi	50000
E102	Priya	45000

Invalid relation:

Emp_ID	Name	Salary
E101	Ravi	50000
E101	Arjun	60000

Here, duplicate values of Emp_ID violate the key constraint.

Importance

- Uniquely identifies tuples.
- Eliminates duplication.
- Facilitates efficient searching.

3. Entity Integrity Constraint

The **entity integrity constraint** states that no attribute forming the primary key of a relation can have a NULL value. Since the primary key uniquely identifies each tuple, its value must always be known.

Therefore,

Primary key attributes cannot contain NULL values.

Example

Consider:

```
STUDENT(USN, Name, Branch)
```

where

```
USN
```

is the primary key.

Valid relation:

USN	Name	Branch
101	Ravi	CSE
102	Priya	ISE

Invalid relation:

USN	Name	Branch
NULL	Arjun	ECE

Since the primary key value is NULL, the entity integrity constraint is violated.

Importance

- Guarantees unique identification of tuples.
- Prevents ambiguity.
- Maintains integrity of relations.

4. Referential Integrity Constraint

The **referential integrity constraint** maintains consistency among relations. It states that the value of a foreign key in one relation must either match an existing primary key value in another relation or be NULL.

This ensures that references between relations are valid.

Example

Consider:

DEPARTMENT

Dept_ID	Dept_Name
D1	CSE
D2	ISE

EMPLOYEE

Emp_ID	Name	Dept_ID
E1	Ravi	D1
E2	Priya	D2

Here,

`Dept_ID`

in EMPLOYEE is a foreign key referring to

`Dept_ID`

in DEPARTMENT.

Invalid tuple:

Emp_ID	Name	Dept_ID
E3	Arjun	D5

Since department D5 does not exist in DEPARTMENT, the referential integrity constraint is violated.

Importance

- Maintains consistency among related tables.

- Prevents dangling references.
- Preserves relationships between tuples.

	b.	By refereeing the following database schema: EMPLOYEE (Fname, Minit, Lname, SSN, Bdate, Address, Sex, Salary, Sup-SSN, Dno) Department (Dname, Dnumber, Mgr-SSN, Mgr-start-date) Dept-Locations (Dnumber, Dlocation) Project (Pname, Pnumber, Plocation, Dnum) Works – on (Essn, Pno, Hours) Dependent (Essn, Dependent-name, sex, Bdate, Relationship) Show the relational algebra expressions for the following queries. i) Retrieve the name and address of all employees who work for the 'Research' department. ii) Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' either as a worker or as a manager of the department that controls the project. iii) List the names of managers who have at least one dependent. iv) Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary. v) For each project, retrieve the project number, the project name, and the number of employees who work on that project.	**10**	**L3**	**CO3**

i) Retrieve the name and address of all employees who work for the 'Research' department.

Step 1: Select Research department

$$D_1 \leftarrow \sigma_{Dname='Research'}(DEPARTMENT)$$

Step 2: Join with EMPLOYEE

$$E_1 \leftarrow EMPLOYEE \bowtie_{EMPLOYEE.Dno=D_1.Dnumber} D_1$$

Step 3: Project required attributes

$$\pi_{Fname,Minit,Lname,Address}(E_1)$$

ii) Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' either as a worker or as a manager of the department that controls the project.

Case 1: Smith works on the project

$$E_1 \leftarrow \sigma_{Lname='Smith'}(EMPLOYEE)$$

$$P_1 \leftarrow \pi_{Pno}(WORKS\_ON \bowtie_{Essn=SSN} E_1)$$

Case 2: Smith is manager of the department controlling the project

$$D_1 \leftarrow DEPARTMENT \bowtie_{Mgr\text{-}SSN=SSN} \sigma_{Lname='Smith'}(EMPLOYEE)$$

$$P_2 \leftarrow \pi_{Pnumber}(PROJECT \bowtie_{Dnum=Dnumber} D_1)$$

iii) List the names of managers who have at least one dependent.

Step 1: Get managers' SSN

$$M \leftarrow \pi_{Mgr\text{-}SSN}(DEPARTMENT)$$

Step 2: Find managers with dependents

$$MD \leftarrow M \bowtie_{Mgr\text{-}SSN=Essn} DEPENDENT$$

Step 3: Get manager names

$$\pi_{Fname,Minit,Lname}(EMPLOYEE \bowtie_{SSN=Mgr\text{-}SSN} MD)$$

iv) Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.

Using **aggregate functions**:

$$\gamma_{SUM(Salary),\ MAX(Salary),\ MIN(Salary),\ AVG(Salary)}(EMPLOYEE)$$

v) For each project, retrieve the project number, the project name, and the number of employees who work on that project.

Step 1: Join PROJECT and WORKS_ON

$$PW \leftarrow PROJECT \bowtie_{Pnumber=Pno} WORKS\_ON$$

Step 2: Group and count employees

$$\gamma_{Pnumber,Pname;\ COUNT(Essn)}(PW)$$

Q.6	a.	Explain how the basic operations deal with constraints violations with example.	10	L2	CO3
	b.	Explain Division operation with example.	10	L2	CO3

Q.6 a. Explain how the basic operations deal with constraints violations with examples. (10M)

The basic operations in a relational database are **INSERT, DELETE, and UPDATE**. These operations may violate **domain constraints, key constraints, entity integrity constraints, and referential integrity constraints**. The DBMS checks these constraints before executing the operations and takes suitable actions to maintain database consistency.

1. INSERT Operation

The **INSERT** operation adds new tuples into a relation. During insertion, the DBMS verifies whether the new tuple satisfies all integrity constraints.

Possible Violations

- **Domain constraint violation:** Attribute values do not belong to the specified domain.
- **Key constraint violation:** Duplicate primary key values.
- **Entity integrity violation:** Primary key value is NULL.
- **Referential integrity violation:** Foreign key value does not exist in the referenced relation.

Example

EMPLOYEE(Emp_ID, Name)

Existing tuple:

```
(E1, Ravi)
```

Insertion:

```
(E1, Arjun)
```

Since **Emp_ID** already exists, the insertion is rejected.

2. DELETE Operation

The **DELETE** operation removes tuples from a relation. Deleting a tuple may violate referential integrity if it is referenced by tuples in another relation.

Example

DEPARTMENT

Dept_ID	Dept_Name
D1	CSE

EMPLOYEE

Emp_ID	Name	Dept_ID
E1	Ravi	D1

Deleting department **D1** leaves the EMPLOYEE relation with an invalid reference.

Actions Taken by DBMS

- Reject the deletion.
- Perform **cascade deletion**.
- Set foreign key values to **NULL**.
- Set foreign key values to default values.

3. UPDATE Operation

The **UPDATE** operation modifies existing tuples. Updating primary key or foreign key values may violate integrity constraints.

Example

Suppose

DEPARTMENT

Dept_ID	Dept_Name
D1	CSE

EMPLOYEE

Emp_ID	Name	Dept_ID
E1	Ravi	D1

Updating

```
UPDATE DEPARTMENT
SET Dept_ID='D5'
WHERE Dept_ID='D1';
```

may leave EMPLOYEE tuples referring to a non-existing department.

Actions Taken by DBMS

- Reject the update.
- Perform **cascade update**.

- Set foreign key values to **NULL**.
- Set foreign key values to default values.

Q.6.b. Explain DIVISION operation with example. (10M)

- The division operation (÷) is used when we want to find tuples in one relation that are related to all tuples in another relation.
- It is usually applied to queries like "Find all X that are related to all Y".

If we have two relations:

- **R(A, B)**
- **S(B)**

The division **R ÷ S** gives a relation **T(A)** such that:

$$T = \{a \mid \text{for every } b \in S, (a, b) \in R\}$$

In other words: Return all **a** values from R that are associated with every **b** in S.

DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

The DIVISION operation. (a) Dividing SSN_PNOS by SMITH_PNOS. (b) $T \leftarrow R \div S$.

(a)

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

(b)

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

S

A
a1
a2
a3

T

B
b1
b4

		Module – 4			
Q.7	a.	Explain INSERT, DELETE, UPDATE statements in SQL taking suitable examples.	10	L2	CO3
	b.	What is Normalization? Explain 1NF, 2NF and 3NF with examples.	10	L2	CO4

Q.7.a. Explain INSERT, DELETE, UPDATE statements in SQL taking suitable examples. (10M)

SQL (**Structured Query Language**) provides various commands for manipulating data stored in a database. The commands used to insert, modify, and delete data are called **Data Manipulation Language (DML)** statements. The three important DML statements are:

1. **INSERT**
2. **DELETE**
3. **UPDATE**

These statements change the state of a relation by adding, removing, or modifying tuples.

1. INSERT Statement

Definition

The **INSERT** statement is used to add new records (tuples) into a table. It inserts values into specified attributes of a relation.

Syntax

```
INSERT INTO table_name
VALUES(value1, value2, ...);
```

Example

Consider the relation:

STUDENT(USN, Name, Branch)

To insert a record:

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

After insertion:

USN	Name	Branch

101	Ravi	CSE

Characteristics

- Adds new tuples to a relation.
- Changes the database state.
- Must satisfy integrity constraints.

2. DELETE Statement

Definition

The **DELETE** statement is used to remove existing records from a table. Tuples satisfying a specified condition are deleted.

Syntax

```
DELETE FROM table_name
WHERE condition;
```

Example

Suppose STUDENT relation contains:

USN	Name	Branch
101	Ravi	CSE
102	Priya	ISE

To delete the student whose USN is 102:

```
DELETE FROM STUDENT
WHERE USN = 102;
```

After deletion:

USN	Name	Branch
101	Ravi	CSE

Characteristics

- Removes tuples from a relation.
- Can delete one or more records.
- Deletion without a WHERE clause removes all tuples.

3. UPDATE Statement

Definition

The **UPDATE** statement is used to modify existing values in one or more tuples of a relation.

Syntax

```
UPDATE table_name
SET attribute = value
WHERE condition;
```

Example

Suppose STUDENT relation contains:

USN	Name	Branch
101	Ravi	CSE
102	Priya	ISE

To change the branch of student 102 from ISE to AIML:

```
UPDATE STUDENT
SET Branch = 'AIML'
WHERE USN = 102;
```

After updation:

USN	Name	Branch
101	Ravi	CSE
102	Priya	AIML

Characteristics

- Modifies existing tuples.
- Can update one or multiple attributes.
- Uses the WHERE clause to specify affected tuples.

Q.7.b. What is Normalization? Explain 1NF, 2NF and 3NF with examples. (10M)

Normalization is a systematic process of organizing data in a database to minimize redundancy and eliminate undesirable characteristics such as insertion, deletion, and update anomalies. It involves decomposing a relation into smaller and well-structured relations while preserving data integrity and consistency.

In poorly designed relations, the same information may be stored repeatedly, leading to wastage of storage space and inconsistency in data. Normalization provides a set of guidelines, known as normal forms, that help in designing efficient relational schemas.

Normalization is required for the following reasons:

- To eliminate data redundancy and duplication.
- To avoid insertion anomalies.
- To prevent deletion anomalies.
- To eliminate update anomalies.
- To ensure data consistency and integrity.
- To simplify the database structure.
- To improve efficiency in storage and maintenance.
- To facilitate easy modification and retrieval of data.
- To achieve a flexible and reliable database design.

The various stages of normalization are First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), Boyce-Codd Normal Form (BCNF), and higher normal forms.

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.

Third Normal Form (3NF)

A relation is said to be in **Third Normal Form (3NF)** if it is already in 2NF and there are no transitive dependencies among non-key attributes. In other words, no non-prime attribute should depend on another non-prime attribute.

The main objective of 3NF is to eliminate transitive dependencies.

Example

Consider the relation:

EMPLOYEE

EmpID	EmpName	DeptID	DeptName
E1	Ravi	D1	CSE
E2	Priya	D2	ISE
E3	Arjun	D1	CSE

Primary Key:

$$EmpID$$

Functional dependencies are:

$$EmpID \rightarrow DeptID$$

$$DeptID \rightarrow DeptName$$

Therefore,

$$EmpID \rightarrow DeptName$$

Here, DeptName depends on DeptID, which in turn depends on EmpID. Hence, a transitive dependency exists, and the relation is not in 3NF.

Decomposition

EMPLOYEE

EmpID	EmpName	DeptID
E1	Ravi	D1
E2	Priya	D2
E3	Arjun	D1

DEPARTMENT

DeptID	DeptName
D1	CSE
D2	ISE

Now,

- EmpName depends directly on EmpID.
- DeptName depends directly on DeptID.

There are no transitive dependencies, and hence the relations are in Third Normal Form.

Characteristics of 3NF

- Relation must already be in 2NF.
- Eliminates transitive dependencies.
- Reduces redundancy further.
- Prevents insertion, deletion, and update anomalies.
- Improves data consistency and integrity.

Q.8	a.	List and explain informal design guidelines for relation schemas.	10	L2	CO4
	b.	By refereeing the following database schema: Employee (Fname, Minit, Lname, SSN, Bdate, Address, Sex, Salary, Sup-SSN, Dno) Department (Dname, Dnumber, Mgr-SSN, Mgr-start-date) Dept-Locations (Dnumber, Dlocations) Project (Pname, Pnumber, Plocation, Dnum) Works-on (Essn, Pno, Hours) Dependent (Essn, Dependent-Name, Sex, Bdate, Relationship) Write queries in SQL i) Retrieve all the employee names who are working for department number 5. ii) Retrieve all the projects which are controlled by department number 4 iii) Retrieve the names of employees who have no dependents iv) Retrieve the employee name who is working on all the projects in which 'John Smith' works on. v) Retrieve all the project numbers along with number of employee working on each project.	10	L3	CO3

Q. 8. a. List and explain informal design guidelines for relation schemas (10M)

Database design is the process of organizing data into relations so that redundancy and anomalies are minimized. A good relation schema should ensure efficient storage, easy retrieval, and consistency of data. To achieve this, certain **informal design guidelines** are followed while designing relation schemas.

1. Clear Semantics of Attributes

Each relation should represent a single entity or relationship type, and the meaning of its attributes should be easy to understand. Attributes that describe different entities should not be mixed in the same relation.

Example

Instead of storing

```
EMP_DEPT(Emp_ID, Emp_Name, Dept_Name, Dept_Location)
```

it is preferable to have separate relations:

```
EMPLOYEE(Emp_ID, Emp_Name)
```

```
DEPARTMENT(Dept_ID, Dept_Name, Dept_Location)
```

Advantages

- Improves clarity and understanding.
- Simplifies maintenance.
- Avoids ambiguity in the database.

2. Reducing Redundant Information in Tuples

A relation should be designed to minimize duplicate storage of data. Excessive redundancy wastes storage space and may lead to update anomalies.

Example

If department information is repeated for every employee,

```
EMPLOYEE(Emp_ID, Name, Dept_Name, Dept_Location)
```

the department details are stored repeatedly.

This redundancy may cause:

- Insertion anomaly
- Deletion anomaly
- Update anomaly

Advantages

- Saves storage space.
- Maintains consistency.
- Reduces anomalies.

3. Minimizing Null Values in Tuples

Attributes that are frequently NULL should be placed in separate relations. Excessive null values waste storage space and complicate query processing.

Example

Relation:

```
EMPLOYEE(Emp_ID, Name, Phone_No, Spouse_Name)
```

If many employees are unmarried, the attribute **Spouse_Name** will contain numerous NULL values.

Hence, spouse information can be stored separately.

Advantages

- Improves storage efficiency.
- Simplifies query processing.
- Avoids unnecessary NULL values.

4. Disallowing Spurious Tuples

Relations should be designed so that decomposition and join operations do not produce invalid or meaningless tuples called **spurious tuples**.

A decomposition should satisfy the **lossless join property**.

Example

Suppose relation

```
R(A, B, C)
```

is decomposed improperly into

```
R1(A, B)
```

```
R2(B, C)
```

Joining these relations may generate extra tuples that were not present in the original relation.

Advantages

- Preserves data consistency.
- Prevents generation of incorrect information.
- Ensures lossless decomposition.

b.	By refereeing the following database schema: Employee (Fname, Minit, Lname, SSN, Bdate, Address, Sex, Salary, Sup-SSN, Dno) Department (Dname, Dnumber, Mgr-SSN, Mgr-start-date) Dept-Locations (Dnumber, Dlocations) Project (Pname, Pnumber, Plocation, Dnum) Works-on (Essn, Pno, Hours) Dependent (Essn, Dependent-Name, Sex, Bdate, Relationship) Write queries in SQL i) Retrieve all the employee names who are working for department number 5. ii) Retrieve all the projects which are controlled by department number 4 iii) Retrieve the names of employees who have no dependents iv) Retrieve the employee name who is working on all the projects in which 'John Smith' works on. v) Retrieve all the project numbers along with number of employee working on each project.	**10**	**L3**	**CO3**

i) Retrieve all the employee names who are working for department number 5.

```
SELECT Fname, Minit, Lname
FROM EMPLOYEE
WHERE Dno = 5;
```

ii) Retrieve all the projects which are controlled by department number 4

```
SELECT Pname
FROM PROJECT
WHERE Dnum = 4;
```

iii) Retrieve the names of employees who have no dependent.

```
SELECT Fname, Minit, Lname
FROM EMPLOYEE
WHERE SSN NOT IN (
    SELECT Essn
    FROM DEPENDENT
);
```

iv) Retrieve the employee name who is working on all the projects on which 'John Smith' works on.

```
SELECT E.Fname, E.Minit, E.Lname
FROM EMPLOYEE E
WHERE NOT EXISTS (
   (SELECT Pno
    FROM WORKS_ON W1, EMPLOYEE J
    WHERE J.Fname = 'John'
      AND J.Lname = 'Smith'
      AND W1.Essn = J.SSN)
   EXCEPT
   (SELECT W2.Pno
    FROM WORKS_ON W2
    WHERE W2.Essn = E.SSN)
);
```

v) Retrieve all the project numbers along with number of employees working on each project.

```
SELECT Pno, COUNT(Essn) AS Number_of_Employees
FROM WORKS_ON
GROUP BY Pno;
```

		Module – 5			
Q.9	**a.**	How are triggers and assertions defined in SQL? Explain.	**10**	**L2**	**CO3**
	b.	Discuss the Two-Phase locking techniques for concurrency control.	**10**	**L2**	**CO5**

Q.9. a. How are triggers and assertions defined in SQL? Explain (10M)

Assertions and triggers are mechanisms provided by SQL to maintain the integrity and consistency of a database. While **assertions** are used to specify constraints that must always hold true for the database, **triggers** are procedures that are automatically executed in response to certain database events.

1. Assertion

An **assertion** is a schema-level constraint that specifies a condition that must always be satisfied by the database. Whenever an insert, delete, or update operation is performed, the DBMS checks whether the assertion condition is violated. If the condition evaluates to false, the operation is rejected.

Assertions are used to enforce complex constraints involving one or more relations.

Syntax

```
CREATE ASSERTION assertion_name
CHECK (condition);
```

Example

Consider a bank database where the balance of an account should never become negative.

ACCOUNT

Acc_No	Name	Balance
A101	Ravi	10000
A102	Priya	15000

The following assertion ensures that all account balances are non-negative.

```
CREATE ASSERTION Positive_Balance
CHECK (NOT EXISTS
       (SELECT *
        FROM ACCOUNT
        WHERE Balance < 0));
```

Working

- Whenever a tuple is inserted or updated, the DBMS checks the assertion.
- If any account balance becomes negative, the assertion fails.
- The operation causing the violation is rejected.
- Thus, assertions maintain database integrity.

Advantages

- Enforces database-wide constraints.
- Ensures consistency among relations.
- Prevents invalid data from entering the database.

2. Trigger

A **trigger** is a stored procedure that is automatically executed whenever a specified event such as INSERT, DELETE, or UPDATE occurs on a table. Triggers are used to enforce business rules, maintain audit trails, and ensure data integrity.

A trigger is associated with a particular table and is activated automatically when the triggering event occurs.

Syntax

```
CREATE TRIGGER trigger_name
BEFORE | AFTER INSERT | DELETE | UPDATE
ON table_name
FOR EACH ROW
BEGIN
   Trigger body;
END;
```

Example

Consider the relation:

EMPLOYEE

EmpID	Name	Salary
E1	Ravi	50000
E2	Priya	60000

Suppose salary should never be less than ₹20,000.

Trigger Definition

```
CREATE TRIGGER Check_Salary
BEFORE INSERT OR UPDATE
ON EMPLOYEE
FOR EACH ROW
BEGIN
   IF :NEW.Salary < 20000 THEN
      RAISE_APPLICATION_ERROR(-20001,
      'Salary cannot be less than 20000');
   END IF;
END;
```

Working

Suppose the following statement is executed:

```
INSERT INTO EMPLOYEE
VALUES('E3','Arjun',15000);
```

Before inserting the tuple, the trigger is activated.

Since

```
Salary = 15000 < 20000
```

the trigger raises an error and the insertion is rejected.

If

```
INSERT INTO EMPLOYEE
VALUES('E3','Arjun',25000);
```

the condition is satisfied, and the tuple is inserted successfully.

Types of Triggers

1. BEFORE Trigger

Executed before the triggering event occurs.

Example:

```
BEFORE INSERT
BEFORE UPDATE
BEFORE DELETE
```

2. AFTER Trigger

Executed after the triggering event has occurred.

Example:

```
AFTER INSERT
AFTER UPDATE
AFTER DELETE
```

3. ROW-Level Trigger

Executed once for each row affected by the statement.

4. STATEMENT-Level Trigger

Executed once for the entire SQL statement.

Differences Between Assertion and Trigger

Assertion	Trigger
Specifies constraints on the database.	Executes automatically when an event occurs.
Checks a condition that must always hold true.	Performs actions in response to INSERT, UPDATE, or DELETE.
Mainly used for integrity constraints.	Used for auditing, logging, and enforcing business rules.
Does not execute procedural code.	Can execute procedural statements.

Q.9.b Discuss the Two-phase locking 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

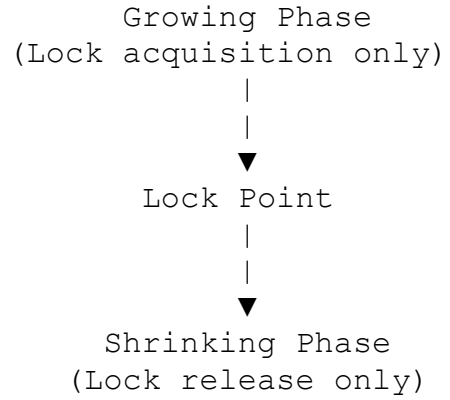
```
Unlock(A)
Unlock(B)
```

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.10	a.	Explain views in SQL with example.	10	L2	CO3
	b.	Explain validation concurrency control techniques in databases.	10	L2	CO5

Q.10.a Explain views in SQL with examples (10M)

A **view** in SQL is a **virtual table** whose contents are derived from one or more base tables. A view does not store data physically; instead, it stores the SQL query used to retrieve data from the underlying tables. Views provide a customized representation of data and enhance security and simplicity.

A **view** is a named table whose contents are obtained dynamically from other tables using a SELECT statement.

General syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Need for Views

- To simplify complex queries.
- To provide different users with different views of the same data.
- To restrict access to sensitive data.
- To improve data security.
- To provide logical data independence.

Creating a View

Consider the relation:

EMPLOYEE(Emp_ID, Name, Salary, Dept_No)

To create a view containing employee details of department 10:

```
CREATE VIEW EMP_DEPT10 AS
SELECT Emp_ID, Name, Salary
FROM EMPLOYEE
WHERE Dept_No = 10;
```

The view **EMP_DEPT10** behaves like a table and contains only employees belonging to department 10.

Retrieving Data from a View

Data from a view can be accessed using the SELECT statement.

```
SELECT * FROM EMP_DEPT10;
```

This displays all records present in the view.

Updating a View

Values in a view can be modified if the view is updateable.

Example:

```
UPDATE EMP_DEPT10
SET Salary = 50000
WHERE Emp_ID = 101;
```

The corresponding value in the underlying EMPLOYEE table is also updated.

Dropping a View

A view can be removed using the DROP VIEW statement.

Syntax:

```
DROP VIEW view_name;
```

Example:

```
DROP VIEW EMP_DEPT10;
```

This removes the view definition from the database.

Types of Views

1. Simple View

A simple view is created from a single table and does not contain grouping functions.

Example

```
CREATE VIEW STUD_VIEW AS
SELECT USN, Name
FROM STUDENT;
```

Characteristics

- Derived from one table.
- Supports INSERT, DELETE, and UPDATE operations.
- Does not contain aggregate functions.

2. Complex View

A complex view is created from multiple tables and may contain aggregate functions and grouping operations.

Example

```
CREATE VIEW DEPT_SALARY AS
SELECT Dept_No, AVG(Salary)
FROM EMPLOYEE
GROUP BY Dept_No;
```

Characteristics

- Derived from multiple tables.
- May contain GROUP BY and aggregate functions.
- Usually not updateable.

Advantages of Views

- Improve database security.
- Simplify query execution.
- Hide complexity of underlying tables.
- Provide logical data independence.
- Allow different users to view different portions of data.

Limitations of Views

- Views do not store data physically.
- Complex views are generally not updateable.
- Excessive use of views may affect query performance.

Q.10.b Explain validation concurrency control techniques in database (10M)

Concurrency control is required to ensure that multiple transactions executing simultaneously do not interfere with one another and that the database remains in a consistent state. One such technique is **Validation Concurrency Control**, also called **Optimistic Concurrency Control (OCC)**.

In this technique, transactions are executed without locking data items. Before committing, each transaction is validated to ensure that no conflict has occurred with other concurrent transactions.

Validation (Optimistic) Concurrency Control

Validation concurrency control assumes that conflicts among transactions are rare. Therefore, transactions proceed freely and are checked only at the time of commit.

A transaction is allowed to commit only if validation is successful; otherwise, it is rolled back and restarted.

Phases of Validation-Based Concurrency Control

The execution of a transaction is divided into three phases:

1. Read Phase

During this phase, the transaction reads data items from the database and performs all computations. Any updates are made on local copies rather than directly on the database.

Characteristics

- Database is not modified.
- Read and write sets are maintained.
- No locks are used.

2. Validation Phase

When the transaction is ready to commit, the system checks whether its execution conflicts with other transactions.

If no conflict exists, the transaction is allowed to proceed. Otherwise, the transaction is aborted and restarted.

Purpose

- Ensures serializability.
- Detects conflicts among concurrent transactions.

3. Write Phase

In this phase, if validation is successful, all updates made on local copies are written to the actual database.

Characteristics

- Database is updated.
- Changes become permanent.
- Transaction enters the committed state.

Working of Validation Technique

Consider two transactions:

T1

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

T2

```
Read(B)
B = B - 50
Write(B)
```

Both transactions execute independently during the read phase. Before writing their results, they enter the validation phase. If no conflict is found, both transactions are committed; otherwise, one transaction is rolled back and restarted.

Conditions for Successful Validation

For transaction T_i and transaction T_j, validation succeeds if one of the following conditions holds:

1. T_i completes its write phase before T_j starts its read phase.

or

2. The write set of T_i does not intersect with the read set of T_j.

Thus, transactions are serializable if these conditions are satisfied.

Advantages

- No deadlocks occur since locking is not used.
- High degree of concurrency.
- Suitable for environments with low data contention.
- Transactions do not wait for one another.
- Better performance when conflicts are rare.

Disadvantages

- Transactions may need to be restarted frequently if conflicts are high.
- Not suitable for heavily loaded databases.
- Validation overhead increases with the number of transactions.
- Long transactions have a higher probability of rollback.