CBCS SCHEME

C.M.R. INSTITUTE OF TECHNOLOGY
CMR
BANGALORE - 37.

USN | | | | | | | | | |

Fourth Semester B.E./B.Tech. Degree Examination, June/July 2025
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.

		Module – 1	M	L	C
Q.1	a.	Explain the types of attributes with example.	4	L2	CO1
	b.	Define database. Explain the main characteristics of the database approach.	8	L2	CO1
	c.	Show the ER diagram for an EMPLOYEE database by assuming your own entities (minimum 4) attributes and relationships, mention cardinality ratios wherever appropriate.	8	L3	CO2
		OR			
Q.2	a.	Describe the three schema architecture.	4	L2	CO1
	b.	Explain the component models of DBMS and their interaction with the help of diagram.	8	L2	CO1
	c.	Design ER diagram for a university database by assuming your own entities (4). Mention primary key , constraints and relationships.	8	L3	CO2
		Module – 2			
Q.3	a.	Explain relational model constraints.	6	L2	CO1
	b.	Explain the characteristics of relations with suitable example for each.	6	L2	CO1
	c.	Considering the following schema : Sailors (<u>sid</u> , sname , rating , age) Boats (<u>bid</u> , bname , color) Reserves (<u>sid</u> , bid , day) Write a relational algebra queries for the following : i) Find the names of sailors, who have reserved red and a green boat. ii) Find the names of sailors who have reserved a red boat. iii) Find the names of sailors who have reserved a red or green boat. iv) Find the names of sailors who have reserved all boats.	8	L3	CO1
		OR			
Q.4	a.	Explain the steps to convert the basic ER model to relational Database schema.	6	L2	CO1
	b.	Explain Unary relational operations with example.	6	L2	CO1

			M	L	C
	c.	Consider the relation schema Employee database. EMPLOYEE (Fname ,Minit , Lname , <u>SSn</u> , Bdates , Address , Sex , Salary Super_SSn , Dno) DEPARTMENT (Dname , <u>Dnumber</u> , Mgr_SSn , Mgr_start_date) PROJECT (Pname , <u>PNumber</u> , Plocation , Dnum) WORKS_ON (<u>Essn</u> , <u>Pno</u> , Hours) DEPENDENT (<u>Essn</u> , Dependent_name , sex, Bdate , Relationship) Write relational algebra queries for the following : i) Retrieve the name and address of all employees who work for the 'Research' department. ii) List the names of all employees with 2 or more dependents. iii) Find the names of employees who work on all the projects controlled by department number 5. iv) List the names of employees who have no dependents.	8	L3	CO3
		Module – 3			
Q.5	a.	What is the need for normalization? Explain second and third normal form with examples.	6	L2	CO4
	b.	Outline constraints in SQL.	6	L2	CO1
	c.	Identify the given Relation R(ABCDE) and its instance, check whether FDS given hold or not. Give reasons. i) $A \rightarrow B$ ii) $B \rightarrow C$ iii) $D \rightarrow E$ iv) $CD \rightarrow E$.	8	L3	CO4

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_2	c_1	d_1	e_1
a_2	b_2	c_1	d_2	e_3
a_2	b_3	c_3	d_2	e_2

CMRIT LIBRARY
BANGALORE - 560 037

		OR	M	L	C
Q.6	a.	What is Multivalued dependency? Explain 4NF and 5NF with suitable example.	6	L2	CO4
	b.	Outline the informal design guidelines for relational schema.	6	L2	CO4
	c.	Consider relation R with following function dependency : EMPPROJ (<u>SSn</u> , <u>Pnumber</u> , Hours , Ename , Pname , Plocation) SSN , Pnumber $\rightarrow$ Hours, SSN $\rightarrow$ Ename Pnumber $\rightarrow$ Pname , Plocation. Is it 2NF? Verify? If no give reason.	8	L3	CO4

		Module – 4			
Q.7	a.	Consider the following schema for a company database : Employee (FName , LName , SSn , Adderss , Sex , Salary , Dno , Super_SSn) Department (Dname , Dnumber , mgr_SSn, mgr_st_date) Project (Pname , Pnumber , Plocation , Dnum) WORKS_on (Essn , Pno , Hours) DEPENDENT (Essn , Dependent name , Sex , Bdate, relationship). Write the SQL queries for the following : i) List the names of managers who have atleast one dependent (use correlated nested). ii) Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee. iii) For each project retrieve the project number , project name and the number of employees who work on that project. iv) Retrieve the SSN of all employees who work on project number 1, 2 or 3. (Use IN). v) Find the sum of the salaries of all employees of the 'Research' department as well as maximum salary , minimum salary , average salary in this department.	10	L3	CO3
	b.	Why concurrency control is needed? Demonstrate with an example.	10	L2	CO5
		OR			
Q.8	a.	Consider the following schedule. The actions are listed in the order they are scheduled and prefixed with the transaction name. S1 : T1 : R(X) , T2 : R(X) T1 : W(Y) , T2 : W(Y) , T1 : R(Y) , T2 : R(Y) S2 : T3 : W(X) , T1 : R(X) , T1 : W(Y) , T2 : R(Z) , T2 : W(Z) , T3 : R(Z) For each schedule answer the following : i) What is the precedence graph for the schedule? ii) Is the schedule conflict_serializable? If so what are all the conflicts equivalent serial schedules? iii) Is the schedule view serializable? If so what are all the view equivalent serial schedules? CMRIT LIBRARY BANGALORE - 560 037	10	L3	CO5
	b.	Explain triggers with example write a trigger in SQL to call a procedure "Inform_Supervisor" whenever an employees salary is greater than the salary of his or her direct supervisor in the COMPANY database.	10	L3	CO5
		Module – 5			
Q.9	a.	Describe the two – phase locking protocol for concurrency control provide example to illustrate how it ensures serializability in transaction schedule.	10	L2	CO5
	b.	Explain the characteristics of NOSQL system.	10	L2	CO6
		OR			
Q.10	a.	Explain binary locks and shared lock with algorithm.	10	L2	CO5
	b.	Explain MongoDB data model, CRUD operations and distributed system characteristics.	10	L2	CO6

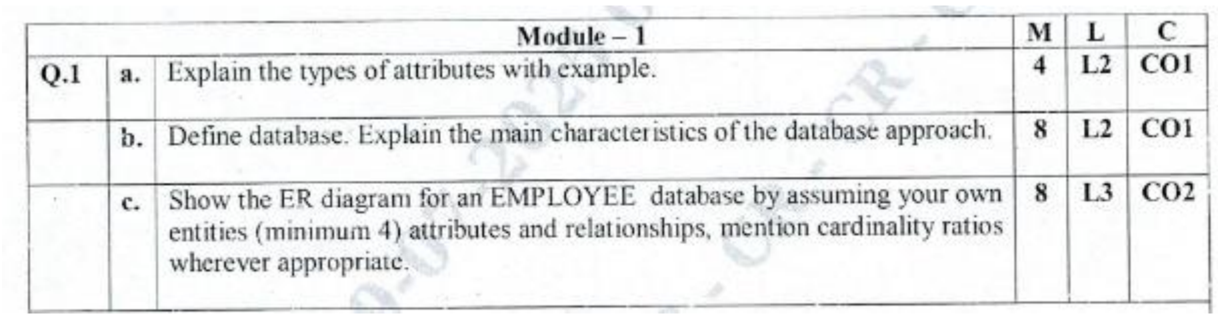
* * * * *

Department of Computer Science & Engineering

BCS403-Database Management System

VTU Exam- June/July 2025 - Solution

Q1)



		Module – 1	M	L	C
Q.1	a.	Explain the types of attributes with example.	4	L2	CO1
	b.	Define database. Explain the main characteristics of the database approach.	8	L2	CO1
	c.	Show the ER diagram for an EMPLOYEE database by assuming your own entities (minimum 4) attributes and relationships, mention cardinality ratios wherever appropriate.	8	L3	CO2

Solution

1. Simple Attribute (Atomic Attribute)

- An attribute that **cannot be divided** further.
- 🔷 **Example**:
  - age, first_name, salary
  - Each holds a single value.

---

2. Composite Attribute

- An attribute that **can be divided** into smaller sub-parts.
- 🔹 **Example**:
    - Name can be split into first_name and last_name.
    - Address can be split into street, city, state, zip.

---

3. Derived Attribute

- An attribute whose value is **derived from other attributes**.
- 🔹 **Example**:
    - age can be derived from date_of_birth.
    - total_price can be derived from unit_price * quantity.

---

4. Multivalued Attribute

- An attribute that can hold **multiple values** for a single entity.
- 🔹 **Example**:
    - A person may have **multiple phone numbers** or **email addresses**.
    - skills for an employee.

---

5. Single-valued Attribute

- An attribute that holds **only one value** for a particular entity.

- 🔹 **Example**:
    - Employee_ID, Date_of_Birth

---

6. Key Attribute

- An attribute (or set of attributes) that **uniquely identifies** a record in a table.
- 🔹 **Example**:
    - student_id in a student table
    - SSN (Social Security Number)

---

7. Optional (Nullable) Attribute

- An attribute that **may or may not have a value** (i.e., it can be NULL).
- 🔹 **Example**:
    - middle_name of a person (not everyone has one)

---

8. Stored Attribute

- The opposite of a derived attribute; **actually stored** in the database.
- 🔹 **Example**:
    - date_of_birth, name

A1b)A **database** is an **organized collection of related data**, stored and accessed electronically. It is designed to efficiently store, manage, and retrieve large amounts of information.

In simple terms, a database allows **storing data in a structured format**, making it easier to **insert, update, delete**, and **query** information as needed.

📎 **Example**:
A **student database** might store information like student ID, name, course, and marks.

Main Characteristics of the Database Approach:

The **database approach** refers to managing data using a **Database Management System (DBMS)** instead of traditional file systems. This approach has several key characteristics:

---

1. Data Abstraction

- Hides the details of how data is stored and maintained.
- Users interact with a high-level view of the data.
- **Three levels**:
  - Physical level (how data is stored)
  - Logical level (what data is stored)
  - View level (how data is presented to users)

---

2. Data Independence

- The ability to change the **data structure** without changing the **application programs**.
- Two types:
  - **Logical data independence** (change in schema without affecting applications)
  - **Physical data independence** (change in storage without affecting schema)

---

3. Reduced Data Redundancy

- Avoids storing the same data in multiple places, which reduces duplication and inconsistency.

---

4. Improved Data Consistency

- By storing data in one centralized place, updates are automatically reflected across the system, reducing inconsistencies.

---

5. Data Sharing

- Multiple users and applications can **access the same database** concurrently while maintaining control and security.

---

6. Data Security

- DBMS provides **access control mechanisms** to restrict unauthorized access.
- Supports **authentication**, **authorization**, and **encryption**.

---

7. Data Integrity

- Ensures **accuracy and correctness** of the data using rules and constraints (e.g., primary key, foreign key, not null).

---

8. Concurrency Control

- Multiple users can access the database at the same time **without interfering** with each other's operations.

---

9. Backup and Recovery

- DBMS provides tools for **automatic backup** and **recovery** in case of system failure.

---

10. Query Language Support

- Most DBMSs use a powerful query language like **SQL** to allow users to **easily interact** with the database.

A1c **Entities & Attributes**

1. **Employee**
   - Emp_ID (PK)
   - Name
   - Gender
   - DOB
   - Salary
2. **Department**
   - Dept_ID (PK)
   - Dept_Name
   - Location
3. **Project**
   - Proj_ID (PK)
   - Proj_Name

- Budget
- Deadline

4. **Dependent**
   - Dep_ID (PK)
   - Dep_Name
   - Relation
   - DOB

Relationships & Cardinalities

1. **Employee – Department**
   - Relationship: *Works_In*
   - Cardinality: **Many-to-One** (Many employees work in one department; each employee belongs to exactly one department).
2. **Department – Project**
   - Relationship: *Manages*
   - Cardinality: **One-to-Many** (One department can manage many projects; each project is managed by one department).
3. **Employee – Project**
   - Relationship: *Assigned_To*
   - Cardinality: **Many-to-Many** (An employee can work on multiple projects; a project can have multiple employees).
   - Needs an **associative entity** (e.g., Assignment with attributes: Role, Hours).
4. **Employee – Dependent**
   - Relationship: *Has*

- Cardinality: **One-to-Many** (One employee can have multiple dependents; each dependent belongs to one employee).

ER Diagram (Textual Representation)

Employee (Emp_ID PK, Name, Gender, DOB, Salary)

| Works_In (M:1)

Department (Dept_ID PK, Dept_Name, Location)

| Manages (1:M)

Project (Proj_ID PK, Proj_Name, Budget, Deadline)

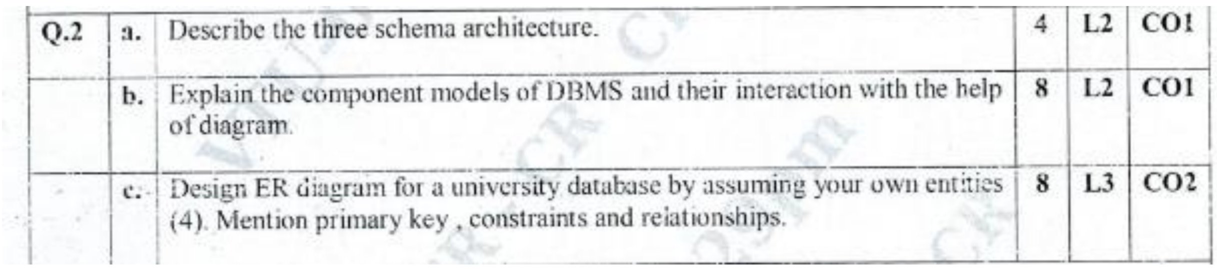
| Assigned_To (M:N) via Assignment(Role, Hours)

Employee

| Has (1:M)

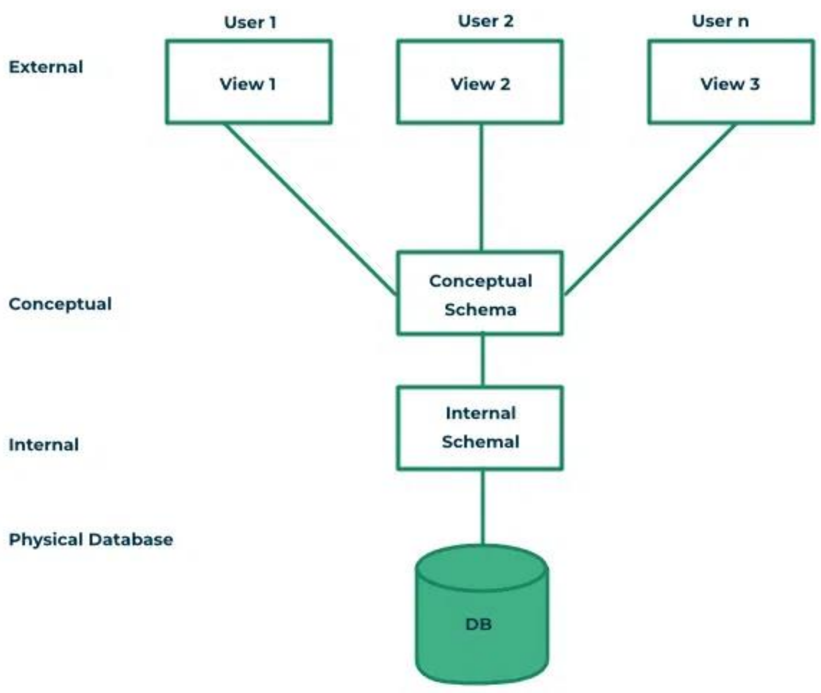
Dependent (Dep_ID PK, Dep_Name, Relation, DOB)

Q2)



Q.2	a.	Describe the three schema architecture.	4	L2	CO1
	b.	Explain the component models of DBMS and their interaction with the help of diagram.	8	L2	CO1
	c.	Design ER diagram for a university database by assuming your own entities (4). Mention primary key , constraints and relationships.	8	L3	CO2

Solution



Three Schema Architecture in DBMS

It is a framework proposed by the ANSI/SPARC committee to separate a database into three levels of abstraction. This architecture helps achieve data abstraction, independence, and security.

---

1. Internal Schema (Physical Level)

- Definition: Describes *how* the data is physically stored in the database.
- Focus: Storage structure, indexing, access paths, file organization, and compression.
- Example:
    - Employee records stored in a heap file with B+ tree indexing on Emp_ID.
    - Data blocks stored in pages of 4KB each.

---

2. Conceptual Schema (Logical Level)

- Definition: Describes *what* data is stored and the relationships among data.
- Focus: Logical structure of the entire database, independent of physical storage.
- Example:
    - Employee table has attributes: Emp_ID, Name, Salary, Dept_ID.
    - Relationship: Employee works in Department.
    - Constraints: Emp_ID is primary key, Dept_ID is foreign key.

---

3. External Schema (View Level)

- Definition: Describes different *views* of the database for different users.
- Focus: User interaction with the database, hiding complexity.
- Example:

  - HR user sees: Emp_ID, Name, Salary.
  - Finance user sees: Emp_ID, Salary, Tax_Info.
  - Manager user sees: Emp_ID, Name, Dept_Name.

Summary Table

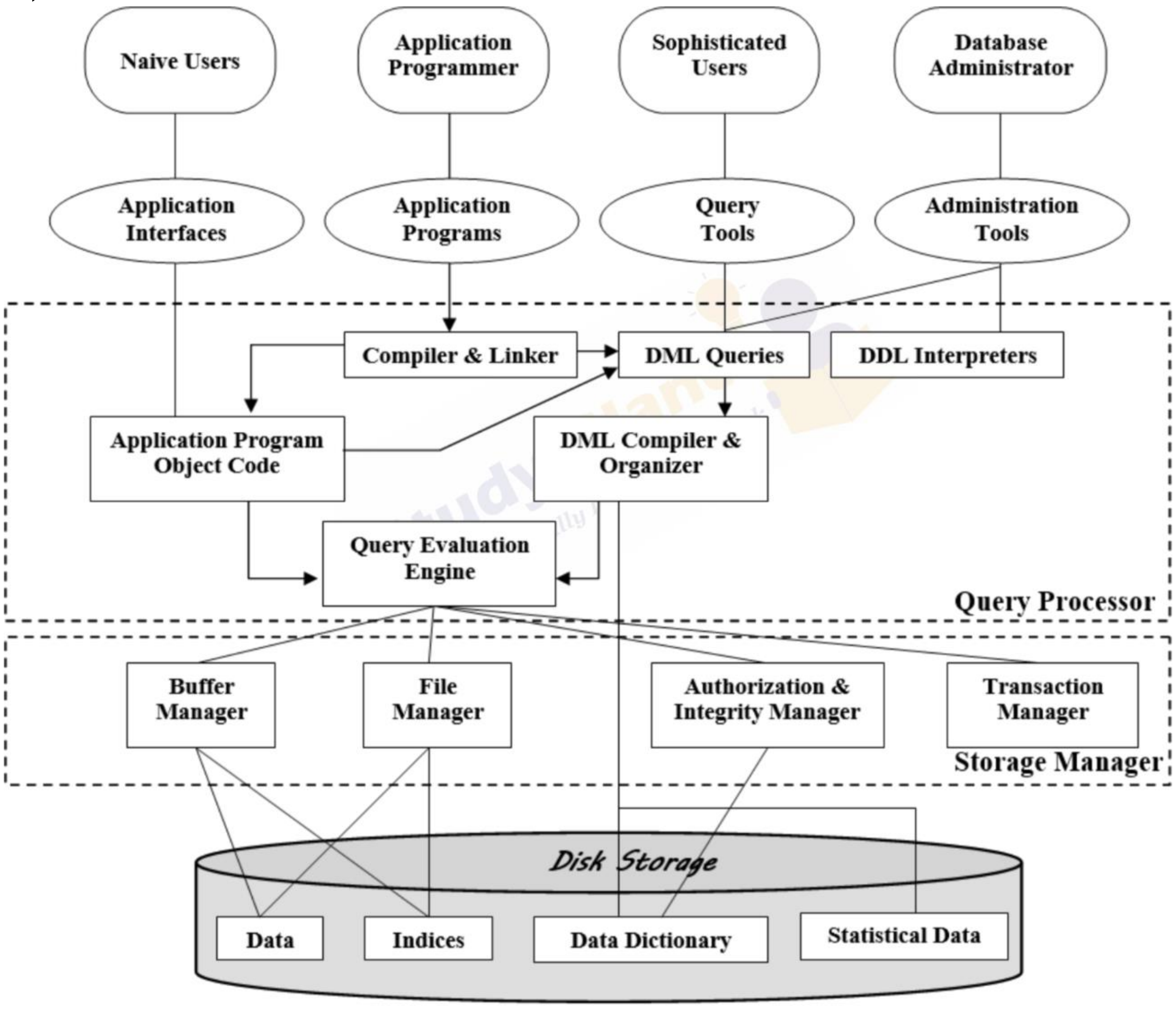
Level	Focus Area	Example Use
Internal	Physical storage, indexing	B+ tree index on Emp_ID
Conceptual	Logical structure of DB	ER model, schema with tables & relationships
External	User views & access control	HR sees employees without sensitive info

---

✅ This separation allows data independence:

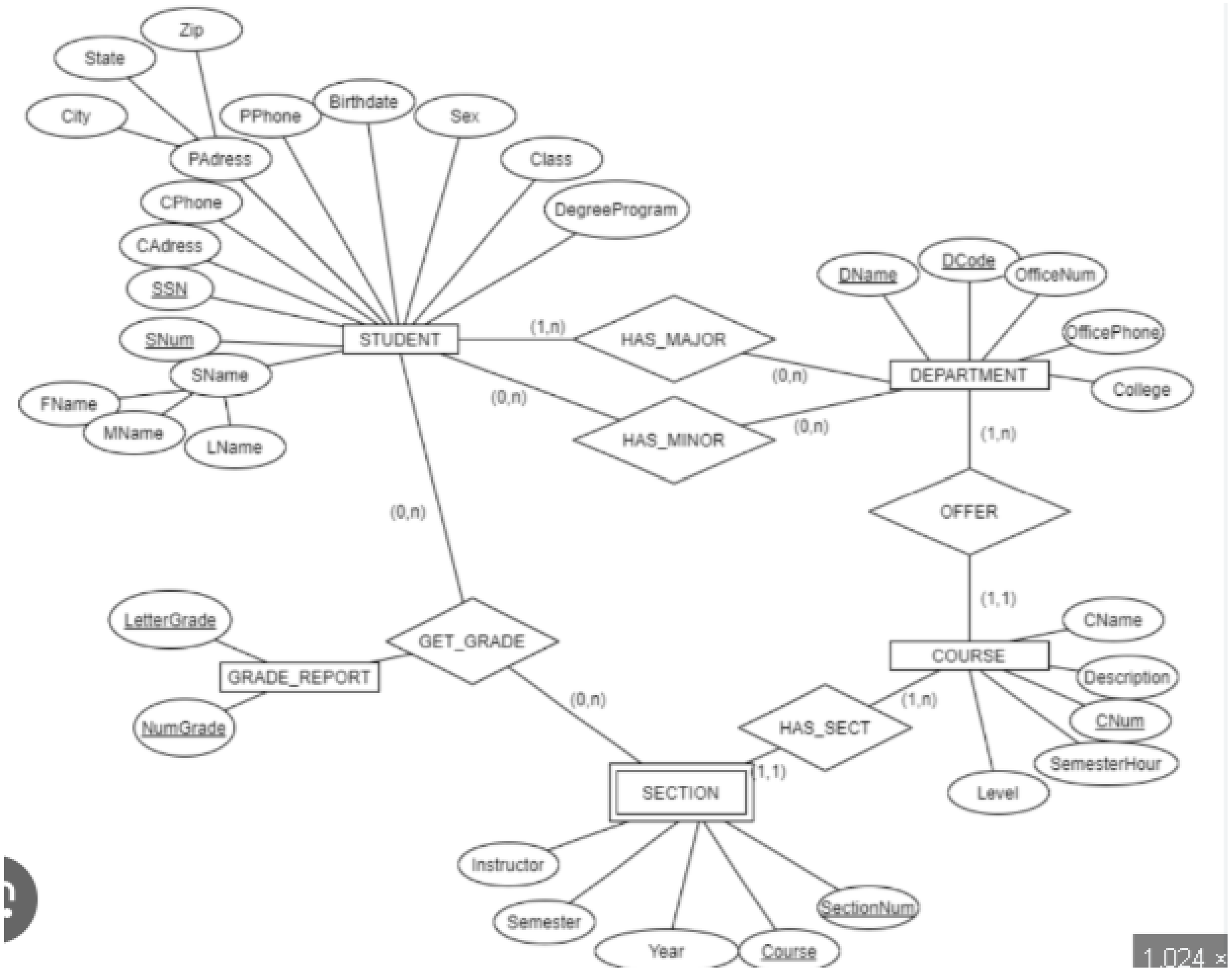
- Logical independence → Changing schema without affecting user views.
- Physical independence → Changing storage structure without affecting schema.

2a)



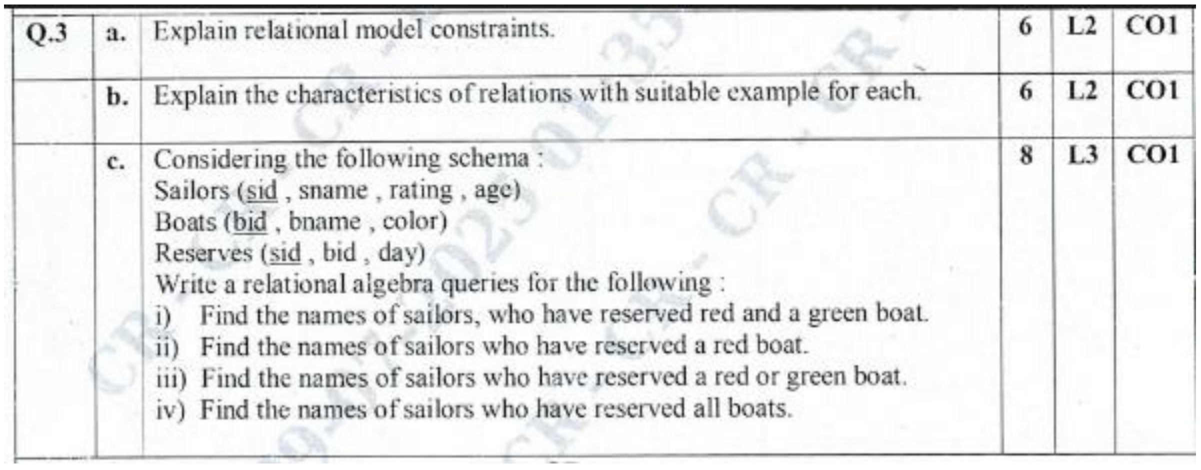
Explain disk storage , different types of users,storage manager,query processors.

2C)



Zip
State
City
PPhone
Birthdate
Sex
PAdress
Class
CPhone
DegreeProgram
CAdress
SSN
SNum
STUDENT
SName
FName
MName
LName
(1,n)
HAS_MAJOR
(0,n)
(0,n)
HAS_MINOR
(0,n)
DName
DCode
OfficeNum
OfficePhone
DEPARTMENT
College
(1,n)
OFFER
(1,1)
(0,n)
LetterGrade
GRADE_REPORT
GET_GRADE
NumGrade
(0,n)
CName
COURSE
Description
(1,n)
HAS_SECT
CNum
SemesterHour
(1,1)
SECTION
Level
Instructor
Semester
Year
Course
SectionNum
1,024 ×

Q3)



Q.3	a.	Explain relational model constraints.	6	L2	CO1
	b.	Explain the characteristics of relations with suitable example for each.	6	L2	CO1
	c.	Considering the following schema : Sailors (sid , sname , rating , age) Boats (bid , bname , color) Reserves (sid , bid , day) Write a relational algebra queries for the following : i) Find the names of sailors, who have reserved red and a green boat. ii) Find the names of sailors who have reserved a red boat. iii) Find the names of sailors who have reserved a red or green boat. iv) Find the names of sailors who have reserved all boats.	8	L3	CO1

Solution

3a)Types of Relational Model Constraints

1. Domain Constraints

- Definition: Values of attributes must come from a defined domain (data type + range of values).
- Purpose: Prevent invalid data entry.
- Example:
  - Age attribute must be an integer between 18 and 60.
  - Email must follow proper format.

---

2. Key Constraints

- Definition: Every relation must have a Primary Key whose values uniquely identify tuples (rows).
- Types:
  - Primary Key → Unique & not null.

    - Unique Key → Unique but may allow nulls.
    - Super Key → Any set of attributes that uniquely identifies tuples.
    - Candidate Key → Minimal super key.
- Example:
    - Emp_ID in Employee table must be unique.

---

3. Entity Integrity Constraint

- Definition: Primary key attribute(s) cannot be NULL.
- Reason: Every entity must be uniquely identifiable.
- Example:
    - An Employee must always have a valid Emp_ID.

---

4. Referential Integrity Constraint

- Definition: A foreign key in one relation must either match a primary key in another relation or be NULL.
- Purpose: Maintains consistency across relations.
- Example:
    - Dept_ID in Employee table must exist in Department table.
    - Cannot assign an employee to a non-existing department.

---

5. General Constraints (Business Rules)

- Definition: Additional rules defined by users or applications.
- Example:
    - An employee's Salary cannot exceed manager's salary.
    - Project Deadline must be a future date.

Summary Table

Constraint Type	Ensures	Example
Domain	Values are valid & within range	Age 18–60
Key	Tuples are uniquely identified	Emp_ID unique
Entity Integrity	PK cannot be NULL	Emp_ID not null
Referential Integrity	FK matches PK or NULL	Dept_ID exists in Department
General	Business-specific rules	`Salary ≤ Manager Salary`

3b)In the Relational Model (DBMS), data is stored in the form of relations (tables). Each relation has some important characteristics that differentiate it from other data models.

Characteristics of Relations

1. Tuples are unordered (Rows)

- A relation is a set of tuples → order of rows does not matter.
- Example:

Emp_ID	Name	Dept
101	John	HR
102	Mary	IT

Both representations are the same relation, regardless of row order.

---

2. Attributes are unordered (Columns)

- Order of attributes (columns) in a relation does not matter.
- Example:

Emp_ID	Name	Dept
101	John	HR

is the same as

Dept	Name	Emp_ID
HR	John	101

---

3. Each Attribute has a Domain

- Each attribute must have a defined domain (set of valid values).
- Example:
  - Emp_ID → Integer (positive only).
  - Name → String (letters only).

- Salary → Decimal(10,2).

---

4. Attribute values are Atomic (No multivalued attributes)

- Each attribute holds a single, indivisible value (1NF rule).
- Example:
  ✗ Wrong: Skills = {Java, SQL, Python}
  ✓ Correct: Represent skills in another table with foreign key.

---

5. Tuples are Unique (No duplicate rows)

- No two tuples (rows) in a relation can be identical.
- Example:
  ✗ Wrong:

Emp_ID	Name	Dept
101	John	HR
101	John	HR

✓ Correct: Each Emp_ID must be unique.

---

6. Null Values are Allowed

- Attributes can have NULL if a value is unknown or not applicable.
- Example:

Emp_ID	Name	Dept	Manager_ID
101	John	HR	NULL

Here, Manager_ID is NULL because John may not have a manager.

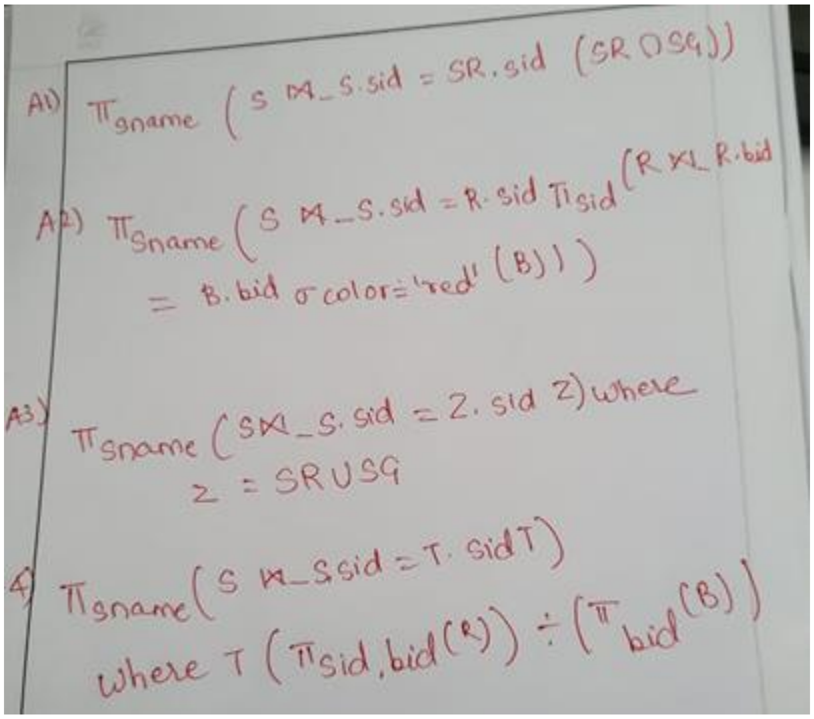
---

7. Relation has a Primary Key

- Each relation must have a primary key that uniquely identifies tuples.
- Example:
  - In Employee relation: Emp_ID is the primary key.
  - In Department relation: Dept_ID is the primary key.
  - 3c)
  - RedB = σ_{color='red'}(B)
  - GreenB = σ_{color='green'}(B)
  - SR = π_{sid}(R ⋈_{R.bid=B.bid} RedB) — sailors who reserved a red boat
  - SG = π_{sid}(R ⋈_{R.bid=B.bid} GreenB) — sailors who reserved a green boat

Note: In (c) I assume “red or green” (the prompt says “red or red”). Adjust the color if needed.

  -
  - 1) Names of sailors who have reserved both a red and a green boat
  -



A1) $\pi_{sname}\left(S \bowtie_{S.sid = SR.sid} (SR \cap SG)\right)$

A2) $\pi_{Sname}\left(S \bowtie_{S.sid = R.sid} \pi_{sid}(R \bowtie_{R.bid = B.bid} \sigma_{color='red'}(B))\right)$

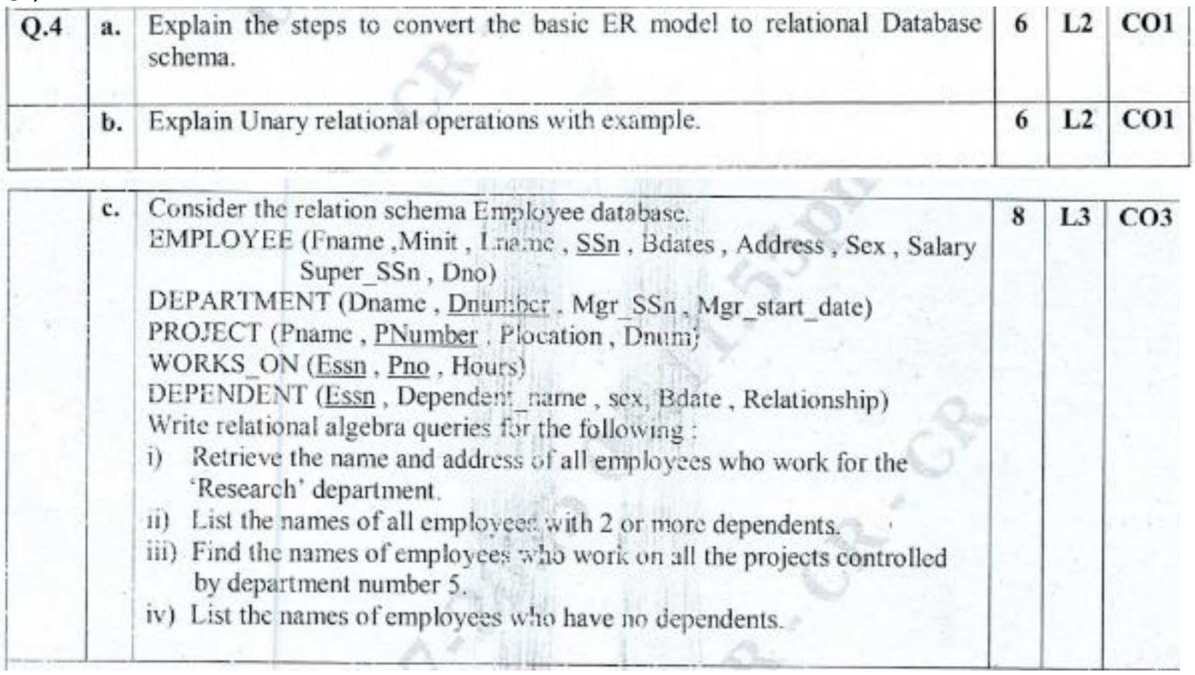
A3) $\pi_{sname}(S \bowtie_{S.sid = Z.sid} Z)$ where

$Z = SR \cup SG$

4) $\pi_{sname}(S \bowtie_{S.sid = T.sid} T)$

where $T\left(\pi_{sid,bid}(R)\right) \div \left(\pi_{bid}(B)\right)$

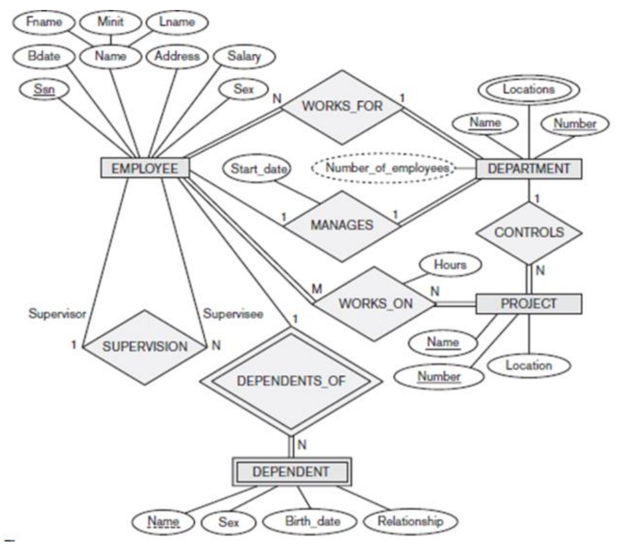
Q4)



Q.4	a.	Explain the steps to convert the basic ER model to relational Database schema.	6	L2	CO1
	b.	Explain Unary relational operations with example.	6	L2	CO1
	c.	Consider the relation schema Employee database. EMPLOYEE (Fname ,Minit , Lname , <u>SSn</u> , Bdates , Address , Sex , Salary Super_SSn , Dno) DEPARTMENT (Dname , <u>Dnumber</u> , Mgr_SSn , Mgr_start_date) PROJECT (Pname , <u>PNumber</u> , Plocation , Dnum) WORKS_ON (<u>Essn</u> , <u>Pno</u> , Hours) DEPENDENT (<u>Essn</u> , Dependent_name , sex, Bdate , Relationship) Write relational algebra queries for the following : i) Retrieve the name and address of all employees who work for the 'Research' department. ii) List the names of all employees with 2 or more dependents. iii) Find the names of employees who work on all the projects controlled by department number 5. iv) List the names of employees who have no dependents.	8	L3	CO3

Solution

(a)



Step 1: Mapping of Regular Entity Types

§ For each regular entity type, create a relation R that includes all the simple attributes of E

§ Include only the simple component attributes of a composite attribute

§ Choose one of the key attributes of E as the primary key for R

§ If the chosen key of E is a composite, then the set of simple attributes that form it will together form the primary key of R.

§ If multiple keys were identified for E during the conceptual design, the information describing the attributes that form each additional key is kept in order to specify secondary (unique) keys of relation R

§ In our example-COMPANY database, we create the relations EMPLOYEE, DEPARTMENT, and PROJECT

§ we choose Ssn, Dnumber, and Pnumber as primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT, respectively

§ The relations that are created from the mapping of entity types are called entity relations

because each tuple represents an entity instance.

Step 2: Mapping of Weak Entity Types

§ For each weak entity type, create a relation R and include all simple attributes of the entity type as attributes of R

§ Include primary key attribute of owner as foreign key attributes of R

§ In our example, we create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT

§ We include the primary key Ssn of the EMPLOYEE relation—which corresponds to the owner entity type—as a foreign key attribute of DEPENDENT; we rename it as Essn

§ The primary key of the DEPENDENT relation is the combination {Essn,Dependent_name}, because Dependent_name is the partial key of DEPENDENT

§ It is common to choose the propagate (CASCADE) option for the referential triggered action on the foreign key in the relation corresponding to the weak entity type, since a weak entity has an existence dependency on its owner entity.

§ This can be used for both ON UPDATE and ON DELETE.

Step 3: Mapping of Binary 1:1 Relationship Types

§ For each binary 1:1 relationship type *R* in the ER schema, identify the relations *S* and *T* that correspond to the entity types participating in *R*

§ There are three possible approaches:

- foreign key approach
- merged relationship approach
- crossreference or relationship relation approach

1. The foreign key approach

§ Choose one of the relations—*S*, say—and include as a foreign key in *S* the primary key of *T*.

§ It is better to choose an entity type with *total participation* in *R* in the role of *S*

§ Include all the simple attributes (or simple components of composite attributes) of the 1:1 relationship type *R* as attributes of *S*.

§ In our example, we map the 1:1 relationship type by choosing the participating entity type DEPARTMENT to serve in the role of *S* because its participation in the MANAGES relationship type is total

§ We include the primary key of the EMPLOYEE relation as foreign key in the DEPARTMENT relation and rename it Mgr_ssn.

§ We also include the simple attribute Start_date of the MANAGES relationship type in the DEPARTMENT relation and rename it Mgr_start_date

2. Merged relation approach:

§ merge the two entity types and the relationship into a single relation

§ This is possible when *both participations are total,* as this would indicate that the two tables will have the exact same number of tuples at all times.

3. Cross-reference or relationship relation approach:

§ set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types.

§ required for binary M:N relationships

- The relation R is called a relationship relation (or sometimes a lookup table), because each tuple in R represents a relationship instance that relates one tuple from S with one tuple from T
- The relation R will include the primary key attributes of S and T as foreign keys to S and T.
- The primary key of R will be one of the two foreign keys, and the other foreign key will be a unique key of R.
- The drawback is having an extra relation, and requiring an extra join operation when combining related tuples from the tables.

Step 4: Mapping of Binary 1:N Relationship Types

- For each regular binary 1:N relationship type *R*, identify the relation *S* that represents the participating entity type at the *N-side* of the relationship type.
- Include as foreign key in *S* the primary key of the relation *T* that represents the other entity type participating in *R*
- Include any simple attributes (or simple components of composite attributes) of the 1:N relationship type as attributes of *S*
- In our example, we now map the 1:N relationship types WORKS_FOR, CONTROLS, and SUPERVISION
- For WORKS_FOR we include the primary key Dnumber of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it Dno.
- For SUPERVISION we include the primary key of the EMPLOYEE relation as foreign key in the EMPLOYEE relation itself—because the relationship is recursive—and call it Super_ssn.

§ The CONTROLS relationship is mapped to the foreign key attribute Dnum of PROJECT, which references the primary key Dnumber of the DEPARTMENT relation.

Step 5: Mapping of Binary M:N Relationship Types

§ For each binary M:N relationship type

- Create a new relation S
- Include primary key of participating entity types as foreign key attributes in S
- Include any simple attributes of M:N relationship type

§ In our example, we map the M:N relationship type WORKS_ON by creating the relation WORKS_ON.We include the primary keys of the PROJECT and EMPLOYEE relations as foreign keys in WORKS_ON and rename them Pno and Essn, respectively.

§ We also include an attribute Hours in WORKS_ON to represent the Hours attribute of the relationship type.

§ The primary key of the WORKS_ON relation is the combination of the foreign key attributes {Essn, Pno}.

§ The propagate (CASCADE) option for the referential triggered action should be specified on the foreign keys in the relation corresponding to the relationship R, since each relationship instance has an existence dependency on each of the entities it relates. This can be used for both ON UPDATE and ON DELETE.

Step 6: Mapping of Multivalued Attributes

§ For each multivalued attribute

- Create a new relation
- Primary key of *R* is the combination of that *Atrribute* and *PK of the attached entity*
- If the multivalued attribute is composite, include its simple components

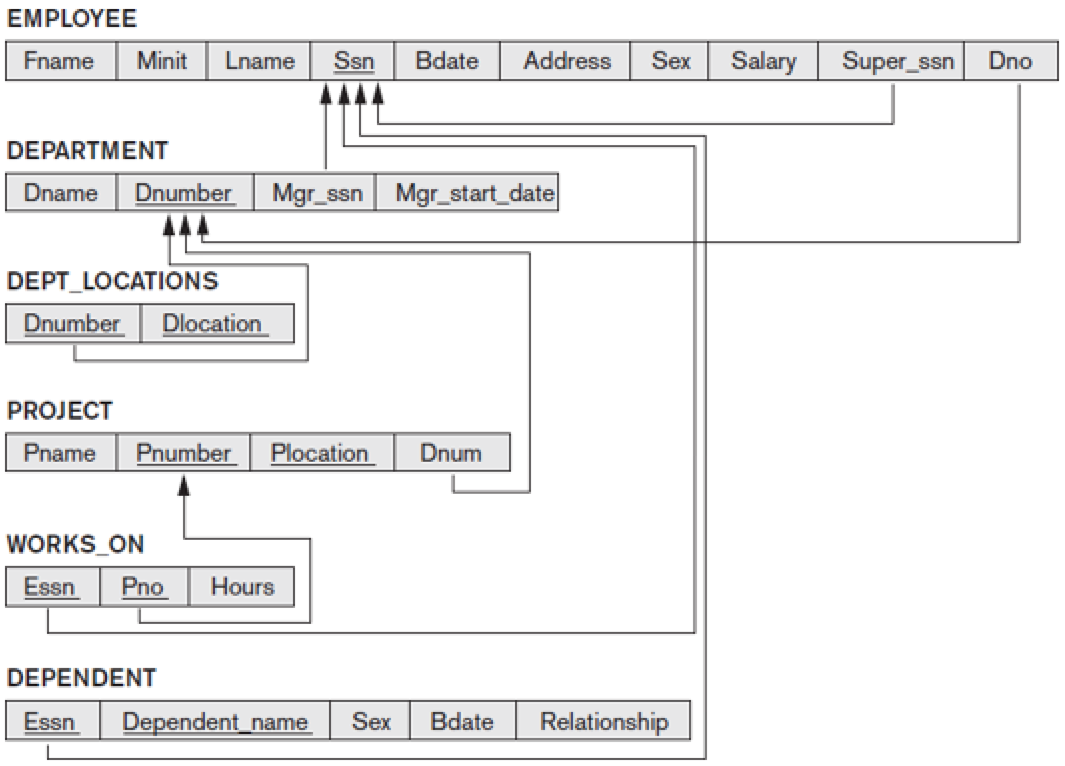
§ In our example, we create a relation DEPT_LOCATIONS

§ The attribute Dlocation represents the multivalued attribute LOCATIONS of DEPARTMENT, while Dnumber—as foreign key—represents the primary key of the DEPARTMENT relation.

§ The primary key of DEPT_LOCATIONS is the combination of {Dnumber, Dlocation}

§ A separate tuple will exist in DEPT_LOCATIONS for each location that a department has

§ The propagate (CASCADE) option for the referential triggered action should be specified on the foreign key in the relation *R* corresponding to the multivalued attribute for both ON UPDATE and ON DELETE.



Step 7: Mapping of *N*-ary Relationship Types

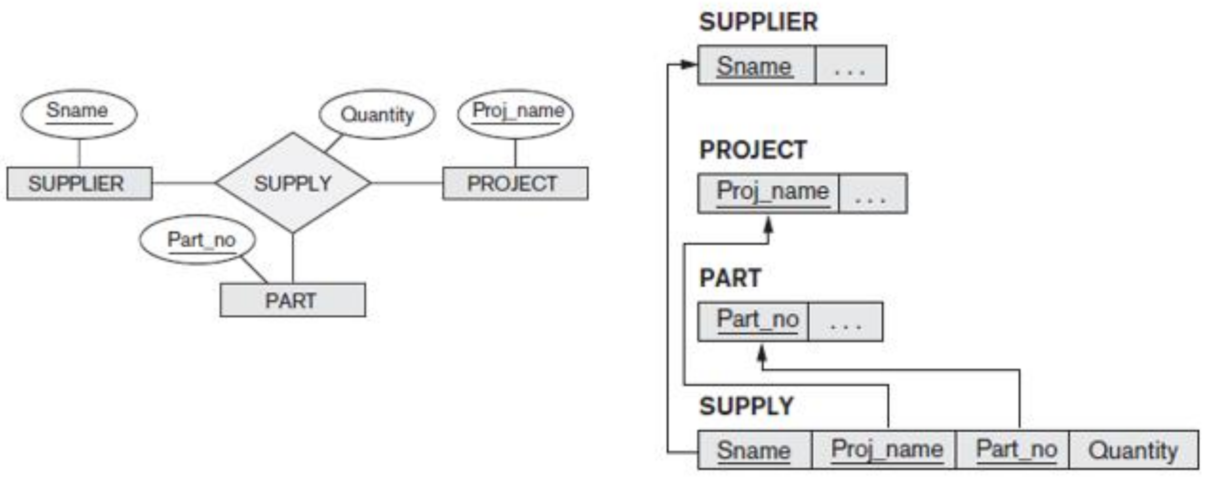
§ For each *n*-ary relationship type *R*

- Create a new relation *S* to represent *R*

- Include primary keys of participating entity types as foreign keys
- Include any simple attributes as attributes

§ The primary key of *S* is usually a combination of all the foreign keys that reference the relations representing the participating entity types.

§ For example, consider the relationship type SUPPLY.This can be mapped to the relation SUPPLY whose primary key is the combination of the three foreign keys {Sname, Part_no, Proj_name}.



(b)

Unary Relational Operations: SELECT, PROJECT, RENAME

<u>The SELECT Operation</u>

The SELECT operation denoted by σ (sigma) is used to select a subset of the tuples from a relation based on a selection condition. The selection condition acts as a filter that keeps only those tuples that satisfy a qualifying condition. Alternatively, we can consider the SELECT operation to *restrict* the tuples in a relation to only those tuples that satisfy the condition.

The SELECT operation can also be visualized as a *horizontal partition* of the relation into two sets of tuples—those tuples that satisfy the condition and are selected, and those tuples that do not satisfy the condition and are discarded.
In general, the select operation is denoted by

$$\sigma_{\text{<selection condition>}}(R)$$

where,

- the symbol σ is used to denote the select operator
- the selection condition is a Boolean (conditional) expression specified on the attributes of relation R
- tuples that make the condition true are selected
   - appear in the result of the operation
- tuples that make the condition false are filtered out
   - discarded from the result of the operation

The Boolean expression specified in <selection condition> is made up of a number of clauses of the form:

where

<attribute name> <comparison op> <constant value>

or

<attribute name> <comparison op> <attribute name>

<attribute name> is the name of an attribute of *R*,

<comparison op> is one of the operators {=, <, ≤, >, ≥, ≠}, and

<constant value> is a constant value from the attribute domain

Clauses can be connected by the standard Boolean operators *and*, *or*, and *not* to form a general selection condition

Examples:

1. Select the EMPLOYEE tuples whose department number is 4.

$$\sigma_{\text{DNO} = 4}(\text{EMPLOYEE})$$

2. Select the employee tuples whose salary is greater than \$30,000.

$$\sigma_{\text{SALARY} > 30,000}(\text{EMPLOYEE})$$

3. Select the tuples for all employees who either work in department 4 and make over \$25,000 per year, or work in department 5 and make over \$30,000

$$\sigma_{(\text{Dno=4 AND Salary>25000})\ \text{OR}\ (\text{Dno=5 AND Salary>30000})}(\text{EMPLOYEE})$$

The result of a SELECT operation can be determined as follows:

- The <selection condition> is applied independently to each individual tuple t in R
- If the condition evaluates to TRUE, then tuple *t* is selected.All the selected tuples appear in the result of the SELECT operation
- The Boolean conditions AND, OR, and NOT have their normal interpretation, as follows:

  - (cond1 AND cond2) is TRUE if both (cond1) and (cond2) are TRUE; otherwise,it is FALSE.
  - (cond1 OR cond2) is TRUE if either (cond1) or (cond2) or both are TRUE; otherwise, it is FALSE.
  - (NOT cond) is TRUE if cond is FALSE; otherwise, it is FALSE.

The SELECT operator is unary; that is, it is applied to a single relation. The degree of the relation resulting from a SELECT operation is the same as the degree of R.The number of tuples in the resulting relation is always less than or equal to the number of tuples in R. That is,

$$|\sigma_c(R)| \leq |R| \text{ for any condition } C$$

The fraction of tuples selected by a selection condition is referred to as the selectivity of the condition.

The SELECT operation is commutative; that is,

$$\sigma_{<cond1>}(\sigma_{<cond2>}(R)) = \sigma_{<cond2>}(\sigma_{<cond1>}(R))$$

Hence, a sequence of SELECTs can be applied in any order.we can always combine a cascade (or sequence) of SELECT operations into a single SELECT operation with a conjunctive (AND) condition; that is,

$\sigma_{<cond1>}(\sigma<cond2>(...(\sigma_{<condn>}(R)) ...)) = \sigma_{<cond1>} \text{ AND}_{<cond2>} \text{ AND} ... \text{ AND }_{<condn>}(R)$ In SQL, the SELECT condition is specified in the WHERE clause of a query.For example, the following operation:

$$\sigma_{Dno=4 \text{ AND } Salary>25000} (\text{EMPLOYEE})$$

would to the following SQL query:

```
SELECT * FROM EMPLOYEE WHERE Dno=4 AND Salary>25000;
```

The PROJECT Operation

The PROJECT operation denoted by π (pi) selects certain columns from the table and discards the other columns. Used when we are interested in only certain attributes of a relation. The result of the PROJECT operation can be visualized as a vertical partition of the relation into two relations:

- one has the needed columns (attributes) and contains the result of the operation
- the other contains

the discarded columns The general form of the PROJECT operation is

$$\pi_{\text{<attribute list>}}(R)$$

where

π (pi) - symbol used to represent the PROJECT operation,

<attributelist> - desired sublist of attributes from the attributes of relation R.

The result of the PROJECT operation has only the attributes specified in <attribute list> in the same order as they appear in the list. Hence, its degree is equal to the number of attributes in <attribute list>

Example :

1. To list each employee's first and last name and salary we can use the PROJECT operation as follows:

$$\pi_{\text{Lname, Fname, Salary}}(\text{EMPLOYEE})$$

If the attribute list includes only nonkey attributes of R, duplicate tuples are likely to occur. The result of the PROJECT operation is a set of distinct tuples, and hence a valid relation. This is known as duplicate elimination.For example, consider the following PROJECT operation:

$$\pi_{\text{gender, Salary}}(\text{EMPLOYEE})$$

The tuple <'F', 25000> appears only once in resulting relation even though this combination of values appears twice in the EMPLOYEE relation.

The number of tuples in a relation resulting from a PROJECT operation is always less than or equal to the number of tuples in *R.* Commutativity does not hold on PROJECT

$$\pi_{\text{<list1>}}(\pi_{\text{<list2>}}(R)) = \pi_{\text{<list1>}}(R)$$

as long as <list2> contains the attributes in <list1>; otherwise, the left-hand side is an incorrect expression.

In SQL, the PROJECT attribute list is specified in the SELECT clause of a query. For example, the following operation:

$$\pi_{\text{gender, Salary}}(\text{EMPLOYEE})$$

would correspond to the following SQL query:

SELECT DISTINCT gender, Salary FROM EMPLOYEE

Sequences of Operations and the RENAME Operation

For most queries, we need to apply several relational algebra operations one after the other. Either we can write the operations as a single relational algebra expression by nesting the operations, or we can apply one operation at a time and create intermediate result relations. In the latter case, we must give names to the relations that hold the intermediate results.

For example, to retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a SELECT and a PROJECT operation. We can write a single relational algebra expression, also known as an in-line expression, as follows:

$$\pi_{\text{Fname, Lname, Salary}}(\sigma_{\text{Dno}=5}(\text{EMPLOYEE}))$$

Alternatively, we can explicitly show the sequence of operations, giving a name to each intermediate relation, as follows:

$$\text{DEP5\_EMPS} \leftarrow \sigma_{\text{Dno}=5}(\text{EMPLOYEE})$$

$$\text{RESULT} \leftarrow \pi_{\text{Fname, Lname, Salary}}(\text{DEP5\_EMPS})$$

We can also use this technique to rename the attributes in the intermediate and result relations. To rename the attributes in a relation, we simply list the new attribute names in parentheses

$$\text{TEMP} \leftarrow \sigma_{\text{Dno}=5}(\text{EMPLOYEE})$$

$$R(\text{First\_name, Last\_name, Salary}) \leftarrow \pi_{\text{Fname, Lname, Salary}}(\text{TEMP})$$

If no renaming is applied, the names of the attributes in the resulting relation of a SELECT operation are the same as those in the original relation and in the same order.For a PROJECT operation with no renaming, the resulting relation has the same attribute names as those in the projection list and in the same order in which they appear in the list.

We can also define a formal RENAME operation—which can rename either the relation name or the attribute names, or both—as a unary operator.

The general RENAME operation when applied to a relation *R* of degree *n* is denoted by any of the following three forms:

1. $\rho_{S(B1, B2, ..., Bn)}(R)$ ρ (rho) – RENAME operator, S is new relation name, B1, B2, Bn are new attribute names

2. $\rho_S(R)$ S – new relation name

3. $\rho_{(B1, B2,.......Bn)}(R)$ B_1,B_2,…..B_n- new attribute names

The first expression renames both the relation and its attributes. Second renames the relation only and the third renames the attributes only.If the attributes of R are (A_1, A_2, ..., A_n) in that order, then each A_i is renamed as B_i.

(c)

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

Result ← π Fname, Minit, Lname, Address (
σ Dname='Research' (DEPARTMENT) ⋈ EMPLOYEE.Dno = DEPARTMENT.Dnumber (EMPLOYEE)
)

ii) List the names of all employees with 2 or more dependents.

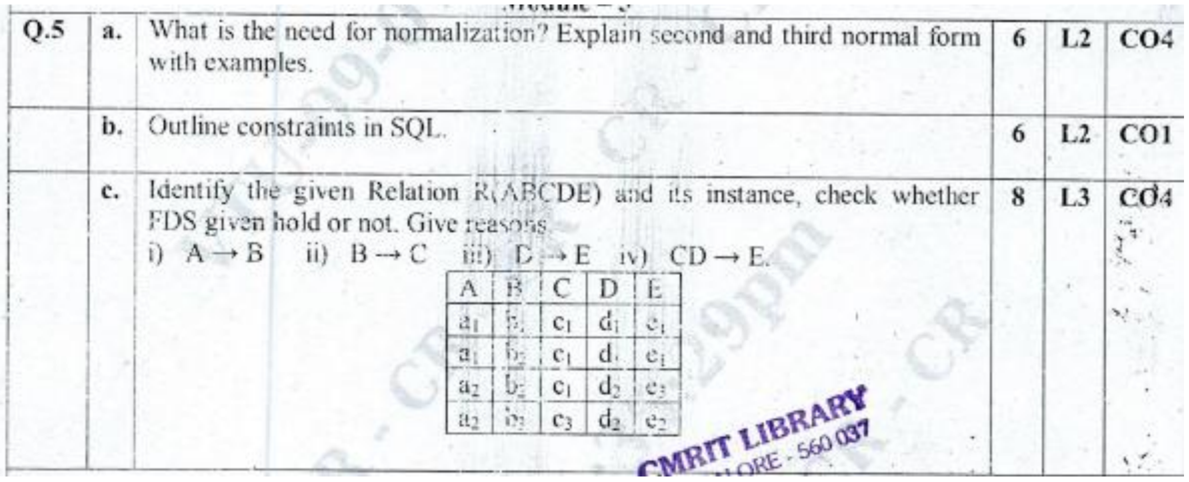
Dependent_Count ← γ Essn, COUNT(Dependent_name)→DepCount (DEPENDENT)
Result ← π Fname, Minit, Lname (
(σ DepCount ≥ 2 (Dependent_Count)) ⋈ EMPLOYEE.SSN = Dependent_Count.Essn
)

iii) Find the names of employees who work on all the projects controlled by department number

Proj_D ← π Pnumber (σ Dnum = 5 (PROJECT))
Emp_Proj ← π Essn, Pno (WORKS_ON)
Result ← π Fname, Minit, Lname (
EMPLOYEE ⋈ SSN = Essn (
(γ Essn (Emp_Proj ÷ Proj_D))
)
)

iv) List the names of employees who have no dependents.
Emp_No_Dep ← π SSN (EMPLOYEE) − π Essn (DEPENDENT)
Result ← π Fname, Minit, Lname (
σ SSN ∈ Emp_No_Dep (EMPLOYEE)
)

Q5)



Q.5	a.	What is the need for normalization? Explain second and third normal form with examples.	6	L2	CO4
	b.	Outline constraints in SQL.	6	L2	CO1
	c.	Identify the given Relation R(ABCDE) and its instance, check whether FDS given hold or not. Give reasons. i) A → B ii) B → C iii) D → E iv) CD → E.	8	L3	CO4

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_2	c_1	d_1	e_1
a_2	b_2	c_1	d_2	e_3
a_2	b_3	c_3	d_2	e_2

CMRIT LIBRARY
…LORE - 560 037

Solution

a) Need for Normalization
- To organize data in a database to reduce redundancy (repetition of data).
- To avoid anomalies in insertion, deletion, and updation.
- To ensure data integrity and efficient storage.
- To make queries faster and more consistent.

Second Normal Form (2NF)

Definition:

- A relation is in 2NF if:
    1. It is already in 1NF (no multivalued or repeating attributes).
    2. No partial dependency exists (i.e., no non-prime attribute depends only on part of a composite key).

Example:
Table: ENROLL(StudentID, CourseID, StudentName, CourseName)

- Primary Key: (StudentID, CourseID)
- Problem: StudentName depends only on StudentID, CourseName depends only on CourseID → partial dependency.

Solution (2NF):
Split into:

1. STUDENT(StudentID, StudentName)
2. COURSE(CourseID, CourseName)
3. ENROLL(StudentID, CourseID)

Third Normal Form (3NF)

Definition:

- A relation is in 3NF if:
    1. It is in 2NF.
    2. No transitive dependency exists (i.e., non-prime attribute should not depend on another non-prime attribute).

Example:
Table: EMP(EmpID, EmpName, DeptID, DeptName)

- Primary Key: EmpID
- Problem: DeptName depends on DeptID, and DeptID depends on EmpID → transitive dependency.

Solution (3NF):

1. EMP(EmpID, EmpName, DeptID)

2. DEPARTMENT(DeptID, DeptName)

b) In SQL, constraints are rules applied on table columns to maintain data accuracy and integrity. The main types of constraints are:

1. NOT NULL Constraint – Ensures that a column cannot have NULL values.
Example: Name VARCHAR(50) NOT NULL

2. UNIQUE Constraint – Ensures that all values in a column are unique.
Example: Email VARCHAR(100) UNIQUE

3. PRIMARY KEY Constraint – Uniquely identifies each record in a table. It combines NOT NULL and UNIQUE.
Example: PRIMARY KEY(StudentID)

4. FOREIGN KEY Constraint – Ensures referential integrity by linking one table to another.
Example: FOREIGN KEY(DeptID) REFERENCES Department(DeptID)

5. CHECK Constraint – Ensures that values in a column satisfy a specific condition.
Example: Age INT CHECK(Age >= 18)

6. DEFAULT Constraint – Provides a default value if no value is specified.
Example: Status VARCHAR(10) DEFAULT 'Active'

c)**Given:** Relation R(A,B,C,D,E)R(A,B,C,D,E)R(A,B,C,D,E) with instance (tuples):

1. (a1,b1,c1,d1,e1)(a_1,b_1,c_1,d_1,e_1)(a1,b1,c1,d1,e1)

2. (a1,b2,c1,d1,e1)(a_1,b_2,c_1,d_1,e_1)(a1,b2,c1,d1,e1)

3. (a2,b2,c1,d2,e3)(a_2,b_2,c_1,d_2,e_3)(a2,b2,c1,d2,e3)

4. (a2,b3,c3,d2,e2)(a_2,b_3,c_3,d_2,e_2)(a2,b3,c3,d2,e2)

Check the FDs:

1. A→B — Does NOT hold.
Counterexample: Tuples 1 and 2 have the same A=a1A=a1 but different B values (b1≠b2)

2. B→C— Holds (in this instance).
For repeated B: B=b2 appears in tuples 2 and 3, both have C=c1. Other B values occur once, so no

violation.

3. D→E — Does NOT hold.
Counterexample: Tuples 3 and 4 have the same D=d2 but different EEE values (e3=e2).

4. CD→E— Holds (in this instance).
Repeated (C,D): (c1,d1) occurs in tuples 1 and 2 with the same E=e1. Other (C,D) pairs are unique, so no violation.

So,

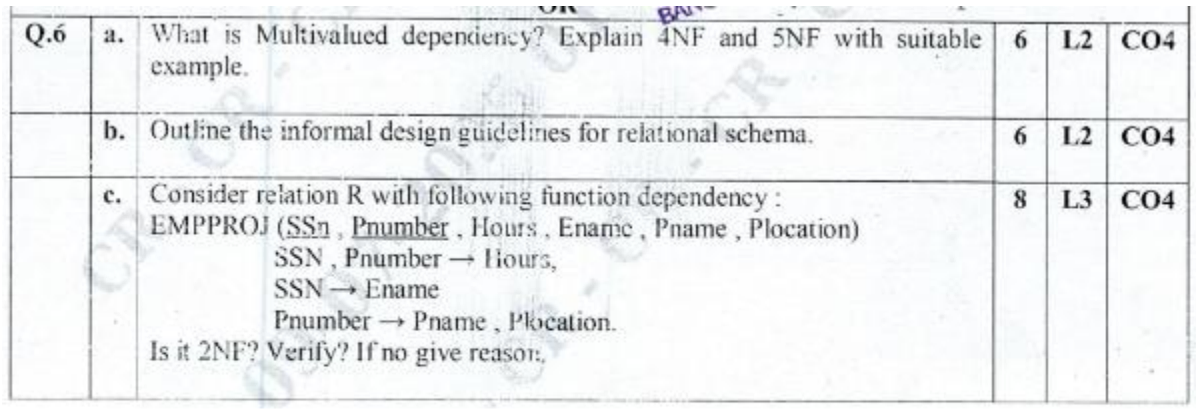
A→B: No;

B→C: Yes;

D→E: No;

CD→E: Yes — with reasons shown above.

Q6)



Q.6	a.	What is Multivalued dependency? Explain 4NF and 5NF with suitable example.	6	L2	CO4
	b.	Outline the informal design guidelines for relational schema.	6	L2	CO4
	c.	Consider relation R with following function dependency : EMPPROJ (SSn , Pnumber , Hours , Ename , Pname , Plocation) SSN , Pnumber → Hours, SSN → Ename Pnumber → Pname , Plocation. Is it 2NF? Verify? If no give reason.	8	L3	CO4

Solution

(a) A multivalued dependency (MVD) X ↠ Y in a relation R means that if two tuples agree on X, then their Y-values can be paired with the X-value independently of the other attributes. Informally, for a given X value there is a set of Y values and a set of Z values (other attributes) and every combination of Y and Z with that X should appear. An MVD is non-trivial if Y is not a subset of X and X ∪ Y ≠ all attributes.

Fourth Normal Form (4NF):
A relation is in 4NF if for every non-trivial MVD X ↠ Y, X is a superkey. 4NF removes independent multivalued facts that cause redundancy.
Example: Consider STUDENT(StudentID, Hobby, Language). If a student can have many hobbies and many spoken languages independently, we have StudentID ↠ Hobby and StudentID ↠ Language. Keeping all three attributes in one relation causes repetition (Cartesian combination of hobbies and languages). To achieve 4NF we decompose into:

- STUDENT_HOBBY(StudentID, Hobby)
- STUDENT_LANGUAGE(StudentID, Language)

Fifth Normal Form (5NF / PJNF):
A relation is in 5NF (Project-Join Normal Form) if every non-trivial join dependency in the relation is implied by the candidate keys. 5NF handles cases where lossless decomposition requires more than binary splits because tuples represent facts that are only valid as combinations of three (or more) components and cannot be captured by simpler FDs or MVDs.
Example (conceptual): Suppose SUPPLY(Supplier, Part, Project) contains only those triples where the supplier supplies a part for a particular project, and there is no simpler FD or MVD that explains the allowable combinations. If the valid triples can be expressed only by joining three binary relations (and that join dependency is not implied by keys), then to avoid redundancy we decompose into SUPPLIER_PART, SUPPLIER_PROJECT, PART_PROJECT. If the join dependency is not implied by keys, R must be decomposed to satisfy 5NF.

(b) When designing a relational schema informally follow these guidelines: use meaningful, stable primary keys and avoid composite keys unless necessary; keep attributes atomic (1NF) and remove repeating groups; normalize up to at least 3NF (remove partial and transitive dependencies) to avoid redundancy and update anomalies; enforce referential integrity with foreign keys; avoid storing derived attributes (compute when needed); minimize NULLs by sensible attribute design and defaults; choose appropriate data types and sizes; name tables and columns consistently and clearly; add constraints (NOT NULL, UNIQUE, CHECK) to enforce domain rules; consider expected queries and performance — apply denormalization only when justified for performance and after analysis; index columns used frequently in WHERE/JOIN to speed queries while being mindful of write cost. These informal rules produce a robust, maintainable schema.

c) Given: Relation EMPPROJ(SSN, Pnumber, Hours, Ename, Pname, Plocation) with FDs:

1. SSN, Pnumber → Hours
2. SSN → Ename

3. Pnumber → Pname, Plocation

Step 1: Identify candidate key(s).
The natural key for the assignment relationship is (SSN, Pnumber) because SSN,Pnumber functionally determines Hours, and together they identify a tuple of EMP assigned to PROJECT. There are no FDs that make a single attribute a key for whole relation, so assume primary key = (SSN, Pnumber).

Step 2: Check 2NF condition.
2NF requires the relation to be in 1NF and have no partial dependencies of non-prime attributes on a part of a composite key. Non-prime attributes are Hours, Ename, Pname, Plocation (attributes not part of key).

- SSN → Ename is a dependency of a non-prime attribute (Ename) on part of the composite key (SSN). This is a partial dependency.
- Pnumber → Pname, Plocation are dependencies of non-prime attributes (Pname, Plocation) on the other part of the composite key (Pnumber). These are also partial dependencies.
- SSN, Pnumber → Hours is OK because Hours depends on the whole key.

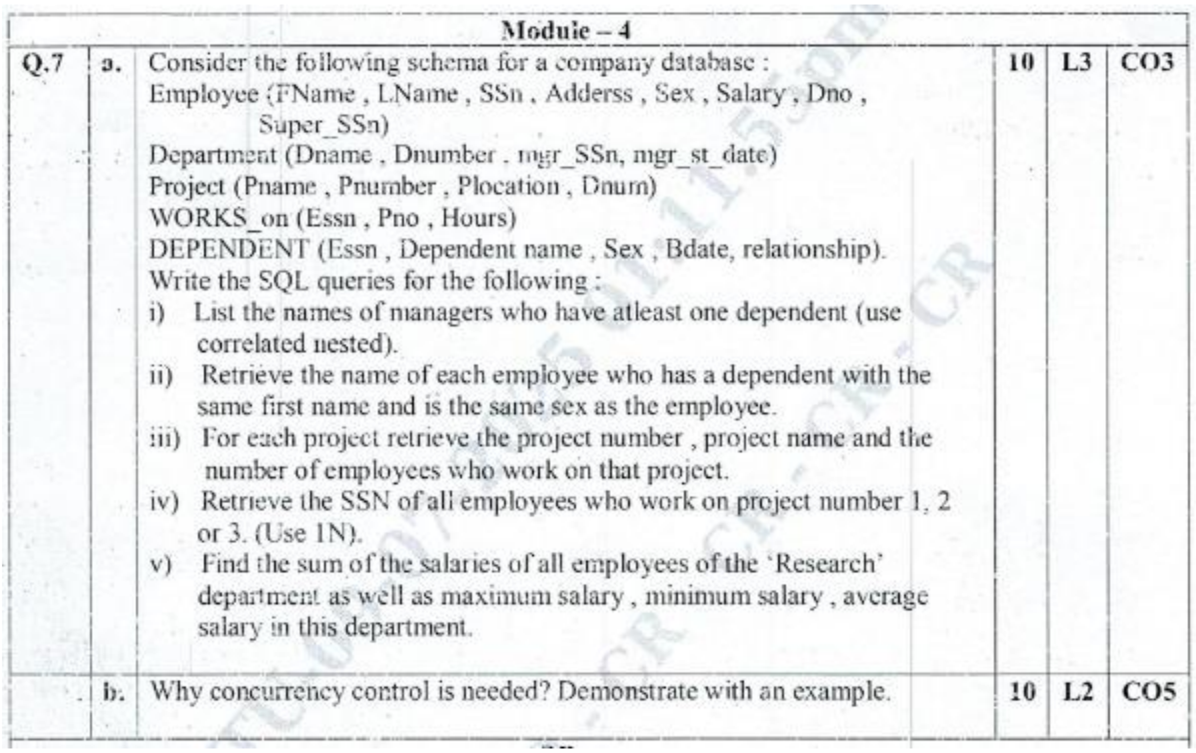
Because there are partial dependencies (SSN → Ename and Pnumber → Pname, Plocation), the relation is NOT in Second Normal Form (2NF).

Step 3: Decomposition to achieve 2NF (corrective decomposition).
Decompose into relations that remove partial dependencies while preserving data:

- EMP(SSN, Ename) — to store employee name determined by SSN.
- PROJECT(Pnumber, Pname, Plocation) — to store project details determined by Pnumber.
- EMPPROJ(SSN, Pnumber, Hours) — associative relation with composite key (SSN, Pnumber) and Hours.

After this decomposition each non-prime attribute depends on the whole key of its relation, so these relations satisfy 2NF (and in fact they are in 3NF because no transitive dependencies remain).

Q7)



Module – 4					
Q.7	a.	Consider the following schema for a company database : Employee (FName , LName , SSn , Adderss , Sex , Salary , Dno , Super_SSn) Department (Dname , Dnumber , mgr_SSn, mgr_st_date) Project (Pname , Pnumber , Plocation , Dnum) WORKS_on (Essn , Pno , Hours) DEPENDENT (Essn , Dependent name , Sex , Bdate, relationship). Write the SQL queries for the following : i) List the names of managers who have atleast one dependent (use correlated nested). ii) Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee. iii) For each project retrieve the project number , project name and the number of employees who work on that project. iv) Retrieve the SSN of all employees who work on project number 1, 2 or 3. (Use IN). v) Find the sum of the salaries of all employees of the 'Research' department as well as maximum salary , minimum salary , average salary in this department.	10	L3	CO3
	b.	Why concurrency control is needed? Demonstrate with an example.	10	L2	CO5

Solution

a) Schema given:

- Employee(FName, LName, SSN, Address, Sex, Salary, Dno, Super_SSN)
- Department(Dname, Dnumber, mgr_SSN, mgr_st_date)
- Project(Pname, Pnumber, Plocation, Dnum)
- Works_on(Essn, Pno, Hours)
- Dependent(Essn, Dependent_name, Sex, Bdate, relationship)

```
i) SELECT E.FName, E.LName
FROM Employee E
WHERE EXISTS (
   SELECT *
   FROM Dependent D
   WHERE E.SSN = D.Essn
)
AND E.SSN IN (
    SELECT mgr_SSN FROM Department
);
```

ii)
```
SELECT E.FName, E.LName
FROM Employee E
WHERE EXISTS (
    SELECT *
    FROM Dependent D
    WHERE E.SSN = D.Essn
      AND E.Sex = D.Sex
      AND E.FName = D.Dependent_name
);
```

iii)
```
SELECT P.Pnumber, P.Pname, COUNT(W.Essn) AS NumEmployees
FROM Project P
LEFT JOIN Works_on W ON P.Pnumber = W.Pno
GROUP BY P.Pnumber, P.Pname;
```

iv)
```
SELECT DISTINCT Essn
FROM Works_on
WHERE Pno IN (1, 2, 3);
```

v)
```
SELECT SUM(E.Salary) AS Total_Salary,
       MAX(E.Salary) AS Max_Salary,
       MIN(E.Salary) AS Min_Salary,
       AVG(E.Salary) AS Avg_Salary
FROM Employee E
JOIN Department D ON E.Dno = D.Dnumber
WHERE D.Dname = 'Research';
```

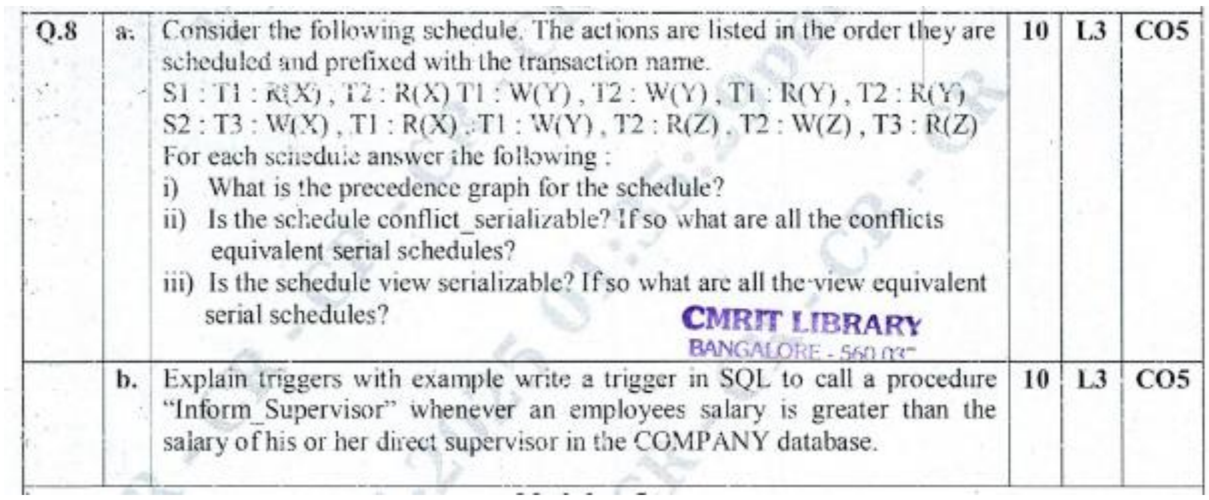
b) Concurrency control is required in a multi-user database system to ensure correctness and consistency of transactions when multiple users access or update data simultaneously. Without concurrency control, problems such as lost updates, dirty reads, uncommitted data dependency, and inconsistent retrievals can occur.

Example (Lost Update problem):
Suppose two transactions T1 and T2 try to update the salary of an employee simultaneously.

- T1 reads salary = 5000 and adds 500 → updates to 5500.
- T2 reads the same salary = 5000 and subtracts 200 → updates to 4800.
  Finally, the database shows 4800, and the increment done by T1 is lost.

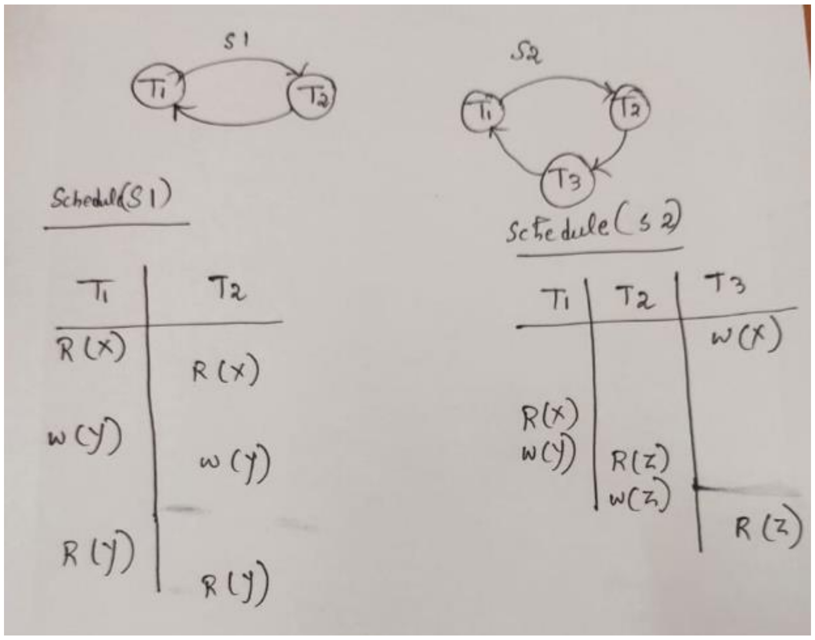
By using concurrency control techniques like locking, timestamp ordering, or two-phase locking, the database ensures that only one transaction updates the record at a time, preserving data integrity and correctness.

Q8)



Q.8	a.	Consider the following schedule. The actions are listed in the order they are scheduled and prefixed with the transaction name. S1 : T1 : R(X) , T2 : R(X) T1 : W(Y) , T2 : W(Y) , T1 : R(Y) , T2 : R(Y) S2 : T3 : W(X) , T1 : R(X) , T1 : W(Y) , T2 : R(Z) , T2 : W(Z) , T3 : R(Z) For each schedule answer the following : i) What is the precedence graph for the schedule? ii) Is the schedule conflict_serializable? If so what are all the conflicts equivalent serial schedules? iii) Is the schedule view serializable? If so what are all the view equivalent serial schedules? CMRIT LIBRARY BANGALORE - 560 037	10	L3	CO5
	b.	Explain triggers with example write a trigger in SQL to call a procedure "Inform_Supervisor" whenever an employees salary is greater than the salary of his or her direct supervisor in the COMPANY database.	10	L3	CO5

Solution
(a)



Schedule(S1)

T1	T2
R(x)	
	R(x)
w(y)	
	w(y)
R(y)	
	R(y)

Schedule(S2)

T1	T2	T3
		w(x)
R(x)		
w(y)		
	R(z)	
	w(z)	
		R(z)

Schedule S1

Precedence Graph:

The precedence graph contains a cycle: T1 -> T2 -> T1. Since there's a cycle, S1 is not conflict serializable. But S1 is view serializable. The view equivalent serial schedules are T1 -> T2 and T2 -> T1.

Conflict Serializability:

S1 is not conflict serializable because its precedence graph is having a cycle. The equivalent serial schedules are T1 -> T2 or T2 -> T1.

View Serializability:

S1 is view serializable because it can be converted to a serial schedule without changing the final state of the data items.

Two serial schedules T1 -> T2 and T2 -> T1

Schedule S2

Precedence Graph:

The precedence graph is having a cycle: T1 -> T2 -> T3 -> T1. So, S2 is not conflict serializable.

Conflict Serializability:

S2 is not conflict serializable because its precedence graph contains a cycle.

View Serializability:

S2 is view serializable because it can be transformed into a serial schedule. The view equivalent serial schedules :

T1 -> T2 -> T3, T1 -> T3 -> T2, T2 -> T1 -> T3, T2 -> T3 -> T1, T3 -> T1 -> T2, or T3 -> T2 -> T1.

(b)

A trigger is a stored procedure in a database that automatically invokes whenever a special event in the database occurs. By using SQL triggers, developers can automate tasks, ensure data consistency, and keep accurate records of database activities. For example, a trigger can be invoked when a row is inserted into a specified table or when specific table columns are updated.

In simple words, a trigger is a collection of SQL statements with particular names that are stored in system memory. It belongs to a specific class of stored procedures that are automatically invoked in response to database server events. Every trigger has a table attached to it.

Syntax

```
create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
FOR EACH ROW
BEGIN
END;
```

<u>Example: Prevent Table Deletions</u>

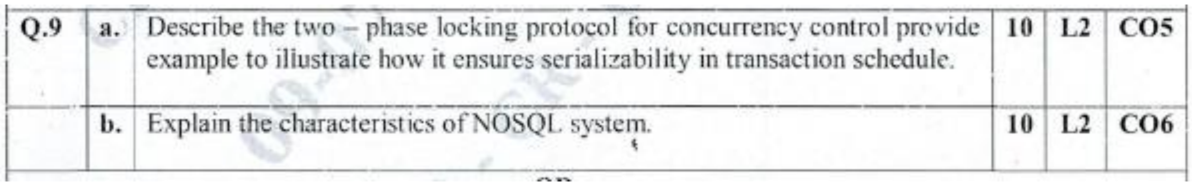
```
CREATE TRIGGER prevent_table_creation
ON DATABASE
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
```

```
 BEGIN
   PRINT 'you can not create, drop and alter table in this database';
   ROLLBACK;
END;
```

<u>Program</u>

```
CREATE PROCEDURE Inform_Supervisor(empSSN CHAR(9))
BEGIN
END;
DELIMITER //
CREATE TRIGGER Check_Salary_Increase
AFTER UPDATE ON Employee
FOR EACH ROW
BEGIN
DECLARE super_salary DECIMAL(10,2);
SELECT Salary INTO super_salary
FROM Employee
WHERE SSN = NEW.Super_SSN;
IF NEW.Salary > super_salary THEN
CALL Inform_Supervisor(NEW.SSN);
END IF;
END;
//
DELIMITER ;
```

Q9)



Q.9	a.	Describe the two – phase locking protocol for concurrency control provide example to illustrate how it ensures serializability in transaction schedule.	10	L2	CO5
	b.	Explain the characteristics of NOSQL system.	10	L2	CO6

Solution

(a)

The concept of locking data items is one of the main techniques used for controlling the concurrent execution of transactions. A lock is a variable associated with a data item in the database. Generally there is a lock for each data item in the database.

A lock describes the status of the data item with respect to possible operations that can be applied to that item. It is used for synchronizing the access by concurrent transactions to the database items.

§ A transaction locks an object before using it

§ When an object is locked by another transaction, the requesting transaction must wait

This Two-phase locking (2PL) protocol divides the execution phase of a transaction into three parts.

<u>In the first part,</u> when the transaction starts executing, it seeks permission for the locks it requires.

It is a concurrency control method that guarantees serializability

<u>The second part</u> is where the transaction acquires all the locks.

<u>As soon as the transaction releases its first lock, the third phase starts</u>.

•In this phase, the transaction cannot demand any new locks.

•It only releases the acquired locks.

•The protocol utilizes locks, applied by a transaction to data, which may block other transactions from accessing the same data during the transaction's life.

A transaction is said to follow the two-phase locking protocol if all locking operations (read_lock, write_lock) precede the first unlock operation in the transaction

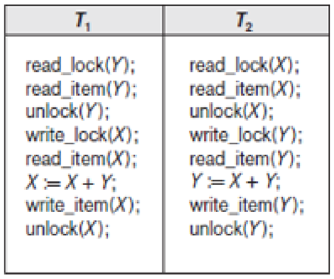
§ Such a transaction can be divided into two phases:

§ Expanding or growing (first) phase, during which new locks on items can be acquired but none can be released

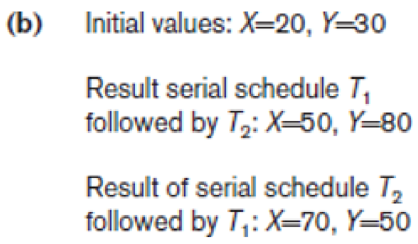
§ Shrinking (second) phase, during which existing locks can be released but no new locks can be acquired

§ If lock conversion is allowed, then upgrading of locks (from read-locked to write-locked) must be done during the expanding phase, and downgrading of locks (from write-locked to read-locked) must be done in the shrinking phase.

(a)



T_1	T_2
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
unlock(Y);	unlock(X);
write_lock(X);	write_lock(Y);
read_item(X);	read_item(Y);
$X := X + Y$;	$Y := X + Y$;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

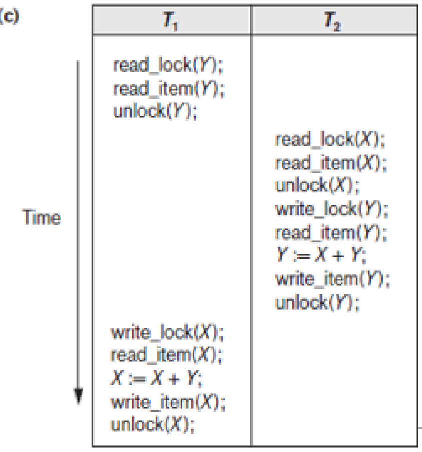


(b) Initial values: X=20, Y=30

Result serial schedule T_1 followed by T_2: X=50, Y=80

Result of serial schedule T_2 followed by T_1: X=70, Y=50

(c)



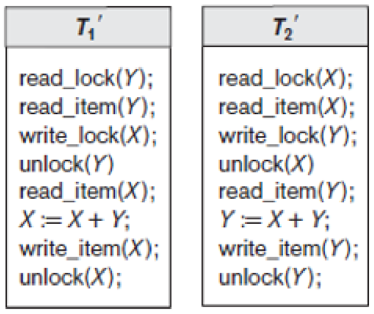
T_1	T_2
read_lock(Y);	
read_item(Y);	
unlock(Y);	
	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	$Y := X + Y$;
	write_item(Y);
	unlock(Y);
write_lock(X);	
read_item(X);	
$X := X + Y$;	
write_item(X);	
unlock(X);	

Transactions T1 and T2 in Figure 22.3(a) do not follow the two-phase locking protocol because the write_lock(X) operation follows the unlock(Y) operation in T1, and similarly the write_lock(Y) operation follows the unlock(X) operation in T2. so not following the rule of growing phase (only can acquire lock, cant release) and shrinking phase (only can release lock, cant acquire).

Figure 21.3 Transactions that do not obey two-phase locking (a) Two transactions T1 and T2 (b) Results of possible serial schedules of T1 and T2 (c) A nonserializable schedule S that uses locks

- If we enforce two-phase locking, the transactions can be rewritten as T1' and T2' as shown in Figure 22.4.
- Now, the schedule shown in Figure 22.3(c) is not permitted for T1_ and T2_ (with their modified order of locking and unlocking operations) under the rules of locking because T1_ will issue its write_lock(X) before it unlocks item Y; consequently, when T2_ issues its read_lock(X), it is forced to wait until T1_ releases the lock by issuing an unlock (X) in the schedule.

Here during growing phase only acquiring locks, not releasing any and during shrinking phase only releasing locks, not acquiring any lock, so satisfies 2PL property.



T_1'	T_2'
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
write_lock(X);	write_lock(Y);
unlock(Y)	unlock(X)
read_item(X);	read_item(Y);
$X := X + Y$;	$Y := X + Y$;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

Figure 22.4
Transactions T_1' and T_2', which are the same as T_1 and T_2 in Figure 22.3, but follow the two-phase locking protocol. Note that they can produce a deadlock.

- **If every transaction in a schedule follows the two-phase locking protocol, schedule guaranteed to be serializable**

(b) Characteristics of NoSQL System

1 - Flexible Data Model: NoSQL databases allow for flexible data models, meaning you can store unstructured, semi structured, and structured data without the need to define a rigid schema.

2) Scalability: NoSQL databases are designed to scale horizontally, which means you can easily add more servers to handle increased load, making them suitable for handling large amounts of data and high traffic.

3) High Performance: NoSQL databases are optimized for high performance and can deliver low latency responses, making them ideal for real time applications and large scale systems.

4) Replication and Fault Tolerance: NoSQL databases typically support automatic data replication and have built in fault tolerance mechanisms, ensuring data durability and high availability even in the event of server failures.

5) Distributed Computing: NoSQL databases are designed for distributed computing environments, allowing data to be distributed across multiple nodes in a cluster, which improves data access speed and fault tolerance.

6) Schema less Design: NoSQL databases do not require a fixed schema, allowing for dynamic and evolving data structures, which is particularly useful in scenarios where the data model may change frequently.

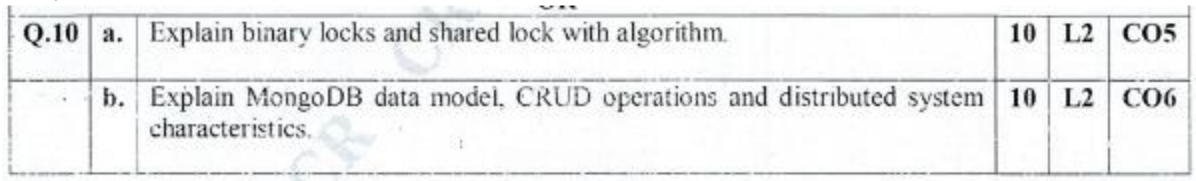
7) Support for Big Data: NoSQL databases are well suited for handling large volumes of data, making them a popular choice for big data applications and analytics.

8) Automatic Data Sharding: Many NoSQL databases support automatic data sharding, which allows data to be partitioned and distributed across multiple servers, enabling horizontal scaling and improved performance.

9) Developer friendly: NoSQL databases are often developer friendly, offering flexible APIs, query languages, and data manipulation tools that simplify application development and integration.

10) Variety of Data Models: NoSQL databases support various data models such as key value stores, document stores, column family stores, and graph databases, providing flexibility in choosing the best data model for specific use cases.

Q10)



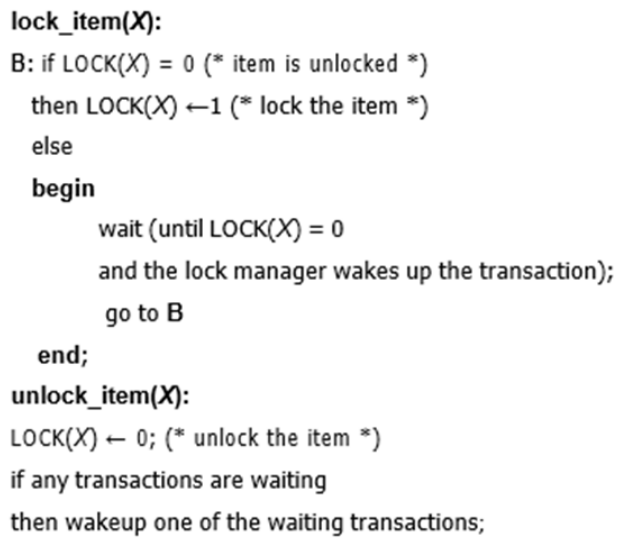
Q.10	a.	Explain binary locks and shared lock with algorithm.	10	L2	CO5
	b.	Explain MongoDB data model, CRUD operations and distributed system characteristics.	10	L2	CO6

Solution

(a)

<u>Binary Lock</u>

§A binary lock can have two states or values: locked and unlocked (or 1 and 0).
§If the value of the lock on X is 1, item X cannot be accessed by a database operation that requests the item
§If the value of the lock on X is 0, the item can be accessed when requested, and the lock value is changed to 1
§We refer to the current value (or state) of the lock associated with item X as lock(X).
§Two operations, lock_item and unlock_item, are used with binary locking.
§If LOCK(X) = 1, the transaction is forced to wait.
§If LOCK(X) = 0, it is set to 1 (the transaction locks the item) and the transaction is allowed to access item X
§When the transaction is through using the item, it issues an unlock_item(X) operation, which sets LOCK(X) back to 0 (unlocks the item) so that X may be accessed by other transactions
§Hence, a binary lock enforces mutual exclusion on the data item.

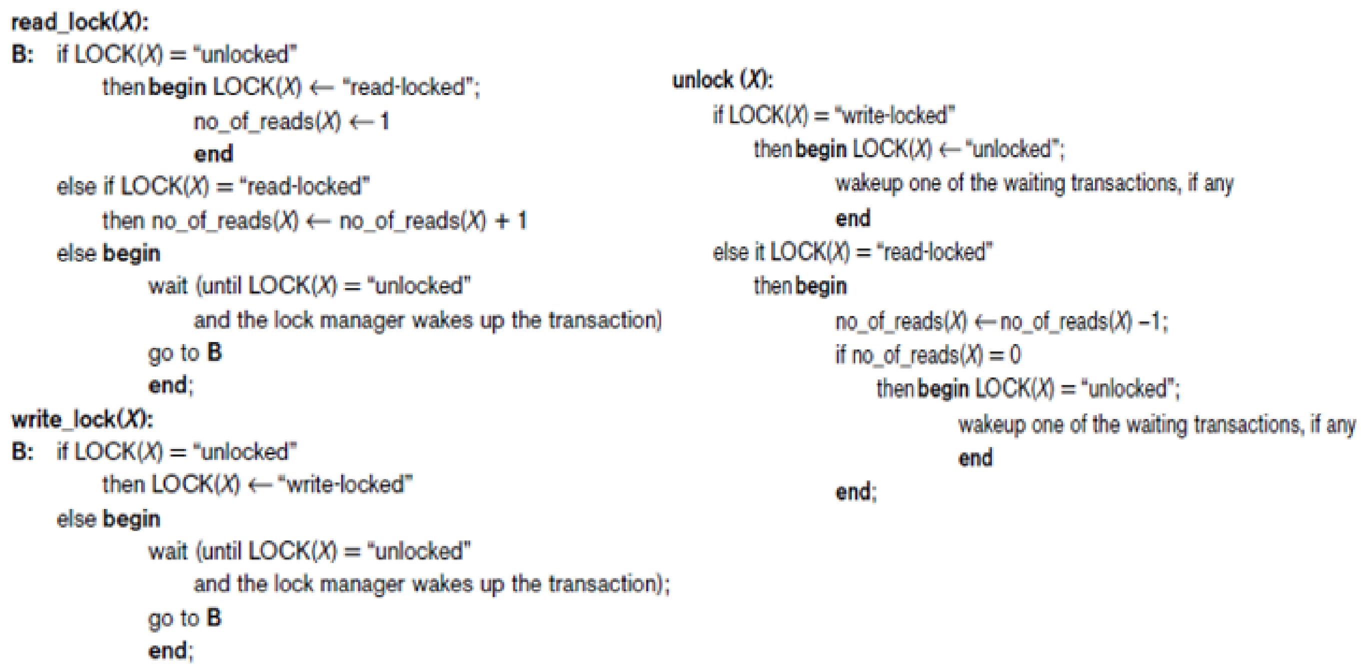


```
lock_item(X):
B: if LOCK(X) = 0 (* item is unlocked *)
   then LOCK(X) ←1 (* lock the item *)
   else
   begin
         wait (until LOCK(X) = 0
         and the lock manager wakes up the transaction);
          go to B
    end;
unlock_item(X):
LOCK(X) ← 0; (* unlock the item *)
if any transactions are waiting
then wakeup one of the waiting transactions;
```

<u>Shared/Exclusive (or Read/Write) Locks</u>

§binary locking scheme is too restrictive for database items because at most, one transaction can hold a lock on a given item.

If we want to enforce following rule:
•should allow several transactions to access the same item X if they all access X for reading purposes only
•
•if a transaction is to write an item X, it must have exclusive access to X
Then, a different type of lock called a multiple-mode lock can be usedà
It is called shared/exclusive or read/write locks—there are three locking operations:
•read_lock(X)
•write_lock(X)
unlock(X)



```
read_lock(X):
B:  if LOCK(X) = "unlocked"
        then begin LOCK(X) ← "read-locked";
                no_of_reads(X) ← 1
                end
    else if LOCK(X) = "read-locked"
        then no_of_reads(X) ← no_of_reads(X) + 1
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction)
            go to B
            end;
write_lock(X):
B:  if LOCK(X) = "unlocked"
        then LOCK(X) ← "write-locked"
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
```

```
unlock (X):
    if LOCK(X) = "write-locked"
        then begin LOCK(X) ← "unlocked";
                wakeup one of the waiting transactions, if any
                end
    else it LOCK(X) = "read-locked"
        then begin
                no_of_reads(X) ← no_of_reads(X) −1;
                if no_of_reads(X) = 0
                    then begin LOCK(X) = "unlocked";
                            wakeup one of the waiting transactions, if any
                            end
                end;
```

(b)

MongoDB Data Model

MongoDB is a NoSQL, document-oriented database. Instead of storing data in tables and rows (like SQL), it stores it in collections and documents.

- Database → Like an SQL database, a container for collections.
- Collection → Like a table in SQL, it stores multiple documents.

- Document → Basic unit of data, stored in JSON-like format (BSON = Binary JSON).

Example document:

```
{
  "_id": 1,
  "name": "Alice",
  "age": 22,
  "address": {
    "city": "Delhi",
    "pincode": 110001
  },
  "skills": ["Python", "MongoDB"]
}
```

Features of the MongoDB data model:

- Schema-less → Different documents in the same collection can have different fields.
- Embedded documents → Nested JSON structures for fast reads.
- Arrays → Store multiple values in a field (skills, tags, etc.).
- References → Linking across collections (similar to joins).

---

2. MongoDB CRUD Operations

CRUD = Create, Read, Update, Delete

a) Create

Insert new documents.

```
db.users.insertOne({ name: "Alice", age: 22 })
db.users.insertMany([
  { name: "Bob", age: 25 },
  { name: "Charlie", age: 30 }
])
```

b) Read

Query documents with filters.

```
db.users.find()                         // all documents
db.users.find({ age: 22 })              // filter
db.users.find({ age: { $gt: 20 } })     // greater than
db.users.find({ name: "Alice" }, { age: 1, _id: 0 }) // projection
```

c) Update

Modify existing documents.

```
db.users.updateOne({ name: "Alice" }, { $set: { age: 23 } })
db.users.updateMany({ age: { $lt: 25 } }, { $inc: { age: 1 } })
```

d) Delete

Remove documents.

```
db.users.deleteOne({ name: "Charlie" })
db.users.deleteMany({ age: { $gt: 30 } })
```

---

3. Distributed System Characteristics

MongoDB can be deployed as a distributed system (replica sets & sharding). Distributed systems in general have the following key characteristics:

1. Scalability
    - System can grow by adding more machines (horizontal scaling).
    - MongoDB supports sharding (partitioning data across servers).

2. Fault Tolerance & High Availability
    - System keeps working even if some nodes fail.
    - MongoDB uses replica sets (multiple copies of data).
3. Transparency
    - Distribution is hidden from the user (location, replication, migration).
    - User queries collections as if it’s a single database.
4. Concurrency
    - Multiple users/processes can access data at the same time.
    - MongoDB ensures consistency via read/write concerns and locks.
5. Consistency Models (CAP Theorem)
    - In distributed databases, you must balance:
        - Consistency (all nodes see same data at the same time)
        - Availability (system always responds)
        - Partition tolerance (system works even if network splits).
    - MongoDB usually provides CP but can be tuned for more availability.
6. Load Balancing
    - Requests are distributed across multiple servers for efficiency.