Modified

CBCS SCHEME

USN | | | | | | | | | | |

BCS403

Fourth Semester B.E./B.Tech. Degree Examination, June/July 2024
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.	Define database. Elaborate component modules of DBMS and their interactions.	10	L2	CO1
	b.	Describe the three-schema architecture. Why do we need mappings among schema levels?	06	L2	CO1
	c.	Explain the difference between logical and physical data independence.	04	L2	CO1
		OR			
Q.2	a.	Draw an ER diagram for an COMPANY database with employee, department, project as strong entities and dependent as weak entity. Specify the constraints, relationships and ratios in the ER diagram.	10	L3	CO3
	b.	Define the following terms with example for each using ER notations: Entity, attribute, composite attribute, multivalued attribute, participation role.	10	L3	CO3
		Module – 2			
Q.3	a.	Discuss the update operations and dealing with constraint violations with suitable examples.	08	L2	CO2
	b.	Illustrate the relational algebra operators with examples for select and project operation.	06	L2	CO2
	c.	Discuss the characteristics of relations that make them different from ordinary table and files.	06	L2	CO2
		OR			
Q.4	a.	Perform (i) Student ∪ instructor (ii) Student ∩ Instructor (iii) Student – Instructor (iv) Instructor – Student on the following tables: Student Fname | Lname Susan | Yao Ramesh | Shah Johnny | Kohler Barbara | Jones Amy | Ford Jimmy | Wang Ernest | Gilbert Instructor Fname | Lname John | Smith Ricardo | Browne Susan | Mao Francis | Johnson Ramesh | Shah	04	L3	CO2
	b.	Consider the following relational database schema and write the queries in relational algebra expressions: EMP(Eno, Ename, Salary, Address, Phone, DNo) DEPT(DNo, Dname, DLoc, MgrEno) DEPENDENT(Eno, Dep_Name, Drelation, Dage) (i) List all the employees who reside in 'Belagavi'. (ii) List all the employees who earn salary between 30000 and 40000 (iii) List all the employees who work for the 'Sales' department (iv) List all the employees who have at least one daughter (v) List the department names along with the names of the managers	10	L3	CO2

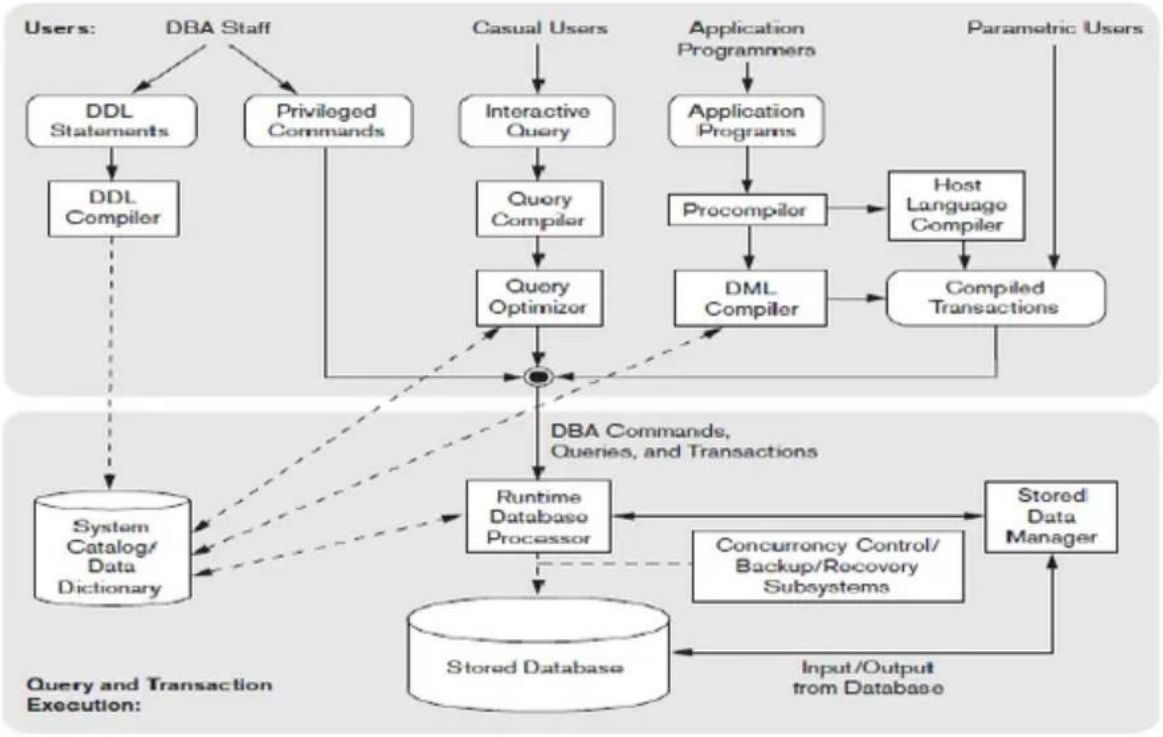
	c.	Consider the two tables T_1 and T_2 shown below: T_1 P | Q | R 10 | a | 5 15 | b | 8 25 | a | 6 T_2 A | B | C 10 | b | 6 25 | c | 3 10 | b | 5 Show the results of the following operations: (i) $T_1 \bowtie_{T_1.P=T_2.A} T_2$ (ii) $T_1 \bowtie_{T_1.Q=T_2.B} T_2$ (iii) $T_1 \bowtie_{(T_1.P=T_2.A \text{ AND } T_1.R=T_2.C)} T_2$	06	L3	CO2
		Module – 3			
Q.5	a.	Discuss the informal design guidelines for relation schema design.	08	L2	CO4
	b.	Define 1NF, 2NF, and 3NF with examples.	06	L2	CO4
	c.	Write the syntax for INSERT, UPDATE and DELETE statements in SQL and explain with suitable examples.	06	L2	CO3
		OR			
Q.6	a.	Discuss insertion, deletion and modification anomalies. Why are they considered bad? Illustrate with examples.	10	L2	CO3
	b.	Illustrate the following with suitable examples: (i) Datatypes in SQL (ii) Substring Pattern Matching in SQL.	10	L2	CO3
		Module – 4			
Q.7	a.	Consider the following relations: Student(Snum, Sname, Branch, level, age) Class(Cname, meet_at, room, fid) Enrolled(Snum, Cname) Faculty(fid, fname, deptid) Write the following queries in SQL. No duplicates should be printed in any of the answers. (i) Find the names of all Juniors (level = JR) who are enrolled in a class taught by I. Teach. (ii) Find the names of all classes that either meet in room R128 or have five or more students enrolled. (iii) For all levels except JR, print the level and rthe average age of students for that level. (iv) For each faculty member that has taught classes only in room R128, print the faculty member's name and the total number of classes she or he has taught. (v) Find the names of students not enrolled in any class.	10	L3	CO3
	b.	What do understand by correlated Nested Queries in SQL? Explain with suitable example.	04	L2	CO3
	c.	Discuss the ACID properties of a database transaction.	06	L2	CO4
		OR			
Q.8	a.	What are the views in SQL? Explain with examples.	04	L3	CO5
	b.	In SQL, write the usage of GROUP BY and HAVING clauses with suitable examples.	06	L2	CO3
	c.	Discuss the types of problems that may encounter with transactions that run concurrently.	10	L2	CO5

		Module – 5			
Q.9	a.	What is the two phase locking protocol? How does it Guarantee serializability.	06	L2	CO5
	b.	Describe the wait-die and wound-wait protocols for deadlock prevention.	08	L2	CO5
	c.	List and explain the four major categories of NOSQL system.	06	L2	CO3
		OR			
Q.10	a.	What is Multiple Granularity locking? How is it implemented using intension locks? Explain.	10	L2	CO5
	b.	Discuss the following MongoDB CRUD operations with their formats: (i) Insert (ii) Delete (iii) Read	06	L2	CO4
	c.	Briefly discuss about Neo4j data model.	04	L2	CO4

* * * * *

		Module – 1	M	L	C
Q.1	a.	Define database. Elaborate component modules of DBMS and their interactions.	10	L2	CO1
	b.	Describe the three-schema architecture. Why do we need mappings among schema levels?	06	L2	CO1
	c.	Explain the difference between logical and physical data independence.	04	L2	CO1

Q.1.a. Define database. Elaborate component modules of DBMS and their interactions (10M)



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

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

Component Modules of DBMS

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

1. Query Processor

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

Components

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

2. Storage Manager

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

Components

a) Authorization and Integrity Manager

- Checks user access privileges.
- Enforces integrity constraints.

b) Transaction Manager

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

c) Buffer Manager

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

d) File Manager

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

3. Database Files

Database files contain the actual data stored in the database.

Example

- Student records
- Employee details
- Project information

4. Data Dictionary (DBMS Catalog)

The data dictionary stores metadata about the database such as:

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

It helps the DBMS process queries efficiently.

Interaction of Components

1. Users submit SQL queries through application programs.
2. The **Query Processor** interprets and optimizes the query.

3. The **Query Evaluation Engine** sends requests to the Storage Manager.
4. The **Storage Manager** coordinates with:
    - Authorization & Integrity Manager
    - Transaction Manager
    - Buffer Manager
    - File Manager
5. Data is retrieved from database files.
6. The processed result is returned to the user.

Q.1.b. Describe the three-schema architecture. why do we need mapping among schema levels? (10M)

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

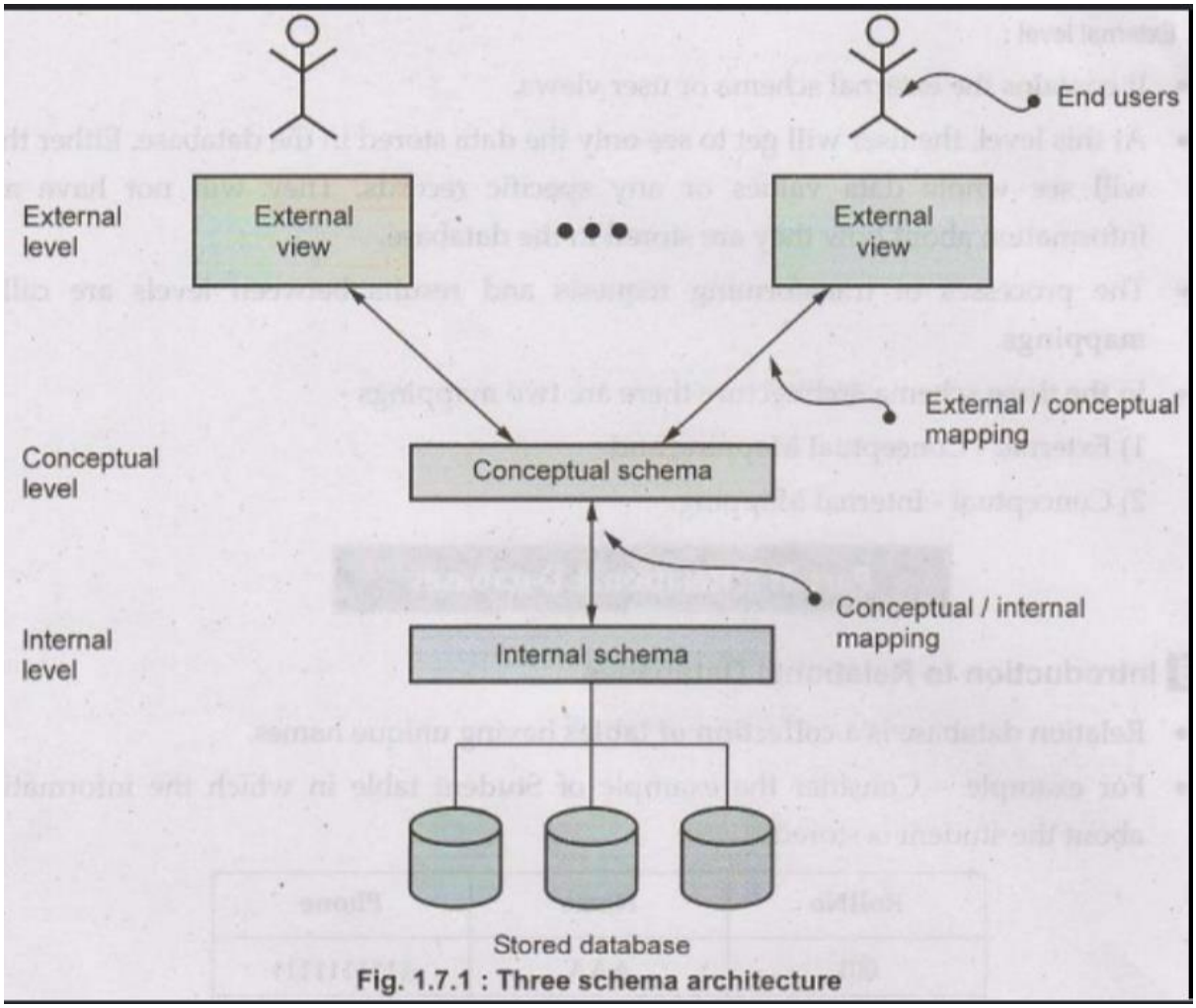


Fig. 1.7.1 : Three schema architecture

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.1.c. Explain the difference between logical and physical data independence. (4M)

Logical Data Independence (3 marks)	**Physical Data Independence (3 marks)**
Involves changes at conceptual level without affecting external schema	Involves changes at physical level without affecting conceptual schema
Deals with changes in logical structure (like adding/removing tables, relationships)	Deals with changes in physical storage methods and access paths
More difficult to achieve as it may involve restructuring of database	Easier to achieve as it only involves internal storage modifications
Protects external user views from changes in conceptual design	Protects database logical design from changes in physical storage

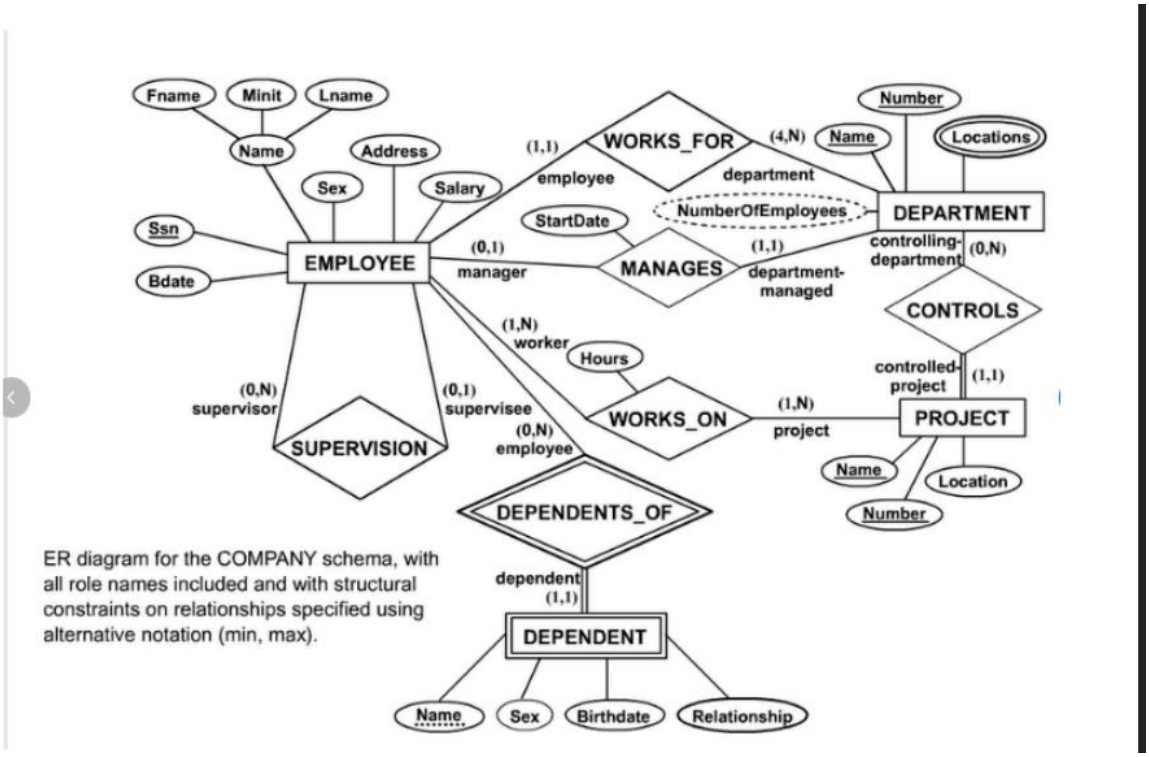
Example:

- Logical: Adding new columns to tables without affecting existing applications
- Physical: Changing from sequential to random file organization without impacting database logic

Q.2	a.	Draw an ER diagram for an COMPANY database with employee, department, project as strong entities and dependent as weak entity. Specify the constraints, relationships and ratios in the ER diagram.	10	L3	CO3
	b.	Define the following terms with example for each using ER notations: Entity, attribute, composite attribute, multivalued attribute, participation role.	10	L3	CO3

Q.2	a.	Draw an ER diagram for an COMPANY database with employee, department, project as strong entities and dependent as weak entity. Specify the constraints, relationships and ratios in the ER diagram.	10	L3	CO3

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.



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.

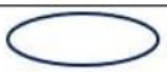
- One employee may have several dependents.
- Each dependent belongs to exactly one employee.

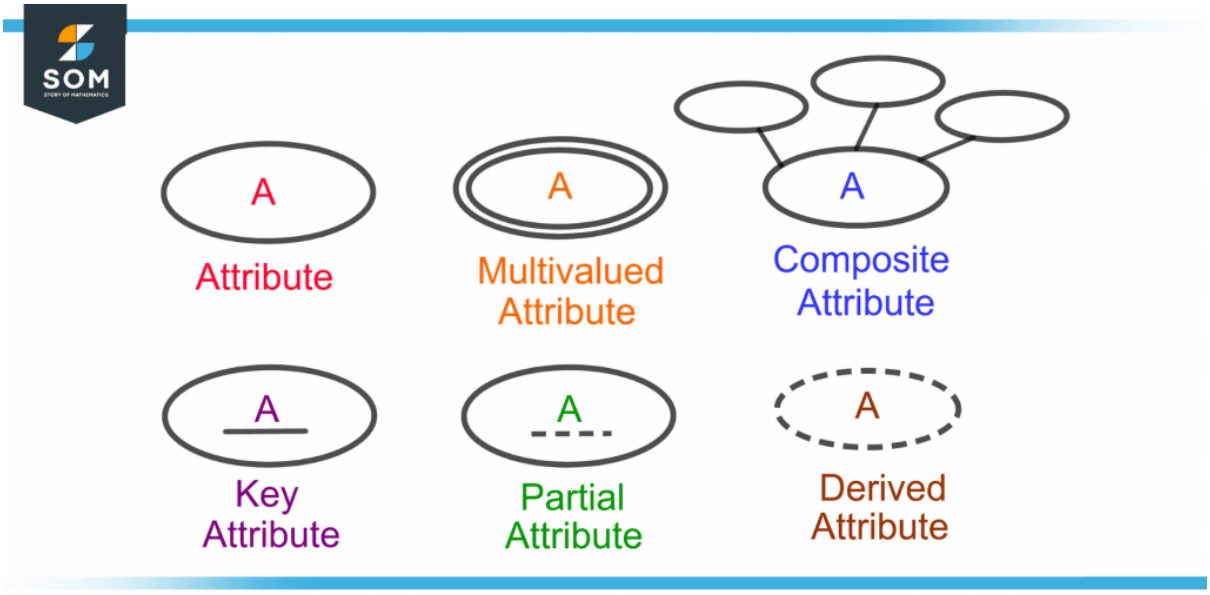
Cardinality Ratio

1:N1:N1:N

Identifying relationship since DEPENDENT is a weak entity.

	b.	Define the following terms with example for each using ER notations: Entity, attribute, composite attribute, multivalued attribute, participation role.	**10**	**L3**	**CO3**

Symbol Name	Shape	Representation
Rectangle		**Entity** example student,teacher,school
Ellipses		**Attributes** example student rollno, name,dob
Diamond		**Relationship** like student with course
Lines		Use for **link** between entity,attributes,relationship
Double Ellipse		**Multivalued Attributes** for example a person can have more than one mobile no
Double Rectangle		**Weak Entity** means entity cannot be uniquely identified by its own attributes



1. Entity

Definition

An **entity** is a real-world object, person, place, event, or concept that has an independent existence and can be uniquely identified.

Example

STUDENT with attributes USN, Name, and Branch.

ER Notation

(Rectangle represents an Entity)

2. Attribute

Definition

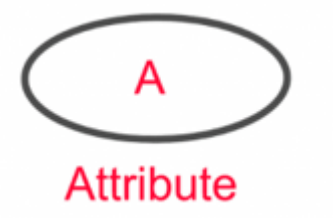
An **attribute** is a property or characteristic that describes an entity or relationship.

Example

For STUDENT entity:

- USN
- Name
- Branch

ER Notation



(Ellipse represents an Attribute)

3. Composite Attribute

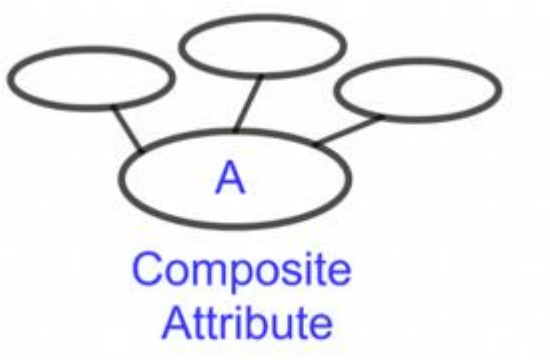
Definition

A **composite attribute** is an attribute that can be divided into smaller subparts, each having independent meaning.

Example

Address can be divided into:

- House_No
- Street
- City
- State



4. Multivalued Attribute

A **multivalued attribute** can have more than one value for a single entity occurrence.

Example

A student may have multiple phone numbers.

Attribute:

Phone_No

(Double ellipse represents a multivalued attribute)

5. Participation Role (Role Name)

A **role** specifies the function or participation of an entity in a relationship. It is particularly useful when the same entity type participates more than once in a relationship.

Example

In the relationship **SUPERVISES**, one EMPLOYEE acts as a **Supervisor** and another EMPLOYEE acts as a **Subordinate**.

ER Notation

```
Supervisor
    |
EMPLOYEE
```

```
      |
 SUPERVISES
      |
EMPLOYEE
      |
 Subordinate
```

Here, **Supervisor** and **Subordinate** are participation roles played by the EMPLOYEE entity.

Q.3	a.	Discuss the update operations and dealing with constraint violations with suitable examples.	08	L2	CO2
	b.	Illustrate the relational algebra operators with examples for select and project operation.	06	L2	CO2
	c.	Discuss the characteristics of relations that make them different from ordinary table and files.	06	L2	CO2

Q. 3. a. Discuss the update operations and dealing with constraint violations with suitable examples (8M)

The **update operations** in a relational database are **INSERT, DELETE, and UPDATE (MODIFY)**. These operations change the state of a database by adding, removing, or modifying tuples. While performing these operations, integrity constraints must be maintained to ensure database consistency.

1. INSERT Operation

The **INSERT** operation is used to add new tuples into a relation.

Example

STUDENT(USN, Name, Branch)

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

Constraint Violations

- **Domain Constraint:** Invalid attribute value.
- **Key Constraint:** Duplicate primary key.
- **Entity Integrity:** NULL primary key.
- **Referential Integrity:** Foreign key value not present in referenced table.

Example: Inserting an employee with a non-existing Dept_ID violates referential integrity.

2. DELETE Operation

The **DELETE** operation removes tuples from a relation.

Example

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

Constraint Violations

Deleting a tuple that is referenced by another relation may violate **referential integrity**.

Example

DEPARTMENT(Dept_ID)

EMPLOYEE(Emp_ID, Dept_ID)

Deleting a department that is referenced by employees creates dangling references.

Actions Taken

- Reject deletion
- Cascade deletion
- Set NULL
- Set Default value

3. UPDATE (MODIFY) Operation

The **UPDATE** operation changes existing attribute values in a relation.

Example

```
UPDATE STUDENT
SET Branch='AIML'
WHERE USN=101;
```

Constraint Violations

- Updating a primary key to a duplicate value violates key constraints.
- Updating a foreign key to a non-existing value violates referential integrity.
- Updating an attribute with an invalid domain value violates domain constraints.

Actions Taken

- Reject update
- Cascade update
- Set NULL
- Set Default value

Q.3.b. Illustrate the relational algebra operations with examples for select and project operations (6M)

Relational Algebra is a procedural query language used to perform operations on relations (tables). Two fundamental operations are **Select (σ)** and **Project (π)**.

1. Select Operation (σ)

The **Select** operation retrieves tuples (rows) from a relation that satisfy a specified condition. It is also called the **horizontal subset operation** because it selects certain rows.

Notation

$$\sigma_{condition}(Relation)$$

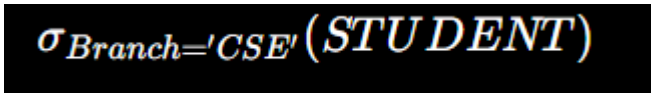
Example

Consider the relation:

STUDENT

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

To select students belonging to the CSE branch:



$$\sigma_{Branch='CSE'}(STUDENT)$$

Result

USN	Name	Branch
101	Ravi	CSE
103	Arun	CSE

Characteristics

- Selects rows based on a condition.
- Number of attributes remains the same.
- Number of tuples may decrease.

2. Project Operation (π)

The **Project** operation retrieves specified attributes (columns) from a relation. It is also called the **vertical subset operation** because it selects certain columns.

Notation

$$\pi_{attribute-list}(Relation)$$

Example

Using the STUDENT relation:

To display only Name and Branch:

$$\pi_{Name,Branch}(STUDENT)$$

Result

Name	Branch
Ravi	CSE
Priya	ISE
Arun	CSE

Characteristics

- Selects specific columns.
- Removes duplicate tuples automatically.
- Number of attributes decreases.

Q. 3. c. Discuss the characteristics of relations that make them different from ordinary table and files (6M)

In the relational model, data is stored in the form of **relations (tables)**. A relation possesses certain characteristics that distinguish it from ordinary tables and conventional file systems. These characteristics ensure data consistency, integrity, and efficient processing.

Characteristics of Relations

1. Relation Has a Unique Name

Each relation in a database must have a unique name to distinguish it from other relations.

Example:

- STUDENT
- EMPLOYEE
- DEPARTMENT

2. Attribute Names Are Unique

Each attribute (column) within a relation must have a distinct name.

Example:

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

Here, USN, Name, and Branch are unique attribute names.

3. Order of Tuples Is Not Important

The order in which rows appear in a relation has no significance. Tuples can be stored in any order.

Example:

```
(101, Ravi, CSE)
(102, Priya, ISE)
```

or

```
(102, Priya, ISE)
(101, Ravi, CSE)
```

Both represent the same relation.

4. Order of Attributes Is Not Important

The sequence of columns in a relation does not affect its meaning.

Example:

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

and

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

represent the same information.

5. No Duplicate Tuples

A relation cannot contain two identical tuples. Each row must be unique.

Example:

```
(101, Ravi, CSE)
(101, Ravi, CSE)
```

is not allowed.

6. Atomic Attribute Values

Each attribute contains only a single value and cannot be further divided.

Example:

Correct:

```
Phone = 9876543210
```

Incorrect:

```
Phone = 9876543210, 8765432109
```

7. Values in a Column Come from the Same Domain

All values of an attribute must belong to the same domain (data type).

Example:

Age column should contain only numeric values.

```
Age = 20, 21, 22
```

not

```
Age = 20, "ABC"
```

Q.4	a.	Perform (i) Student ∪ instructor (ii) Student ∩ Instructor (iii) Student – Instructor (iv) Instructor – Student on the following tables:	04	L3	CO2

Student

Fname	Lname
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

Instructor

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Mao
Francis	Johnson
Ramesh	Shah

(i) Student ∪ Instructor (Union)

Union contains all distinct tuples from both relations.

Fname	Lname
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang

Ernest	Gilbert
John	Smith
Ricardo	Browne
Susan	Mao
Francis	Johnson

Result = 11 tuples (Ramesh Shah appears only once).

(ii) Student ∩ Instructor (Intersection)

Intersection contains tuples present in both relations.

Fname	Lname
Ramesh	Shah

(iii) Student − Instructor (Difference)

Contains tuples present in Student but not in Instructor.

Fname	Lname
Susan	Yao
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(iv) Instructor − Student (Difference)

Contains tuples present in Instructor but not in Student.

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Mao
Francis	Johnson

	b.	Consider the following relational database schema and write the queries in relational algebra expressions: EMP(Eno, Ename, Salary, Address, Phone, DNo) DEPT(DNo, Dname, DLoc, MgrEno) DEPENDENT(Eno, Dep_Name, Drelation, Dage) (i) List all the employees who reside in 'Belagavi'. (ii) List all the employees who earn salary between 30000 and 40000 (iii) List all the employees who work for the 'Sales' department (iv) List all the employees who have at least one daughter (v) List the department names along with the names of the managers	10	L3	CO2

1) σAddress='Belagavi'(EMP)

2) σ30000≤Salary≤40000(EMP)

3) σDname='Sales'(EMP⋈EMP.DNo=DEPT.DNoDEPT)

4) $\sigma_{Drelation='Daughter'}(EMP \bowtie_{EMP.Eno=DEPENDENT.Eno} DEPENDENT)$

5) $\pi_{Dname,\ Ename}(DEPT \bowtie_{DEPT.MgrEno=EMP.Eno} EMP)$

	c.	Consider the two tables T_1 and T_2 shown below:	06	L3	CO2

T_1

P	Q	R
10	a	5
15	b	8
25	a	6

T_2

A	B	C
10	b	6
25	c	3
10	b	5

Show the results of the following operations:

(i) $T_1 \bowtie_{T_1.P=T_2.A} T_2$

(ii) $T_1 \bowtie_{T_1.Q=T_2.B} T_2$

(iii) $T_1 \bowtie_{(T_1.P=T_2.A \text{ AND } T_1.R=T_2.C)} T_2$

(i) $T_1 \bowtie_{T_1.P=T_2.A} T_2$

P	Q	R	A	B	C
10	a	5	10	b	6
10	a	5	10	b	5
25	a	6	25	c	3

(ii) $T_1 \bowtie_{T_1.Q=T_2.B} T_2$

P	Q	R	A	B	C
15	b	8	10	b	6
15	b	8	10	b	5

(iii) $T_1 \bowtie_{(T_1.P=T_2.A \text{ AND } T_1.R=T_2.C)} T_2$

P	Q	R	A	B	C
10	a	5	10	b	5

Q.5	a.	Discuss the informal design guidelines for relation schema design.	08	L2	CO4
	b.	Define 1NF, 2NF, and 3NF with examples.	06	L2	CO4
	c.	Write the syntax for INSERT, UPDATE and DELETE statements in SQL and explain with suitable examples.	06	L2	CO3

Q. 5. a. Discuss the informal design guidelines for relation schema design (8M)

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.

Q. 5. b. Define 1NF,2NF, and 3NF with examples (6M)

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. 5. c.Write the syntax for INSERT, UPDATE and DELETE statements in SQL and explain with suitable examples (6M)

INSERT, UPDATE, and DELETE are Data Manipulation Language (**DML**) commands in SQL. They are used to add, modify, and remove data from database tables.

1. INSERT Statement

The **INSERT** statement is used to add new records (tuples) into a table.

Syntax

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

Example

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

Result

USN	Name	Branch
101	Ravi	CSE

2. UPDATE Statement

The **UPDATE** statement is used to modify existing records in a table.

Syntax

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

Example

```
UPDATE STUDENT
SET Branch = 'ISE'
WHERE USN = 101;
```

Result

USN	Name	Branch
101	Ravi	ISE

3. DELETE Statement

The **DELETE** statement is used to remove records from a table.

Syntax

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

Example

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

Result

The record of student 101 is removed from the table.

Q.6	a.	Discuss insertion, deletion and modification anomalies. Why are they considered bad? Illustrate with examples.	10	L2	CO3
	b.	Illustrate the following with suitable examples: (i) Datatypes in SQL (ii) Substring Pattern Matching in SQL.	10	L2	CO3

Q. 6. a. Discuss insertion, deletion, and modification anomalies. Why are they considered bad? Illustrate with examples (10M)

Database design should minimize **data redundancy** and maintain **data consistency**. When the same information is stored repeatedly in a relation, various problems called **update anomalies** may occur. These anomalies make database maintenance difficult and may lead to inconsistent or lost information.

The three major update anomalies are:

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

Example Relation

Consider the relation:

EMPLOYEE_DEPARTMENT(Emp_ID, Emp_Name, Dept_ID, Dept_Name)

Emp_ID	Emp_Name	Dept_ID	Dept_Name
E1	Ravi	D1	CSE
E2	Priya	D1	CSE
E3	Arun	D2	ISE

Notice that department information is repeated for every employee belonging to the same department.

1. Insertion Anomaly

An **insertion anomaly** occurs when certain information cannot be inserted into the database unless some other information is also inserted.

It arises because unrelated facts are stored together in the same relation.

Example

Suppose a new department:

```
D3 - AIML
```

is created, but no employee has yet been assigned to it.

Since the table requires employee information, the department details cannot be inserted independently.

Problems

- New information cannot be stored immediately.
- Requires insertion of unnecessary or NULL values.
- Causes difficulty in maintaining accurate records.
- Leads to incomplete representation of data.

Why It Is Considered Bad

A database should allow independent storage of different types of information. Insertion anomalies prevent this and reduce flexibility.

2. Deletion Anomaly

A **deletion anomaly** occurs when deleting a record unintentionally removes additional valuable information from the database.

Example

Suppose employee E3 leaves the company and the following tuple is deleted:

```
(E3, Arun, D2, ISE)
```

Since this is the only tuple containing department D2, information about the ISE department is also lost.

Problems

- Causes loss of important data.
- Information unrelated to the deletion disappears.
- Recovery of lost information may be difficult.

Why It Is Considered Bad

Deleting one fact should not remove another independent fact. Deletion anomalies reduce data reliability and integrity.

3. Modification (Update) Anomaly

A **modification anomaly** occurs when the same information is stored in multiple rows and must be updated repeatedly.

Example

Suppose the department name:

```
CSE
```

is changed to

```
Computer Science Engineering
```

The change must be made in every tuple containing Dept_ID D1.

Emp_ID	Emp_Name	Dept_ID	Dept_Name
E1	Ravi	D1	Computer Science Engineering
E2	Priya	D1	Computer Science Engineering

If one row is updated and another is not, inconsistent data results.

Problems

- Multiple updates are required.
- High maintenance cost.
- Increased possibility of errors.
- Causes inconsistent data.

Why It Is Considered Bad

The database may contain different values for the same fact, leading to confusion and incorrect query results.

Why Are Update Anomalies Considered Bad?

Update anomalies are undesirable because they:

- Increase data redundancy.
- Waste storage space.

- Cause data inconsistency.
- Lead to accidental loss of information.
- Increase maintenance effort.
- Make database updates more complex.
- Reduce data integrity and reliability.

A well-designed database should avoid these problems through proper normalization.

Q. 6. b. Illustrate the following with suitable examples:

(i) Datatypes in SQL

(ii) Substring Pattern Matching in SQL (10M)

(i) Data Types in SQL

A **data type** specifies the type of values that can be stored in a table column. SQL supports various data types for storing numbers, characters, dates, and other kinds of data.

Common SQL Data Types

1. Numeric Data Types

Used to store numbers.

- **INT / INTEGER** – Stores whole numbers.
- **FLOAT** – Stores real numbers with decimal points.
- **DECIMAL(p,d)** – Stores fixed-point numbers.

Example:

```
Salary DECIMAL(8,2)
Age INT
```

2. Character String Data Types

Used to store text values.

- **CHAR(n)** – Fixed-length string.
- **VARCHAR(n)** – Variable-length string.

Example:

```
Name VARCHAR(30)
Gender CHAR(1)
```

3. Date and Time Data Types

Used to store dates and times.

- **DATE**
- **TIME**
- **TIMESTAMP**

Example:

```
DOB DATE
Joining_Date TIMESTAMP
```

Example Table

```
CREATE TABLE STUDENT(
USN INT,
Name VARCHAR(30),
Age INT,
DOB DATE,
CGPA DECIMAL(3,2)
);
```

(ii) Substring Pattern Matching in SQL

Substring pattern matching is used to search for strings that match a specified pattern. SQL provides the **LIKE** operator along with wildcard characters.

Wildcards Used

Wildcard	Meaning
%	Represents zero or more characters
_	Represents exactly one character

Syntax

```
SELECT column_name
FROM table_name
WHERE attribute LIKE pattern;
```

Example 1: Using %

Consider STUDENT table:

USN	Name
Ravi	
Rajesh	
Ramesh	
Priya	

Find names starting with 'Ra':

```
SELECT Name
FROM STUDENT
WHERE Name LIKE 'Ra%';
```

Result

```
Ravi
Rajesh
Ramesh
```

Example 2: Using %

Find names ending with 'i':

```
SELECT Name
FROM STUDENT
WHERE Name LIKE '%i';
```

Result

```
Ravi
Priya
```

Example 3: Using _

Find names having exactly four letters and ending with 'i':

```
SELECT Name
FROM STUDENT
WHERE Name LIKE '___i';
```

Result

```
Ravi
```

Advantages of Pattern Matching

- Easy searching of text data.
- Useful for filtering records.
- Supports flexible string comparisons.
- Widely used in query processing.

Q.7	a.	Consider the following relations: Student(Snum, Sname, Branch, level, age) Class(Cname, meet_at, room, fid) Enrolled(Snum, Cname) Faculty(fid, fname, deptid) Write the following queries in SQL. No duplicates should be printed in any of the answers. (i) Find the names of all Juniors (level = JR) who are enrolled in a class taught by I. Teach. (ii) Find the names of all classes that either meet in room R128 or have five or more students enrolled. (iii) For all levels except JR, print the level and rthe average age of students for that level. (iv) For each faculty member that has taught classes only in room R128, print the faculty member's name and the total number of classes she or he has taught. (v) Find the names of students not enrolled in any class.	10	L3	CO3
	b.	What do understand by correlated Nested Queries in SQL? Explain with suitable example.	04	L2	CO3
	c.	Discuss the ACID properties of a database transaction.	06	L2	CO4

Q.7	a.	Consider the following relations: Student(Snum, Sname, Branch, level, age) Class(Cname, meet_at, room, fid) Enrolled(Snum, Cname) Faculty(fid, fname, deptid) Write the following queries in SQL. No duplicates should be printed in any of the answers. (i) Find the names of all Juniors (level = JR) who are enrolled in a class taught by I. Teach. (ii) Find the names of all classes that either meet in room R128 or have five or more students enrolled. (iii) For all levels except JR, print the level and rthe average age of students for that level. (iv) For each faculty member that has taught classes only in room R128, print the faculty member's name and the total number of classes she or he has taught. (v) Find the names of students not enrolled in any class.	10	L3	CO3

(i) Find the names of all Juniors (level = 'JR') who are enrolled in a class taught by "I. Teach".

```
SELECT DISTINCT S.Sname
FROM Student S, Enrolled E, Class C, Faculty F
WHERE S.Snum = E.Snum
  AND E.Cname = C.Cname
  AND C.fid = F.fid
  AND S.level = 'JR'
  AND F.fname = 'I. Teach';
```

(ii) Find the names of all classes that either meet in room R128 or have five or more students enrolled.

```
SELECT DISTINCT Cname
FROM Class
WHERE room = 'R128'
```

```
UNION

SELECT Cname
FROM Enrolled
GROUP BY Cname
HAVING COUNT(Snum) >= 5;
```

(iii) For all levels except JR, print the level and the average age of students for that level.

```
SELECT level, AVG(age) AS Avg_Age
FROM Student
WHERE level <> 'JR'
GROUP BY level;
```

(iv) For each faculty member that has taught classes only in room R128, print the faculty member's name and the total number of classes he or she has taught.

```
SELECT F.fname, COUNT(C.Cname) AS Total_Classes
FROM Faculty F, Class C
WHERE F.fid = C.fid
GROUP BY F.fid, F.fname
HAVING COUNT(DISTINCT room) = 1
   AND MAX(room) = 'R128';
```

(v) Find the names of students not enrolled in any class.

```
SELECT Sname
FROM Student
WHERE Snum NOT IN
      (SELECT Snum
       FROM Enrolled);
```

Q. 7. b. What do understand by correlated Nested Queries in SQL? Explain with suitable example (4M)

A **Correlated Nested Query** (or Correlated Subquery) is a subquery whose execution depends on the values obtained from the outer query. Unlike ordinary nested queries, a correlated subquery is evaluated **once for each row** processed by the outer query.

The inner query references one or more attributes from the outer query.

Syntax

```
SELECT column_name
FROM table1 T1
WHERE condition
(
   SELECT ...
   FROM table2 T2
   WHERE T2.attribute = T1.attribute
);
```

Example

Consider the relation:

EMPLOYEE(Emp_ID, Name, Salary, Dept_ID)

Find employees whose salary is greater than the average salary of their department.

```
SELECT E1.Name
FROM EMPLOYEE E1
WHERE Salary >
      (SELECT AVG(E2.Salary)
       FROM EMPLOYEE E2
       WHERE E2.Dept_ID = E1.Dept_ID);
```

- The outer query processes each employee.
- The inner query calculates the average salary of the department to which that employee belongs.
- Since the inner query uses **E1.Dept_ID** from the outer query, it is a **correlated subquery**.
- The subquery is executed for every row of the outer query.

Characteristics

- Inner query depends on the outer query.
- Executed repeatedly for each tuple of the outer query.
- Cannot be executed independently.
- Useful for complex comparisons and existence checks.

Q. 7. c. Discuss the ACID properties of a database transaction (6M)

A **transaction** is a sequence of one or more database operations that performs a single logical unit of work. A transaction may consist of operations such as insertion, deletion, updation, or retrieval of data. A transaction is considered complete only when all its operations are executed successfully. If any operation fails, the entire transaction is rolled back to preserve the consistency of the database.

Thus, a transaction transforms the database from one consistent state to another consistent state.

A transaction generally consists of the following operations:

- **BEGIN TRANSACTION** – Marks the beginning of a transaction.
- **READ(X)** – Reads the value of data item X.
- **WRITE(X)** – Updates the value of data item X.
- **COMMIT** – Permanently saves the changes made by the transaction.
- **ROLLBACK (ABORT)** – Cancels all changes made by the transaction and restores the database to its previous state.

Example

Consider a bank fund transfer of ₹5000 from Account A to Account B.

Read(A)

A = A – 5000

Write(A)

Read(B)

B = B + 5000

Write(B)

Commit

Both debit and credit operations together constitute a single transaction. If the system fails after deducting money from Account A but before adding it to Account B, the transaction must be rolled back to maintain consistency.

ACID Properties

ACID properties are a set of four properties that guarantee the correctness, reliability, and consistency of database transactions. ACID stands for **Atomicity, Consistency, Isolation, and Durability**.

1. Atomicity

Atomicity states that a transaction is treated as an indivisible unit of work. Either all the operations of a transaction are executed successfully, or none of them are executed. There is no concept of partial execution.

This property ensures that if any operation within the transaction fails, the entire transaction is aborted and the database is restored to its previous state.

Example

Consider transferring ₹5000 from Account A to Account B.

A = A – 5000

B = B + 5000

If the system crashes after deducting money from Account A but before crediting Account B, both operations are rolled back. Hence, either both operations occur or neither occurs.

Importance

- Prevents partial updates.
- Maintains data reliability.
- Ensures complete execution of transactions.

2. Consistency

Consistency ensures that every transaction preserves the integrity constraints of the database and transforms the database from one valid state to another valid state.

A transaction should not violate constraints such as primary key, foreign key, or domain constraints.

Example

Suppose Account A contains ₹20,000 and Account B contains ₹10,000.

Before transfer:

Total Balance = ₹30,000

After transferring ₹5000 from A to B:

A = ₹15,000

B = ₹15,000

Total Balance remains:

₹30,000

Thus, the consistency of the database is maintained.

Importance

- Preserves database integrity.
- Ensures valid data at all times.
- Prevents constraint violations.

3. Isolation

Definition

Isolation ensures that multiple transactions executing concurrently do not interfere with each other. The intermediate results of one transaction are hidden from other transactions until the transaction is completed.

From the user's perspective, concurrent transactions appear as though they are executed serially.

Example

Suppose two users simultaneously withdraw money from the same account.

Without isolation, both transactions may access the same balance and produce incorrect results. With isolation, one transaction completes before the other accesses the updated data, thereby preventing inconsistencies.

Importance

- Prevents interference among concurrent transactions.
- Avoids problems such as dirty reads and lost updates.
- Ensures correctness in multi-user environments.

4. Durability

Durability guarantees that once a transaction is committed, its effects are permanently stored in the database. Even if a system failure or power outage occurs after the commit operation, the committed changes are not lost.

The recovery mechanism ensures that committed transactions survive failures.

Example

Suppose a customer deposits ₹10,000 into an account and the transaction is committed.

Even if the system crashes immediately afterward, the deposited amount remains stored in

the database and can be recovered.

Importance

- Guarantees permanent storage of committed data.
- Protects against hardware and software failures.
- Provides reliability and fault tolerance.

Q.8	a.	What are the views in SQL? Explain with examples.	04	L3	CO5
	b.	In SQL, write the usage of GROUP BY and HAVING clauses with suitable examples.	06	L2	CO3
	c.	Discuss the types of problems that may encounter with transactions that run concurrently.	10	L2	CO5

Q. 8. a. What are the views in SQL? Explain with examples (4M)

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. 8. b. In SQL, write the usage of GROUP BY and HAVING clauses with suitable examples (6M)

In SQL, **GROUP BY** and **HAVING** clauses are used with aggregate functions such as **COUNT(), SUM(), AVG(), MAX(), and MIN()** to perform calculations on groups of rows. GROUP BY forms groups, while HAVING filters those groups based on a condition.

1. GROUP BY Clause

The **GROUP BY** clause groups rows having the same values in specified columns and applies aggregate functions to each group.

Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
GROUP BY column_name;
```

Example

Consider the EMPLOYEE table:

Emp_ID	Name	Dept_No	Salary
E1	Ravi	10	50000
E2	Priya	10	60000
E3	Arun	20	55000

Find the average salary of employees in each department:

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

Result

Dept_No	AVG(Salary)
10	55000
20	55000

2. HAVING Clause

The **HAVING** clause is used to specify conditions on groups created by the GROUP BY clause. It filters groups after aggregation.

Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
GROUP BY column_name
HAVING condition;
```

Example

Find departments having more than one employee:

```
SELECT Dept_No, COUNT(*)
FROM EMPLOYEE
GROUP BY Dept_No
HAVING COUNT(*) > 1;
```

Result

Dept_No	COUNT(*)
10	2

Q. 8. c. Discuss the types of problems that may encounter with transactions that run concurrently (10M)

In a multi-user database system, several transactions may execute simultaneously. This is known as **concurrent execution of transactions**. Although concurrency improves system performance and resource utilization, it may lead to inconsistencies if transactions access the same data items at the same time.

Such undesirable situations are called **concurrency problems** or **concurrency anomalies**.

1. Lost Update Problem

A **lost update** occurs when two transactions update the same data item concurrently and one update overwrites the other.

Example

Assume Account Balance = ₹10,000.

Transaction T1

```
Read(Balance)
Balance = Balance + 1000
Write(Balance)
```

Transaction T2

```
Read(Balance)
Balance = Balance - 500
Write(Balance)
```

If both transactions read ₹10,000 simultaneously:

- T1 writes ₹11,000
- T2 writes ₹9,500

The update made by T1 is lost.

Result

Final balance becomes ₹9,500 instead of ₹10,500.

2. Dirty Read (Temporary Update Problem)

A **dirty read** occurs when one transaction reads data written by another transaction that has not yet committed.

Example

Transaction T1 updates a salary from ₹50,000 to ₹60,000 but has not committed.

Transaction T2 reads ₹60,000.

Later T1 aborts, restoring salary to ₹50,000.

Result

T2 used incorrect data that never became permanent.

Problem

Leads to inconsistent and unreliable results.

3. Unrepeatable Read Problem

An **unrepeatable read** occurs when a transaction reads the same data item twice and obtains different values because another transaction modified the data in between.

Example

T1 reads salary = ₹50,000.

Before T1 reads again:

T2 updates salary to ₹60,000 and commits.

T1 reads salary again and gets ₹60,000.

Result

The same query produces different results within the same transaction.

4. Incorrect Summary Problem

An **incorrect summary** occurs when one transaction calculates an aggregate value while another transaction updates some of the involved data items.

Example

T1 calculates total balance of all accounts.

At the same time, T2 transfers money from Account A to Account B.

While T1 is computing the total, it may read:

- Updated value of A
- Old value of B

Result

Incorrect total balance is obtained.

5. Phantom Read Problem

A **phantom read** occurs when a transaction re-executes a query and finds additional or missing rows because another transaction inserted or deleted records.

Example

T1 executes:

```
SELECT * FROM EMPLOYEE
WHERE Dept_No = 10;
```

Before T1 repeats the query:

T2 inserts a new employee into Department 10 and commits.

When T1 executes the same query again, an extra row appears.

Result

The new row is called a **phantom tuple**.

		Module – 5			
Q.9	a.	What is the two phase locking protocol? How does it Guarantee serializability.	06	L2	CO5
	b.	Describe the wait-die and wound-wait protocols for deadlock prevention.	08	L2	CO5
	c.	List and explain the four major categories of NOSQL system.	06	L2	CO3

Q. 9. a. What is the two-phase locking protocol? How does it Guarantee serializability (6M)

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

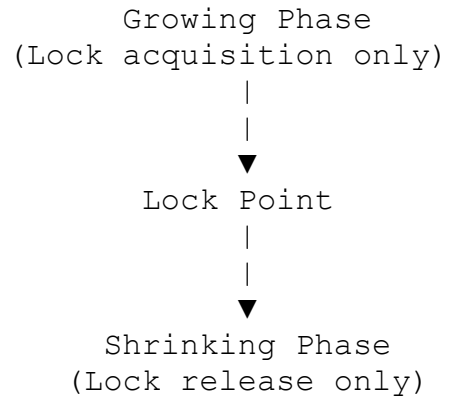
```
Unlock(A)
Unlock(B)
```

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.

How 2PL Guarantees Serializability

A schedule is **serializable** if its result is equivalent to some serial execution of transactions.

Under 2PL:

- Transactions acquire all required locks before releasing any lock.
- Conflicting operations are properly ordered.
- The order of lock points determines the equivalent serial order.

Example

If

```
Lock Point(T1) occurs before Lock Point(T2)
```

then the schedule is equivalent to:

```
T1 → T2
```

Thus, all schedules generated by 2PL are **conflict serializable**.

Q. 9. b. Describe the wait-die and wound-wait protocols for deadlock prevention (8M)

In a DBMS, a **deadlock** occurs when two or more transactions wait indefinitely for resources locked by one another. To prevent deadlocks, timestamp-based deadlock prevention protocols such as **Wait-Die** and **Wound-Wait** are used.

In these protocols, each transaction is assigned a unique **timestamp** when it starts. The transaction with the smaller timestamp is considered **older**, and the one with the larger timestamp is considered **younger**.

1. Wait-Die Protocol

The **Wait-Die** protocol is a **non-preemptive** deadlock prevention technique.

When transaction **Ti** requests a lock held by transaction **Tj**:

- If **Ti is older** than Tj, Ti is allowed to wait.
- If **Ti is younger** than Tj, Ti is aborted (dies) and restarted later with the same timestamp.

Rule

```
Older transaction  → Wait
Younger transaction → Die (Rollback)
```

Example

Assume:

```
T1 = Timestamp 10 (Older)
T2 = Timestamp 20 (Younger)
```

If T1 requests a resource held by T2:

```
T1 waits
```

If T2 requests a resource held by T1:

```
T2 is rolled back
```

Advantages

- Prevents circular waiting.
- Simple to implement.

Disadvantage

- Younger transactions may be rolled back repeatedly, causing starvation.

2. Wound-Wait Protocol

The **Wound-Wait** protocol is a **preemptive** deadlock prevention technique.

When transaction **Ti** requests a lock held by transaction **Tj**:

- If **Ti is older** than Tj, Ti wounds (forces rollback of) Tj.
- If **Ti is younger** than Tj, Ti waits.

Rule

```
Older transaction  → Wound (Abort younger)
Younger transaction → Wait
```

Example

Assume:

```
T1 = Timestamp 10 (Older)
T2 = Timestamp 20 (Younger)
```

If T1 requests a resource held by T2:

```
T2 is rolled back
T1 gets the lock
```

If T2 requests a resource held by T1:

```
T2 waits
```

Advantages

- Prevents deadlocks.
- Fewer rollbacks compared to Wait-Die.
- Older transactions complete faster.

Disadvantage

- Requires transaction rollback and restart mechanisms.

Q. 9. c. List and explain the four major categories of NOSQL system (6M)

NoSQL (Not Only SQL) databases are non-relational database management systems designed to handle large volumes of structured, semi-structured, and unstructured data. Unlike traditional relational databases, NoSQL databases do not require a fixed schema and provide high scalability, availability, and performance.

They are widely used in applications such as social networking sites, cloud computing, e-commerce platforms, big data analytics, content management systems, and real-time web applications.

The four major categories of NoSQL systems are:

1. Key-Value Stores
2. Document Databases
3. Column-Family Databases

4. Graph Databases

1. Key-Value Stores

A **Key-Value Store** is the simplest form of NoSQL database. Data is stored as a collection of key-value pairs, where each key uniquely identifies a value.

Structure

```
Key → Value
```

Example

```
101 → Ravi
102 → Priya
```

Here, 101 and 102 are keys, and the corresponding student names are values.

Characteristics

- Simple data model.
- Fast insertion and retrieval.
- Highly scalable.
- Suitable for distributed systems.
- Efficient for caching and session management.

Applications

- User session storage
- Shopping carts
- Caching systems

Examples

- Redis
- Amazon DynamoDB
- Riak

2. Document Databases

A **Document Database** stores data in the form of documents rather than rows and columns. Documents are usually represented using JSON, BSON, or XML formats.

Each document can have a different structure, providing schema flexibility.

Example

```
{
  "USN": 101,
  "Name": "Ravi",
```

```
  "Branch": "CSE"
}
```

Characteristics

- Schema-less design.
- Supports nested structures.
- Easy storage of semi-structured data.
- Flexible and scalable.
- Documents are self-describing.

Applications

- Content management systems
- E-commerce applications
- Mobile applications
- Web applications

Examples

- MongoDB
- CouchDB

3. Column-Family Databases

A **Column-Family Database** stores data in columns rather than rows. Related columns are grouped together into column families.

This model is inspired by Google's BigTable.

Example

```
Student
------------------------------
USN   Name     Branch
101   Ravi     CSE
102   Priya    ISE
```

Instead of storing complete rows together, data is organized by columns.

Characteristics

- High scalability.
- Efficient storage of large datasets.
- Fast read and write operations.
- Suitable for distributed environments.
- Handles huge amounts of data across multiple servers.

Applications

- Data warehousing
- Big data analytics
- Real-time data processing

Examples

- Apache Cassandra
- HBase
- Google BigTable

4. Graph Databases

A **Graph Database** stores data using **nodes**, **edges (relationships)**, and **properties**. It is specifically designed for applications where relationships among data are important.

Example

```
(Student) ---- ENROLLED_IN ----> (Course)
```

Here:

- Student and Course are nodes.
- ENROLLED_IN is the relationship.

Characteristics

- Efficiently represents interconnected data.
- Fast traversal of relationships.
- Flexible schema.
- Suitable for complex relationship queries.

Applications

- Social networking sites
- Recommendation systems
- Fraud detection
- Network analysis

Examples

- Neo4j
- OrientDB

Q.10	**a.**	What is Multiple Granularity locking? How is it implemented using intension locks? Explain.	**10**	**L2**	**CO5**
	b.	Discuss the following MongoDB CRUD operations with their formats: (i) Insert (ii) Delete (iii) Read	**06**	**L2**	**CO4**
	c.	Briefly discuss about Neo4j data model.	**04**	**L2**	**CO4**

Q. 10. a. What is Multiple Granularity locking? How is it implemented using intension locks? Explain (6M)

In a database system, data can be locked at different levels such as **database, table, page, or record level**. Locking every individual record may cause high overhead, whereas locking the entire database reduces concurrency. To overcome this problem, **Multiple Granularity Locking (MGL)** is used.

Multiple Granularity Locking allows transactions to lock data items at different levels of a hierarchy while maintaining consistency and improving concurrency.

Multiple Granularity Locking

Multiple Granularity Locking (MGL) is a locking technique in which data items are organized in a hierarchy, and locks can be applied at various levels of granularity.

Hierarchy Example

```
Database
   |
 Tables
   |
 Pages
   |
 Records
```

A transaction may lock:

- Entire database
- A table
- A page
- An individual record

depending on its requirements.

Advantages

- Improves concurrency.
- Reduces locking overhead.
- Provides flexibility in lock management.
- Suitable for large databases.

Intention Locks

To support multiple granularity locking, the DBMS uses **Intention Locks**. These locks indicate that a transaction intends to acquire locks at lower levels of the hierarchy.

The three types of intention locks are:

1. Intention Shared (IS)

Indicates that the transaction intends to acquire **Shared (S) locks** on lower-level nodes.

Example

A transaction wants to read certain records in a table.

```
Database (IS)
   |
Table (IS)
   |
Record (S)
```

2. Intention Exclusive (IX)

Indicates that the transaction intends to acquire **Exclusive (X) locks** on lower-level nodes.

Example

```
Database (IX)
   |
Table (IX)
   |
Record (X)
```

The transaction intends to update a record.

3. Shared and Intention Exclusive (SIX)

A transaction holds a **Shared lock** on a node and intends to acquire **Exclusive locks** on some lower-level nodes.

Example

```
Table (SIX)
   |
Record (X)
```

The entire table is read, but some records are updated.

Rules for Implementation

1. Lock the root node first.
2. To acquire an S or IS lock on a node, hold an IS or IX lock on its parent.
3. To acquire an X, IX, or SIX lock on a node, hold an IX or SIX lock on its parent.
4. Locks are released from bottom to top.

Example

Suppose a transaction wants to update a record:

```
Database (IX)
     |
  Table (IX)
     |
 Record (X)
```

The transaction first acquires IX locks on the database and table, then obtains an X lock on the record to perform the update.

Q. 10. b. Discuss the following MongoDB CRUD operations with their formats (i) Insert (ii) Delete (iii) Read (6M)

CRUD stands for **Create, Read, Update, and Delete**. These are the fundamental operations used to manipulate documents in MongoDB. MongoDB is a **NoSQL document-oriented database** that stores data in the form of **BSON (Binary JSON) documents** inside collections. CRUD operations provide mechanisms to insert, retrieve, modify, and remove documents efficiently.

1. Create Operation

The **Create** operation is used to add new documents into a collection. If the collection does not exist, MongoDB automatically creates it when the first document is inserted.

Operations Used

- `insertOne()`
- `insertMany()`

Syntax

```
db.collection_name.insertOne(document)
```

Example

```
db.Student.insertOne(
{
sid:101,
name:"Ravi",
branch:"CSE",
age:21
}
)
```

Working

- A new document is inserted into the collection.
- MongoDB automatically generates a unique `_id` field.
- Multiple documents can be inserted simultaneously using `insertMany()`.

2. Read Operation

Definition

The **Read** operation is used to retrieve documents stored in a collection. MongoDB provides various methods to query data based on specified conditions.

Operations Used

- `find()`
- `findOne()`

Syntax

```
db.collection_name.find(condition)
```

Example

Display all students:

```
db.Student.find()
```

Display students belonging to CSE branch:

```
db.Student.find(
{
branch:"CSE"
}
)
```

Working

- Searches the collection for matching documents.
- Returns all documents satisfying the specified condition.
- If no condition is given, all documents are returned.

3. Update Operation

Definition

The **Update** operation modifies existing documents in a collection. It allows changing one or more fields of selected documents.

Operations Used

- `updateOne()`
- `updateMany()`

Syntax

```
db.collection_name.updateOne(
condition,
{$set:{field:value}}
)
```

Example

Update age of student 101:

```
db.Student.updateOne(
{
sid:101
},
{
$set:{age:22}
}
)
```

Working

- Finds the document matching the condition.
- Modifies only the specified fields.
- Other fields remain unchanged.

4. Delete Operation

Definition

The **Delete** operation removes documents from a collection. MongoDB provides methods to delete one or multiple documents.

Operations Used

- `deleteOne()`
- `deleteMany()`

Syntax

```
db.collection_name.deleteOne(condition)
```

Example

Delete student whose SID is 101:

```
db.Student.deleteOne(
{
sid:101
}
)
```

Working

- Searches for documents satisfying the condition.
- Removes the selected documents permanently.
- `deleteMany()` removes all matching documents.

Advantages of CRUD Operations in MongoDB

- Provide simple and efficient data manipulation.
- Support flexible schema design.
- Handle large volumes of data efficiently.
- Enable fast retrieval and modification of documents.
- Suitable for scalable and distributed applications.

Q. 10. c. Briefly discuss about Neo4j data model (4M)

Neo4j is a popular **NoSQL graph database** that stores data in the form of a graph. Unlike relational databases that use tables and rows, Neo4j represents data using **nodes, relationships, and properties**. It is designed to efficiently manage highly connected data and complex relationships.

Components of Neo4j Data Model

1. Nodes

Nodes represent entities or objects in the real world.

Examples:

- Student
- Employee
- Course

```
(Student)
```

Each node can store properties.

2. Relationships

Relationships connect nodes and represent how entities are associated.

Example:

```
(Student) ---- ENROLLED_IN ----> (Course)
```

Here, **ENROLLED_IN** is the relationship between Student and Course.

Relationships are directed and have meaning.

3. Properties

Properties are attributes stored in nodes and relationships as key-value pairs.

Example:

```
(Student)
Name = Ravi
USN = 101
```

Properties provide additional information about nodes and relationships.

Example of Neo4j Graph

```
(Ravi) ---- STUDIES ----> (DBMS)

Node          Relationship      Node
Student       STUDIES           Subject
```

Advantages of Neo4j

- Efficient storage of connected data.
- Fast relationship traversal.
- Flexible schema.
- Suitable for social networks, recommendation systems, and network analysis.