

Code: 23CS3402, 23IT3402, 23AM3402, 23DS3402

II B.Tech - II Semester – Regular Examinations - MAY 2025**DATABASE MANAGEMENT SYSTEMS****(Common for CSE, IT, AIML, DS)**

Duration: 3 hours

Max. Marks: 70

Note: 1. This question paper contains two Parts A and B.

2. Part-A contains 10 short answer questions. Each Question carries 2 Marks.

3. Part-B contains 5 essay questions with an internal choice from each unit. Each Question carries 10 marks.

4. All parts of Question paper must be answered in one place.

BL – Blooms Level

CO – Course Outcome

	b)	Write about loss-less join decomposition with an example.	L3	CO3	5 M
UNIT-V					
10		Explain in detail about transaction management with an example.	L2	CO1	10 M
OR					
11	a)	What is concurrency control? Explain two phase locking protocol with an example.	L2	CO1	5 M
	b)	Give a note on log based recovery.	L2	CO1	5 M

PART – A

		BL	CO
1.a)	List the role of database administrator.	L2	CO1
1.b)	What is a Data model?	L2	CO1
1.c)	List the design issues in ER diagrams.	L2	CO1
1.d)	Define composite attribute.	L2	CO1
1.e)	What is the purpose of relational algebra?	L2	CO1
1.f)	What is the difference between drop and delete in SQL?	L2	CO1
1.g)	Write about Attributes and its Types.	L2	CO1
1.h)	What are the different problems caused by redundancy?	L2	CO1
1.i)	List the types of serializability.	L2	CO1
1.j)	Give an example for concurrency control validation.	L2	CO1

PART – B

			BL	CO	Max. Marks
UNIT-I					
2	a)	Discuss the main characteristics of the database approach and how it differs from traditional file systems.	L2	CO1	5 M
	b)	What are the data models in database system and explain with examples?	L2	CO1	5 M
OR					
3	a)	Describe the three-schema architecture. Why do we need mappings between schema levels?	L2	CO1	5 M
	b)	Explain the difference between Two-tier and Three-tier architectures. Which is better suited for Web applications? why?	L2	CO1	5 M
UNIT-II					
4	a)	Explain the naming conventions and notations used in ER diagram.	L3	CO2	5 M
	b)	Explain Strong Entity Sets and Weak Entity Sets with examples.	L3	CO2	5 M
OR					
5		Draw an E-R diagram for a University Database, identify the key entities and the relationships between them.	L4	CO4	10 M

UNIT-III

6		Consider the following tables: Student_Details (<u>Reg_no</u> , Name, Address, Phone_No, Grade) Project_Details (<u>Project_Code</u> , Project_Name, Project_Cost) Student_Project (<u>Reg_No</u> , <u>Project_Code</u>) i. Write SQL statement to create SQL tables with suitable integrity constraints. ii. Write SQL statement to retrieve the names of the students who works on more than one project.	L4	CO4	10 M
OR					
7	a)	Enumerate various select and projection operations on relations with relational algebra.	L3	CO2	5 M
	b)	Discuss various set operations in relational algebra with examples.	L3	CO2	5 M
UNIT-IV					
8	a)	Explain the process of decomposition using multivalued dependencies with suitable example.	L3	CO3	5 M
	b)	Define normalization. Why do we need to normalize the database?	L3	CO3	5 M
OR					
9	a)	Describe the process of normalization in order to set the database to 3rd normal form.	L3	CO3	5 M

P.V.P.SIDDHARTHA INSTITUTE OF TECHNOLOGY
II B.Tech – II Semester – Regular Examinations – May, 2025
DATABASE MANAGEMENT SYSTEMS
Code: 23CS3402, 23AM3402, 23DS3402, 231T3402
(COMPUTER SCIENCE & ENGINEERING)
Scheme of Valuation

(Common for CSE, IT, AIML, DS)

PART A

Part-A contains 10 short answer questions. Each Question carries 2 Marks.

1. a) List the role of database administrator. L2, CO1
- b) What is a Data model? L2, CO1
- c) List the design issues in ER diagrams. L2, CO1
- d) Define composite attribute. L2, CO1
- e) What is the purpose of relational algebra? L2, CO1
- f) What is the difference between drop and delete in SQL? L2, CO1
- g) Write about Attributes and its Types. L2,CO1
- h) What are the different problems caused by redundancy? L2,CO1
- i) List the types of serializability. L2,CO1
- j) Give an example for concurrency control validation. L2,CO1

PART B

UNIT-I

2 a) Discuss the main characteristics of the database approach and how it differs from traditional file systems. L2 CO1 5 M

- i. Key Characteristics (3M)
- ii. Differentiate the database approach from traditional file processing systems. (2M)

2 b) What are the data models in database system and explain with examples? L2 CO1 5 M

Note: Describe any four Data Models

OR

3 a) Describe the three-schema architecture. Why do we need mappings between schema levels? L2 CO1 5 M

- i. The three-schema architecture (3M)
- ii. mappings between schema levels(2M)

3 b) Explain the difference between Two-tier and Three-tier architectures. Which is better suited for Web applications? why? L2 CO1 5 M

- i. Difference between Two-tier and Three-tier architectures (3M)
- ii. Importance of Web applications(2M)

UNIT-II

4 a) Explain the naming conventions and notations used in ER diagram. L3 CO2 5 M

- i. Naming conventions (2.5 M)
- ii. Notations(2.5 M)

4 b) Explain Strong Entity Sets and Weak Entity Sets with examples. L3 CO2 5 M

- i. Strong Entity Sets(2.5 M)
- ii. Weak Entity Sets (2.5 M)

OR

5. Draw an E-R diagram for a University Database, identify the key entities and the relationships between them L4 CO4 10 M

- i. An E-R diagram for a University Database(3M)
- ii. identify the key entities and the relationships(2M)

UNIT-III

6. Consider the following tables:

Student_Details (Reg_no, Name, Address, Phone_No, Grade)

Project_Details (Project Code, Project_Name, Project_Cost)

Student_Project (Reg_No, Project_Code)

i. Write SQL statement to create SQL tables with suitable integrity constraints.
ii. Write SQL statement to retrieve the names of the students who works on more than one project L4 CO4 10 M

- i. Write SQL statement to create SQL tables with suitable integrity constraints.(7M)
- ii. Write SQL statement to retrieve the names of the students who works on more than one project (3M)

OR

7 a) Enumerate various select and projection operations on relations with relational algebra. L3 CO2 5 M

i. Describe select and projection operations on relations with relational algebra.(3M)

ii. Examples (Provide one Example) (2M)

7.b) Discuss various set operations in relational algebra with examples.

L3 CO2 5 M

i. Describe set operations(3M)

ii. Followed Examples (2M)

UNIT-IV

8 a) Explain the process of decomposition using multivalued dependencies with suitable example. L3 CO3 5 M

i. The process of decomposition using multivalued dependencies(3M)

ii. Suitable Example (2M)

8.b) Define normalization. Why do we need to normalize the database?

L3 CO3 5 M

i. Define about Normalization (1M)

ii. Need to normalize the database(4M)

OR

9 a) Describe the process of normalization in order to set the database to 3rd normal form. L3 CO3 5 M

i. The process of normalization in order to set the database to 3rd normal form. (3M)

ii.Followed Example (2M)

9.b) Write about loss-less join decomposition with an example. L3 CO3 5 M

i. Description about loss-less join decomposition (3M)

ii. Suitable Example (2M)

UNIT-V

10. Explain in detail about transaction management with an example.

L2 CO1 10 M

i. Description about Transaction management with properties (8M)

ii. Example (2M)

OR

11 a) What is concurrency control? Explain two phase locking protocol with an example. L2 CO1 5 M

i. Definition about concurrency control (1M)

ii. Two phase locking protocol with an example.(4M)

11. b) Give a note on log based recovery. L2 CO1 5 M

i. Definition and Types of Operations in the Log.(3M)

ii. Recovery Process (2M)

P.V.P.SIDDHARTHA INSTITUTE OF TECHNOLOGY
II B.Tech – II Semester – Regular Examinations – May, 2025

DATABASE MANAGEMENT SYSTEMS

Code: 23CS3402, 23AM3402, 23DS3402, 231T3402

(COMPUTER SCIENCE & ENGINEERING)

Scheme of Valuation

(Common for CSE, IT, AIML, DS)

PART A

Part-A contains 10 short answer questions. Each Question carries **2 Marks**.

1. a) **List the role of database administrator.** 2M L2, CO1

Ans: Key roles and responsibilities of a DBA:

- i. Database Design
- ii. Installation and Configuration
- iii. Data Security and Access Control
- iv. Performance Monitoring and Tuning
- v. Backup and Recovery
- vi. Data Integrity and Consistency

Note: Mention any 4 roles

- b) **What is a Data model?** 2M L2, CO1

Ans: A data model is a conceptual framework used to organize and define how data is stored, connected, processed, and accessed within a database system.

- c) **List the design issues in ER diagrams.** 2M L2, CO1

Ans:

- i. Use of Entity Sets vs. Attributes
- ii. Use of Entity Sets vs. Relationship Sets
- iii. Use of Binary vs. n-ary Relationships
- iv. Placement of Relationship Attributes
- v. Identifying Keys
- vi. Participation Constraints

Note: Mention any 4 roles

- d) **Define composite attribute.** 2M L2, CO1

A composite attribute is an attribute that can be divided into smaller subparts, each representing more basic attributes with independent meanings.

Example:

Name can be a composite attribute:

Name → {First_Name, Middle_Name, Last_Name}

e) What is the purpose of relational algebra? 2M L2, CO1

Ans: Relational algebra is a formal query language used to manipulate and retrieve data stored in relational databases.

f) What is the difference between drop and delete in SQL? 2M L2, CO1

Ans:

Feature	DELETE	DROP
Purpose	Removes data (rows) from a table	Removes the entire object (table, view, etc.)
Effect on Table	Table structure remains	Table structure is completely removed
Can Use WHERE	<input checked="" type="checkbox"/> Yes (to delete specific rows)	<input checked="" type="checkbox"/> No (removes entire object)
Used On	Data (rows)	Schema objects (tables, views, indexes, etc.)
Example	DELETE FROM employees WHERE department = 'HR'; (Deletes only rows from employees table where department is 'HR')	DROP TABLE employees; (Completely deletes the employees table and all its data)

g) Write about Attributes and its Types. 2M L2,CO1

Ans: An attribute is a property or characteristic of an entity in a database. In simple terms, an attribute represents a column in a table and holds information about an entity.

Types of Attributes:

1. **Simple (Atomic) Attribute**
 - o Example: Age, Salary, Student_ID
2. **Composite Attribute**
 - o Example: Name → {First_Name, Last_Name}
3. **Derived Attribute**
 - o Example: Age can be derived from Date_of_Birth.
4. **Multi-valued Attribute**
 - o Example: Phone_Numbers, Email_Addresses
5. **Single-valued Attribute**
 - o Example: Roll_Number, Gender
6. **Key Attribute**
 - o Example: Employee_ID in an employee table.
7. **Stored Attribute**
 - o Example: Date_of_Birth is stored;

h) What are the different problems caused by redundancy? 2M L2,CO1

Problems Caused by Redundancy in Databases:

Data redundancy means the same piece of data is stored in **multiple places** unnecessarily. It leads to several serious problems, especially in large or unnormalized databases.

- i. Inconsistency
- ii. Update Anomalies
- iii. Insertion Anomalies
- iv. Deletion Anomalies

- i) List the types of serializability. 2M
L2,CO1

Ans:

- i. Conflict Serializability
- ii. View Serializability

- j) Give an example for concurrency control validation. 2M
L2,CO1

Ans:

Concurrency control validation is a technique used in database systems to ensure that transactions are executed in a way that maintains consistency, even when multiple transactions are running concurrently. One common method is **Optimistic Concurrency Control (OCC)**, which involves three phases: **Read, Validation, and Write**.

Example of Concurrency Control Validation (Optimistic Concurrency Control)

Two transactions, T1 and T2 are executing concurrently on a banking database.
T1 transfers \$100 from Account A to Account B.
T2 deposits \$50 into Account A.

Steps in Optimistic Concurrency Control:

1. Read Phase:

T1 reads Account A (balance = \$500) and Account B (balance = \$300).
T2 reads Account A (balance = \$500).

2. Validation Phase (Conflict Check):

Before committing, each transaction checks if any conflicts occurred.

Conflict Rules:

- If two transactions wrote to the same data item, they conflict.
- If one transaction read data that another transaction wrote, they conflict.
- Here:
 - T1 writes to A and B.
 - T2 writes to A.
 - Since both modify Account A, a conflict exists.

3. Write Phase (if Validation Passes):

- If no conflict, changes are applied:
 - T1 updates A to \$400 and B to \$400.
 - T2 (if retried) updates A to \$450.

Optimistic Concurrency Control assumes conflicts are rare and checks only at commit time.

Validation ensures serializability by aborting conflicting transactions.

PART B

UNIT-I

2 a) Discuss the main characteristics of the database approach and how it differs from traditional file systems **L2 CO1 5 M**

Ans: Characteristics of the Database Approach: -

The main characteristics of the database approach versus the file-processing approach are the following:

- Self-describing nature of a database system.
- Insulation between programs and data, and data abstraction.
- Support of multiple views of the data.
- Sharing of data and multiuser transaction processing

Feature	Traditional File System	Database Approach
Data Redundancy	High (multiple copies in different files)	Low (centralized control avoids duplication)
Data Integrity	Hard to enforce	Enforced using constraints
Data Independence	Low	High (schemas abstract physical storage)
Security	File-level only	Fine-grained access control
Concurrency Control	Manual and error-prone	Handled automatically by DBMS
Backup and Recovery	Manual	Built-in recovery mechanisms
Data Sharing	Difficult across programs	Easy due to centralized database
Scalability and Performance	Limited	Efficient for large, multi-user systems

The database approach offers a more structured, secure, and efficient way to manage data compared to traditional file systems. It enables better scalability, consistency, and ease of management, especially in multi-user environments.

2 .b) What are the data models in database system and explain with examples? **L2 CO1 5 M**

Ans: DataModel : A collection of concepts that can be used to describe the structure of a database— provides the necessary means to achieve this abstraction. By structure of a database that is data types, relationships, and constraints that apply to the data.

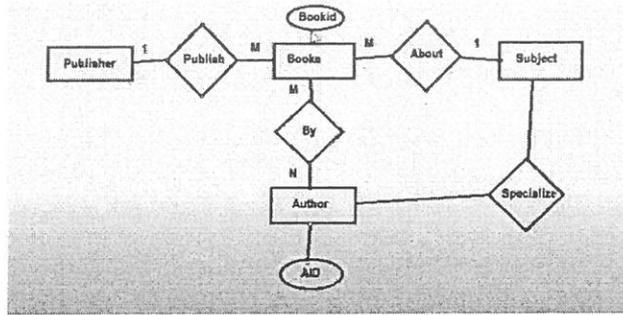
Examples:

Main categories of **Data Models**

i.. ER (Entity-Relationship) Model

The ER Model establishes the theoretical view of the database. It works around the real-time entities and the relationships among them. In View level, we consider ER models as the best option to design the databases.

ER Diagram Example:

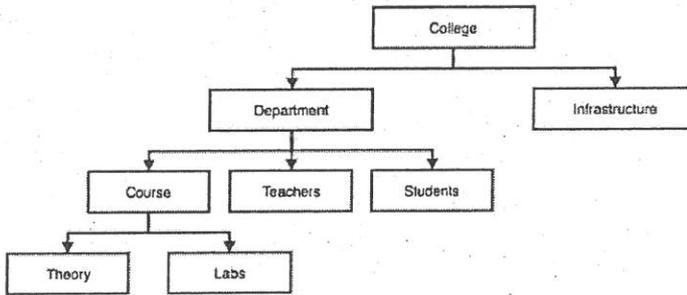


1.

2.

ii. Hierarchical Model

This data model arranges the data in the form of a tree with one root, to which other data is connected. The tree hierarchy begins with the "Root" data, and extends like a tree, by inserting the child nodes to the parent node.



iii. Relational Model

The relational model is the most common data model. It arranges the data into the tables, and tables are also known as relations. Tables will have columns and rows. Every column catalogs an attribute present in the entity like zip code, price, etc.

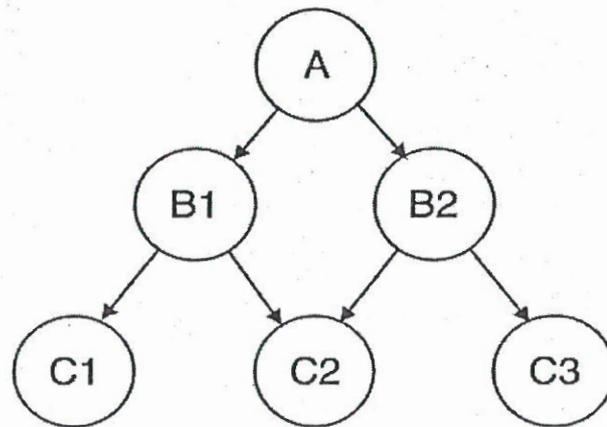
Student_Id	Student_Name	Student_Age
01	Vijay	20
02	Ramesh	22
03	Rakesh	21
04	Varun	25

iv. Network Model

In this database model data is more related as more relationships are established in this database model. Also, as the data is more related,

hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships.

This was the most widely used database model, before Relational Model was introduced.



OR

3 a) Describe the three-schema architecture. Why do we need mappings between schema levels? L2 CO1 5 M

Three-schema architecture was proposed to help achieve and visualize these characteristics

The goal of the three-schema architecture is to separate the user applications from the physical database. In this architecture, schemas can be defined at the following three levels:

⇒ The **internal level** has an internal schema, which describes the physical storage structure of the database.

The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database

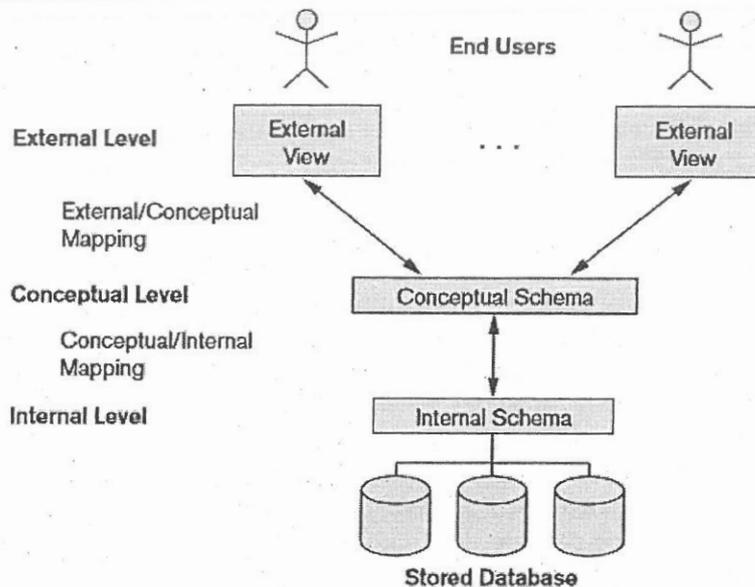
⇒ The **conceptual level** has a conceptual schema, which describes the structure of the whole database for a community of users.

The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints.

A representational data model is used to describe the conceptual schema when a database system is implemented.

⇒ The **external or view level** includes a number of external schemas or user views.

Each external schema 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. Each external schema is typically implemented using a representational data model.



Most DBMSs do not separate the three levels completely and explicitly, but support the three-schema architecture to some extent.

The three-level ANSI architecture has an important place in database technology development because it clearly separates the users' external level, the database's conceptual level, and the internal storage level for designing a database.

Example: Universal Data Base (UDB), a DBMS from IBM, which uses the relational model to describe the conceptual schema, but may use an object-oriented model to describe an external schema.

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 called mappings.*

Mappings connect and translate data between these three levels. They serve two major purposes:

i. Data Abstraction

- Users do not need to know **how data is stored** or **where it's stored**.
- Example: A user queries "SELECT Name FROM Student" without knowing file paths or indexes.

ii. Data Independence

- **Logical Data Independence:** Change in conceptual schema **does not affect** external views.
 - e.g., Adding a new attribute to the Student entity shouldn't affect applications using existing views.
- **Physical Data Independence:** Change in internal schema **does not affect** the conceptual schema.
 - e.g., Moving from heap file to B+ tree indexing shouldn't affect the logical structure.

3.b) Explain the difference between Two-tier and Three-tier architectures. Which is better suited for Web applications? why? L2 CO1 5 M

Two-Tier client/Server architecture for DBMS:

The architecture is called two-tier architectures because the software components are distributed over two systems: client and server.

The query and transaction functionality related to SQL processing remained on the server side. In such an architecture, the server is often called a **query server** or **transaction server** because it provides these two functionalities. The server is also often called an **SQL server**.

The user interface programs and application programs can run on the client side. When DBMS access is required, the program establishes a connection to the DBMS (which is on the server side); once the connection is created, the client program can communicate with the DBMS.

A standard called **Open Database Connectivity (ODBC)** provides an **application programming interface (API)**, which allows client-side programs to call the DBMS, as long as both client and server machines have the necessary software installed.

Most DBMS vendors provide ODBC drivers for their systems. A client program can actually connect to several RDBMSs and send query and transaction requests using the ODBC API, which are then processed at the server sites.

Any query results are sent back to the client program, which can process and display the results as needed. A related standard for the Java programming language, called **JDBC**, has also been defined.

This allows Java client programs to access one or more DBMSs through a standard interface.

In one variation of client/server architecture, the server can be a data server because it provides data in disk pages to the client. This data can then be structured into objects for the client programs by the client-side DBMS software.

The advantages of this architecture are its simplicity and seamless compatibility with existing systems.

Three-tier client/server architecture for DBMS:

The emergence of the Web changed the roles of clients and servers, leading to the three-tier architecture.

It adds an intermediate layer between the client and the database server.

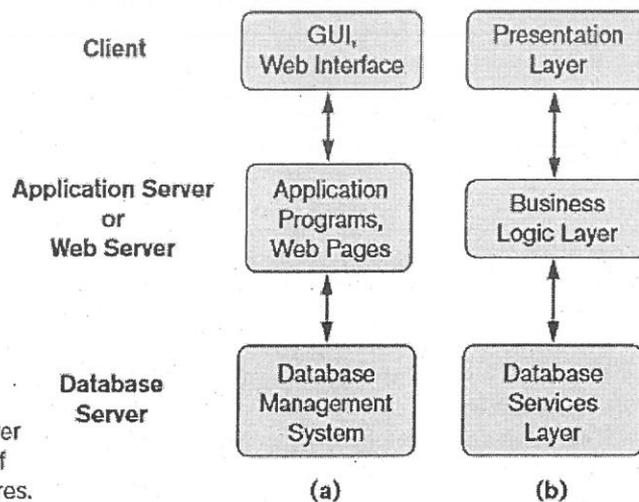


Figure 2.7
 Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.

This intermediate layer or middle tier is called the application server or the Web server. This server plays an intermediary role by running application programs and storing business rules (procedures or constraints) that are used to access data from the database server.

It can also improve database security by checking a client's credentials before forwarding a request to the database server.

Clients contain GUI interfaces and some additional application-specific business rules.

The intermediate server accepts requests from the client, processes the request and sends database queries and commands to the database server, and then acts as a conduit for passing (partially) processed data from the database server to the clients, where it may be processed further and filtered to be presented to users in GUI format.

Thus, the user interface, application rules, and data access act as the three tiers. In the above diagram part (b), the **presentation layer** displays information to the user and allows data entry.

The **business logic layer** handles intermediate rules and constraints before data is passed up to the user or down to the DBMS.

The **bottom layer** includes all data management services.

The middle layer can also act as a Web server, which retrieves query results from the database server and formats them into dynamic Web pages that are viewed by the Web browser at the client side. It is also possible to design n-tier architectures. Vendors of ERP (enterprise resource planning) and CRM (customer relationship management) packages often use a middleware layer, which accounts for the front-end modules (clients) communicating with a number of back-end databases (servers)

Better suited for web applications:

Three layers in the three tier architecture are as follows: 1) Client layer 2) Business layer 3) Data layer. **Three-tier architecture** is better suited for web applications. 3-tier application architecture provides a model by which developers can create flexible and reusable applications.

The best DBMS architecture for this Web-based system would be the three-tier architecture for web applications. This is because in this architecture the client is actually a GUI that the user interacts with. The data about the airline reservations is held in a database that the web server that the GUI lives on

interacts with. The web interface calls upon functions from the web server, but the application logic and the GUI do not have to live on the same machine.

UNIT-II

4 a) Explain the naming conventions and notations used in ER diagram.

L3 CO2 5 M

Naming Conventions and Notations in ER Diagrams

An **Entity-Relationship (ER) Diagram** visually represents the data and relationships in a database using a set of standard symbols and **naming rules**.

Naming Conventions

Entity-Relationship (ER) Diagram: Naming Conventions and Notations

ER diagrams are a **visual representation** of database structure, showing **entities, attributes, relationships, and constraints**. Below are the standard naming rules and notations used.

A. Entities (Tables)

- **Definition:** Real-world objects (e.g., Student, Employee, Order).
- **Notation:**
 - **Rectangle** (Strong Entity)
 - **Double Rectangle** (Weak Entity – depends on another entity)
- **Naming Rules:**
 - Use **singular nouns** (Course, not Courses).
 - **Capitalize** (Employee, not employee).

B. Attributes (Columns)

- **Definition:** Properties describing an entity (e.g., student_id, name).
- **Notation:**
 - **Oval** (Simple Attribute)
 - **Double Oval** (Multi-valued, e.g., phone_numbers)
 - **Dashed Oval** (Derived, e.g., age from birth_date)
 - **Underlined** (Primary Key)
- **Naming Rules:**
 - **Lowercase with underscores** (order_date, not OrderDate).
 - Avoid spaces/special characters.

C. Relationships (Associations)

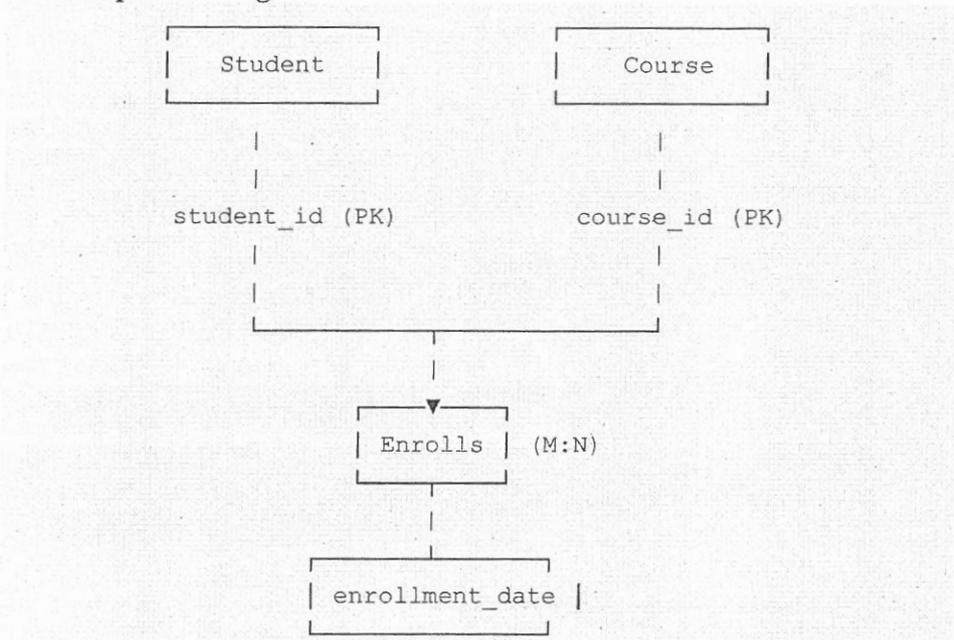
- **Definition:** How entities interact (e.g., "Student **enrolls in** Course").
- **Notation:**
 - **Diamond** (e.g., Enrolls, Manages)
 - **Lines connect entities** (with cardinality marks).
- **Naming Rules:**
 - **Verb phrases** (registers_for, works_in).
 - **Lowercase with underscores** (has_dependency).

D. Cardinality (Relationship Constraints)

- **Notation:**

- **1:1 (One-to-One)** → || or 1
- **1:N (One-to-Many)** → | (one) and Crow's Foot (☞)
or N (many)
- **M:N (Many-to-Many)** → Crow's Foot on both sides
- **Participation Constraints:**
 - **Single Line** (Partial Participation) → Optional
 - **Double Line** (Total Participation) → Mandatory

2. Example ER Diagram



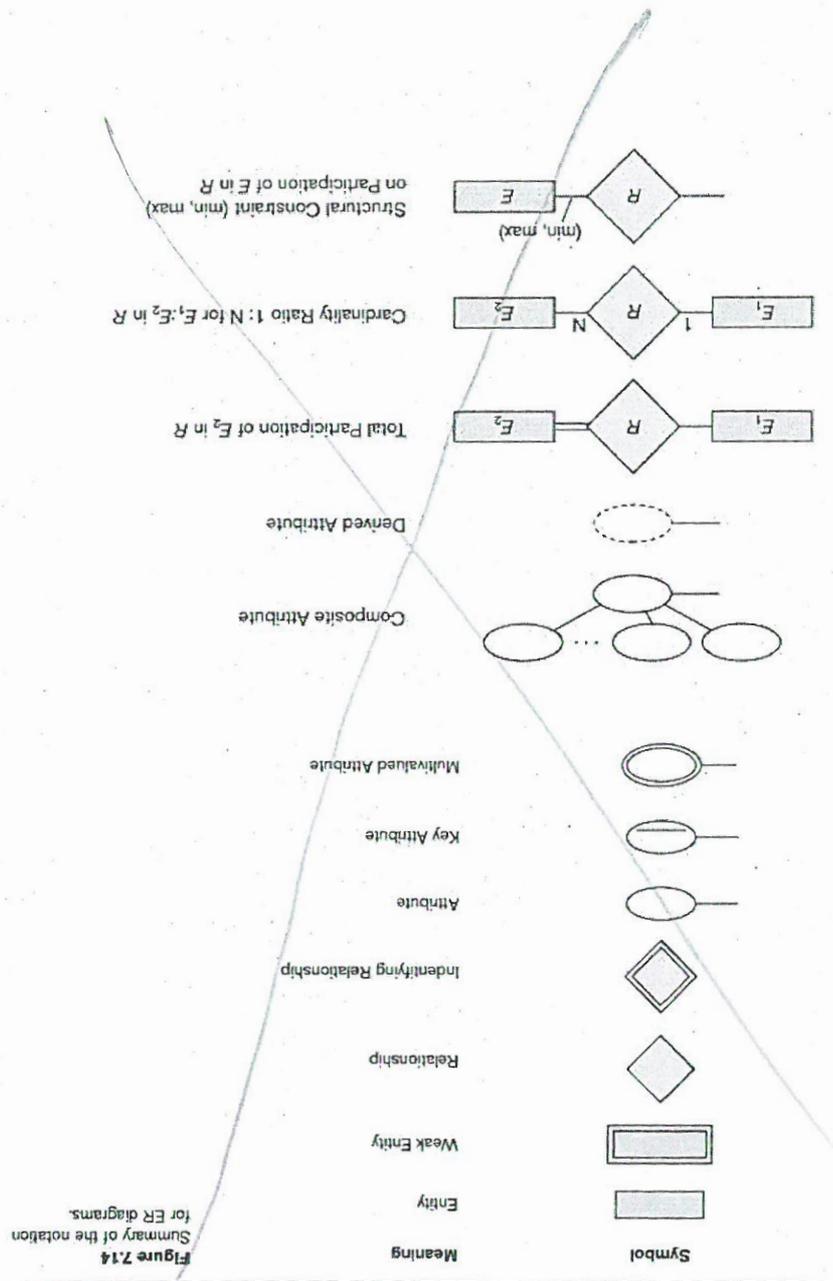
- **Entities:** Student, Course
- **Attributes:** student_id, course_id, enrollment_date
- **Relationship:** Enrolls (Many-to-Many)

3. Best Practices for Naming

- ✓ **Entities:** Singular & capitalized (Employee, not Employees).
- ✓ **Relationships:** Verb-based (manages, registers_for).
- ✓ **Attributes:** Lowercase, underscores (birth_date, not BirthDate).
- ✓ **Primary Keys:** Use `_id` suffix (employee_id, order_id).
- ✗ **Avoid:** Spaces (order date → order_date), SQL keywords (user → app_user).

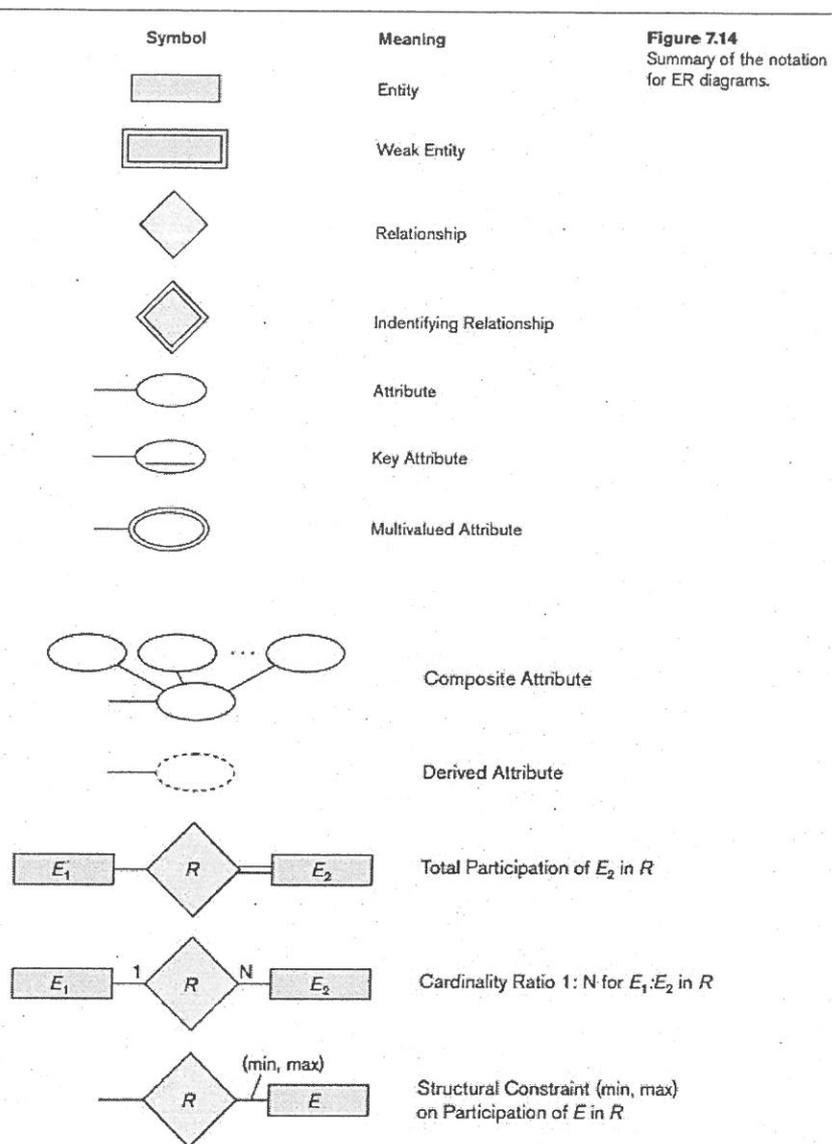
Design Choices for ER Conceptual Design

It is occasionally difficult to decide whether a particular concept in the miniworld should be modeled as an entity type, an attribute, or a relationship type. In this



Design Choices for ER Conceptual Design

It is occasionally difficult to decide whether a particular concept in the miniworld should be modeled as an entity type, an attribute, or a relationship type. In this



4. b) Explain Strong Entity Sets and Weak Entity Sets with examples.

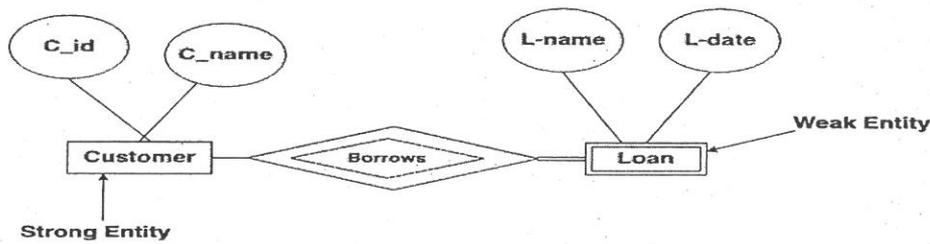
L3 CO2 5 M

Strong Entity

A strong entity is not dependent on any other entity in the schema. A strong entity will always have a primary key. Strong entities are represented by a single rectangle. The relationship of two strong entities is represented by a single diamond. Various strong entities, when combined together, create a strong entity set.

Weak Entity

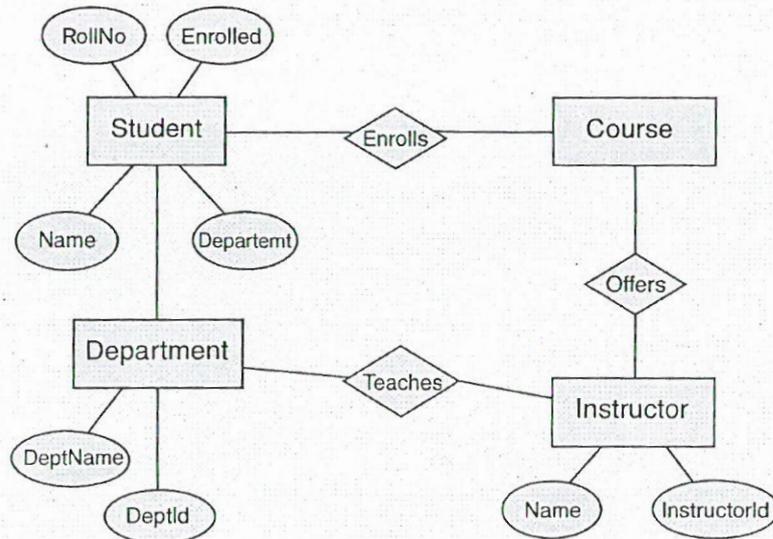
A weak entity is dependent on a strong entity to ensure its existence. Unlike a strong entity, a weak entity does not have any primary key. It instead has a partial discriminator key. A weak entity is represented by a double rectangle. The relation between one strong and one weak entity is represented by a double diamond. This relationship is also known as an identifying relationship.



OR

5. Draw an E-R diagram for a University Database, identify the key entities and the relationships between them **L4 CO4 10 M**

E-R diagram for a University Database:



Key Entities:

1. **Student**
 - o **Attributes:** RollNo (Primary Key), Name, Department
2. **Course**
 - o **Attributes:** CourseID (assumed), Name
3. **Instructor**
 - o **Attributes:** InstructorID (Primary Key), Name
4. **Department**
 - o **Attributes:** DeptID (Primary Key), DeptName

Relationships:

1. **Enrolls**
 - o Between Student and Course
 - o Represents which students are enrolled in which courses.
2. **Offers**
 - o Between Instructor and Course
 - o Indicates which instructor offers which course.
3. **Teaches**
 - o Between Department and Instructor
 - o Shows that instructors are associated with a department.

- The diagram uses standard ER notation: rectangles for entities, diamonds for relationships, ellipses for attributes, and underlined attributes for keys.
- Total or partial participation and cardinality can be added if more constraints are needed.

UNIT-III

6. Consider the following tables:

Student_Details (**Reg_no**, Name, Address, Phone_No, Grade)

Project_Details (**Project Code**, Project_Name, Project_Cost)

Student_Project (**Reg_No**, **Project_Code**)

i. Write SQL statement to create SQL tables with suitable integrity constraints.

ii. Write SQL statement to retrieve the names of the students who works on more than one project

L4 CO4 10 M

i. SQL Statements to Create Tables with Integrity Constraints

1. Create Student_Details Table

```
CREATE TABLE Student_Details ( Reg_no INT PRIMARY KEY, Name
VARCHAR(100) NOT NULL, Address VARCHAR(255), Phone_No
VARCHAR(15), Grade CHAR(1));
```

2. Create Project_Details Table

```
CREATE TABLE Project_Details ( Project_Code INT PRIMARY KEY,
Project_Name VARCHAR(100) NOT NULL, Project_Cost
DECIMAL(10, 2));
```

3. Create Student_Project Table (junction table for many-to-many relationship)

```
CREATE TABLE Student_Project ( Reg_no INT, Project_Code INT,
PRIMARY KEY (Reg_no, Project_Code), FOREIGN KEY (Reg_no)
REFERENCES Student_Details(Reg_no), FOREIGN KEY
(Project_Code) REFERENCES Project_Details(Project_Code));
```

ii. SQL Query to Retrieve Names of Students Working on More Than One Project

```
SELECT SD.Name
FROM Student_Details SD
JOIN Student_Project SP ON SD.Reg_no = SP.Reg_no
GROUP BY SD.Reg_no, SD.Name
HAVING COUNT(SP.Project_Code) > 1;
```

- The query joins Student_Details with Student_Project.
- It groups by Reg_no and Name.
- The HAVING clause filters students with a project count greater than 1.

OR

7 a) Enumerate various select and projection operations on relations with relational algebra. L3 CO2 5 M

Select and Projection Operations in Relational Algebra

Relational algebra provides a formal foundation for querying relational databases. Two fundamental operations are:

1. Select Operation (σ)

Purpose:

Retrieves rows (tuples) from a relation that satisfy a given predicate (condition).

Notation:

σ <condition>(Relation)

Examples:

1. Select students with Grade 'A'

σ Grade = 'A' (Student_Details)

2. Select projects with cost > 50000

σ Project_Cost > 50000 (Project_Details)

2. Projection Operation (π)

Purpose:

Retrieves columns (attributes) from a relation.

Notation:

π <attribute_list>(Relation)

Examples:

1. List all student names

π Name (Student_Details)

2. Get all unique project names and their costs

π Project_Name, Project_Cost (Project_Details)

Combined Example:

List names of students with Grade = 'A'

π Name (σ Grade = 'A' (Student_Details))

7.b) Discuss various set operations in relational algebra with examples.

L3 CO2 5 M

Set Operations in Relational Algebra

Relational algebra provides a set of operations that are fundamental for querying relational databases. Set theory operations—such as **union**, **intersection**, and **difference**—are commonly used to combine or compare sets of data. These operations are powerful tools to perform complex queries in a relational database, especially when dealing with multiple relations (tables) and performing operations like finding common data, excluding data, or combining results.

i. Union (\cup)

The **union** operation combines the tuples (rows) from two relations (tables), but only includes distinct tuples. It requires that the two relations being unioned must have the same **arity** (i.e., the same number of columns) and compatible **domains** (i.e., the data types of corresponding columns must be the same).

Syntax:

- R1 \cup R2

Example:

Consider two tables: Employee and Contractor.

Employee Table:

EmployeeID	Name	Department
101	Alice	HR
102	Bob	Engineering
103	Charlie	HR

Contractor Table:

EmployeeID	Name	Department
201	David	HR
202	Eve	Engineering
103	Charlie	HR

We want to find all the distinct employees (whether full-time or contractors) from both tables.

Query (Union):

sql

```
SELECT EmployeeID, Name, Department FROM Employee
UNION
SELECT EmployeeID, Name, Department FROM Contractor;
```

Result:

EmployeeID	Name	Department
101	Alice	HR
102	Bob	Engineering
103	Charlie	HR
201	David	HR
202	Eve	Engineering

The `UNION` operation returns a combined list of employees and contractors, removing any duplicates (in this case, Charlie appears in both tables but only once in the result).

ii. Intersection (\cap)

The **intersection** operation returns the set of tuples that are present in both relations. Like the union operation, the relations involved must have the same number of columns and compatible domains.

Syntax:

- $R1 \cap R2$

Example:

Let's use the same `Employee` and `Contractor` tables. We want to find all individuals who are both full-time employees and contractors (i.e., they appear in both the `Employee` and `Contractor` tables).

Query (Intersection):

sql

```
SELECT EmployeeID, Name, Department FROM Employee
INTERSECT
SELECT EmployeeID, Name, Department FROM Contractor;
```

Result:

EmployeeID	Name	Department
103	Charlie	HR

The `INTERSECT` operation returns only the tuple for Charlie, as he appears in both the `Employee` and `Contractor` tables.

iii. Difference (-)

The **difference** operation returns the set of tuples that are in the first relation but not in the second. Like the union and intersection operations, the relations involved must have the same number of columns and compatible domains.

Syntax:

- $R1 - R2$

Example:

Let's use the same tables, but this time, we want to find employees who are not contractors. This can be achieved using the difference operation.

Query (Difference):

```
sql
SELECT EmployeeID, Name, Department FROM Employee
MINUS
SELECT EmployeeID, Name, Department FROM Contractor;
```

Result:

EmployeeID	Name	Department
101	Alice	HR
102	Bob	Engineering

The `MINUS` operation returns the set of tuples that are in the `Employee` table but not in the `Contractor` table. Alice and Bob are full-time employees, but not contractors, so they appear in the result.

UNIT-IV

8 a) Explain the process of decomposition using multivalued dependencies with suitable example. L3 CO3 5M

Multi-Valued Dependency (MVD)

A multi-valued dependency (MVD) occurs in a relational database when one attribute (or a set of attributes) determines a set of values for another attribute, but the values in the determined set are independent of each other.

More formally, for a relation R , a multi-valued dependency between attributes X and Y is denoted as $X \twoheadrightarrow Y$ if:

- For each value of **X**, the corresponding values of **Y** are fully determined, but the values in **Y** do not depend on each other.
- The values of **Y** are independent from each other, meaning that the combination of **X** and **Y** does not imply any further dependency.

In essence, $X \twoheadrightarrow Y$ means that if two rows have the same value for **X**, then the values for **Y** are independent of each other, and each combination of **X** and **Y** can appear in multiple rows without causing any inconsistency.

Example of Multi-Valued Dependency:

Consider a table that contains information about students, their skills, and their languages.

StudentID	Skill	Language
1	Programming	Spanish
1	Programming	French
1	Data Analysis	Spanish
1	Data Analysis	French
2	Programming	English
2	Data Analysis	German

Here, $\text{StudentID} \twoheadrightarrow \text{Skill}$ and $\text{StudentID} \twoheadrightarrow \text{Language}$ are both multi-valued dependencies. A single **StudentID** can have multiple values for both **Skill** and **Language** independently, leading to repetition of the **StudentID** in multiple rows.

This redundancy results in wasted storage space and can make the database difficult to maintain because updating one value requires updating multiple rows.

4NF Resolves Multi-Valued Dependencies

Fourth Normal Form (4NF) is designed to eliminate multi-valued dependencies that cause redundancy in a relational database. A table is in 4NF if:

- It is in **Boyce-Codd Normal Form (BCNF)**.
- It has no multi-valued dependencies.

In other words, 4NF requires that **if there is a multi-valued dependency, the relation must be decomposed** into smaller relations so that each multi-valued dependency is represented separately. This eliminates the redundancy caused by the independent values for attributes that are multi-valued.

Decomposing to Achieve 4NF

To bring a relation into 4NF, we need to decompose the relation into multiple smaller relations such that each multi-valued dependency is handled in a separate table, ensuring that each combination of multi-valued attributes is stored independently without repetition.

In our example, the `StudentID` determines multiple `Skills` and multiple `Languages`. To normalize this into 4NF, we would decompose the table into two relations:

1. A table for **Skills** associated with a `StudentID`.
2. A table for **Languages** associated with a `StudentID`.

Decomposition to 4NF:

Skills Table:

StudentID	Skill
1	Programming
1	Data Analysis
2	Programming
2	Data Analysis

Languages Table:

StudentID	Language
1	Spanish
1	French
2	English
2	German

Now, the multi-valued dependency is handled correctly. Each table stores independent sets of values, and there is no unnecessary redundancy. If we need to update the language or skill of a student, we only need to update one row in the respective table, which improves data integrity and reduces redundancy.

Multi-valued dependencies can lead to redundancy in a database by causing repeated data for combinations of attributes. **Fourth Normal Form (4NF)** addresses this by decomposing tables in such a way that multi-valued dependencies are eliminated, thus improving data integrity, reducing redundancy, and enhancing the overall efficiency of the database schema.

8.b) Define normalization. Why do we need to normalize the database?

L3 CO3 5 M

Definition: Normalization

Normalization is a systematic process in database design used to:

- Organize data efficiently,
- Eliminate data redundancy (duplicate data),
- And reduce undesirable characteristics like **insertion, update, and deletion anomalies**.

It involves dividing large tables into smaller, related tables and defining relationships between them using **primary keys** and **foreign keys**.

Why Do We Need to Normalize the Database?

1. Eliminate Redundancy

- Repeated data wastes storage and leads to inconsistency.
- Normalization ensures that each piece of data is stored only once.

2. Prevent Anomalies

- **Insertion anomaly:** Can't add data because other required data is missing.
- **Update anomaly:** Need to update data in multiple places.
- **Deletion anomaly:** Deleting one piece of data may delete unintended information.

3. Improve Data Integrity

- Ensures that relationships between data are logically sound.
- Enforces rules through **constraints** and **keys**.

4. Maintain Consistency

- Changes made in one place reflect correctly throughout the database.

5. Efficient Data Organization

- Small, well-structured tables make querying, updating, and indexing more efficient.

Normalization is essential to ensure:

- Data is logically stored
- Redundancy is minimized
- Anomalies are prevented
- Database is easy to maintain and scale

OR

9 a) Describe the process of normalization in order to set the database to 3rd normal form. L3 CO3 5 M

Criteria for Third Normal Form (3NF)

A database schema is in **Third Normal Form (3NF)** if it meets the following criteria:

- **It is in Second Normal Form (2NF).**
- **It has no transitive dependencies:** A **transitive dependency** occurs when a non-prime attribute depends on another non-prime attribute through a third attribute. In other words, if $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$ is a transitive dependency, and the schema is not in 3NF.

Example of 3NF:

Consider a table with information about employees, their departments, and the department managers.

EmployeeID	DepartmentID	DepartmentName	ManagerName
1	D01	HR	Mr. White
2	D02	Finance	Ms. Black

Here:

- **DepartmentName** and **ManagerName** are functionally dependent on **DepartmentID**.
- **ManagerName** is dependent on **DepartmentName**, creating a transitive dependency because **DepartmentName** is dependent on **DepartmentID**, and **ManagerName** is dependent on **DepartmentName**.

To bring this schema into 3NF, we decompose it into two tables:

1. **Employee Table:**

EmployeeID	DepartmentID
1	D01
2	D02

2. **Department Table:**

DepartmentID	DepartmentName	ManagerName
D01	HR	Mr. White
D02	Finance	Ms. Black

Now, there are no transitive dependencies, as **ManagerName** depends directly on **DepartmentID** and not

indirectly via **DepartmentName**.

9.b) Write about loss-less join decomposition with an example.

L3 CO3 5 M

A **lossless join decomposition** (also called **non-loss decomposition**) ensures that if a relation **R** is decomposed into two or more relations **R1, R2, ...,** then we can **reconstruct the original relation R without losing any data** by performing a **natural join** on the decomposed relations.

Lossless Join Decomposition

If we decompose a relation **R** into relations **R1** and **R2**,

Example of Lossless Join Decomposition

Original Table (R): Employee_Project

EmpID	EmpName	Dept	ProjectID	ProjectName
E001	Alice	HR	P100	Payroll
E002	Bob	IT	P200	Database
E003	Carol	Finance	P100	Payroll

Functional Dependencies (FDs):

1. EmpID \rightarrow EmpName, Dept
2. ProjectID \rightarrow ProjectName
3. EmpID, ProjectID \rightarrow EmpName, Dept, ProjectName

Decomposition into R₁ and R₂:

R₁: Employee (Lossless because EmpID is a key)

EmpID	EmpName	Dept
E001	Alice	HR
E002	Bob	IT
E003	Carol	Finance

R₂: Project_Assignment (Lossless because (EmpID, ProjectID) is a key)

EmpID	ProjectID	ProjectName
E001	P100	Payroll
E002	P200	Database
E003	P100	Payroll

Verification of Lossless Join:

1. **Common Attribute:** EmpID (present in both R₁ and R₂).
2. **Check if EmpID is a superkey in either R₁ or R₂:**
 - In R₁, EmpID is a **primary key** (unique).
 - Hence, $R_1 \cap R_2 \rightarrow R_1$ holds.

Natural Join Result:

Employee \bowtie Project_Assignment

Output: Original table Employee_Project is perfectly reconstructed.

- **Lossless decomposition ensures no data loss when reconstructing the original table.**
- **The intersection of decomposed tables must be a superkey in at least one table.**

UNIT-V

10. Explain in detail about transaction management with an example.

L2 CO1 10 M

A **transaction** is a **sequence of operations** performed as a single logical unit of work on a database. These operations must either **all succeed** or **all fail**, to maintain the **integrity and consistency** of the database.

Why Transaction Management is Important:

- Ensures **data correctness** even in the presence of system crashes or concurrent users.
- Prevents problems such as **data inconsistency, lost updates, dirty reads, and phantom reads**.
- Guarantees that the database moves from **one consistent state to another**.

ACID Properties of Transactions:

Property	Description
Atomicity	A transaction is indivisible. If one part fails, the entire transaction fails.
Consistency	A transaction preserves database consistency (e.g., referential integrity).
Isolation	Concurrent transactions are isolated from each other.
Durability	Once a transaction is committed, changes are permanent, even after a crash.

Transaction States:

1. **Active** – Transaction is executing.
2. **Partially Committed** – Last statement executed, waiting for commit.
3. **Committed** – Changes permanently saved.
4. **Failed** – Error occurred during execution.
5. **Aborted** – Changes undone; transaction rolled back.

Example of Transaction: Bank Transfer

Scenario: Transfer ₹1000 from Account A to Account B

SQL Transaction:

```
BEGIN TRANSACTION;
```

```
UPDATE Accounts SET Balance = Balance - 1000 WHERE AccountID = 'A';  
UPDATE Accounts SET Balance = Balance + 1000 WHERE AccountID = 'B';
```

```
COMMIT;
```

Explanation:

- Deducts ₹1000 from Account A
- Adds ₹1000 to Account B
- Both actions must succeed together to maintain correctness

If system fails after first UPDATE:

- Without transaction management → A loses ₹1000, B gets nothing →
✗ Inconsistent
- With transaction management → Whole transaction is **rolled back** →
 Database remains consistent

Components of Transaction Management System:

Component	Role
Transaction Manager	Controls execution of transactions
Concurrency Control Manager	Manages concurrent transaction execution (e.g., 2PL)
Recovery Manager	Ensures durability using logs and checkpoints
Log Manager	Maintains logs for rollback or redo in case of failure

Concurrency Control Techniques:

- **Locks (2PL)** – Prevents simultaneous access to data
- **Timestamps** – Orders transactions logically
- **Optimistic Control** – Validates transactions before commit

Recovery Techniques:

- **Undo** – Rollback changes made by a failed transaction
- **Redo** – Reapply changes made by a committed transaction
- **Shadow Paging** – Maintains a shadow copy of data pages

Transaction management is **essential** for:

- Preserving **data integrity**
- Supporting **concurrent access**
- Ensuring **reliability and recovery**
- Providing **safe execution** of complex database operations

OR

11 a) What is concurrency control? Explain two phase locking protocol with an example. L2 CO1 5 M

Concurrency Control is a mechanism in a Database Management System (DBMS) that ensures **correct and consistent execution** of multiple **simultaneous transactions** without interfering with each other. It helps maintain **isolation**, one of the key **ACID** properties.

Two-Phase Locking (2PL) Protocol

Definition:

The **Two-Phase Locking protocol (2PL)** is a popular **concurrency control** technique that ensures **serializability** by dividing the transaction execution into two distinct phases:

Phases of 2PL:

1. **Growing Phase:**

- A transaction may **acquire locks** (read/write).
- It **cannot release** any locks.

2. **Shrinking Phase:**

- A transaction may **release locks**.
- It **cannot acquire** any new locks.

Once a transaction releases its first lock, it **cannot obtain any new locks**.

Example:

Transactions:

- T1: Reads A and writes A
- T2: Reads A and writes A

Assume both need **exclusive (write) lock** on A.

Scenario with 2PL:

Time	Action
t1	T1 acquires lock on A
t2	T1 writes A
t3	T1 releases lock on A (begin shrinking phase)
t4	T2 acquires lock on A
t5	T2 writes A
t6	T2 releases lock on A

✓ Transactions follow 2PL

✓ **Serializability is preserved**

Types of Locks Used:

Lock Type	Purpose
Shared Lock (S)	For reading (multiple can hold it)
Exclusive Lock (X)	For writing (only one holder)

Advantages of 2PL:

- Guarantees **conflict-serializability**
- Prevents many concurrency issues

Disadvantages:

- May cause **deadlocks** (when two transactions wait for each other's locks)

Concurrency Control, especially via the **Two-Phase Locking protocol**, is vital for:

- Ensuring **correctness**
- Avoiding anomalies
- Maintaining **transaction isolation**

11 b) Give a note on log based recovery.

L2 CO1 5 M

Definition:

Log-based recovery is a technique used in DBMS to ensure that the database can be restored to a consistent state after a system crash or failure. It uses a **log file** to record all changes made to the database.

What is a Log?

A **log** is a **sequential record** of all the operations performed by transactions. It is stored on **stable storage** and includes:

- **Transaction start**
- **Before and after values** of updates
- **Commit or abort information**

Typical Log Entry Format:

<Ti, X, old_value, new_value>

Where:

- Ti is the transaction ID
- X is the data item being modified
- old_value is the value before the update
- new_value is the value after the update

Types of Operations in the Log:

Operation	Description
<START Ti>	Transaction Ti has started
<Ti, X, old, new>	Update operation on X by Ti
<COMMIT Ti>	Transaction Ti successfully committed
<ABORT Ti>	Transaction Ti was aborted

Recovery Process:

Two major phases during recovery:

1. Undo (Rollback):

- For all **uncommitted transactions**
- Database is rolled back to the **old value**
- Uses **old_value** from log

2. Redo (Reapply):

- For all **committed transactions**
- Ensures all effects are **reflected** in the database
- Uses **new_value** from log

Log-based recovery is a **robust and reliable mechanism** that plays a vital role in maintaining **data integrity** and **recoverability** in a DBMS by using logs to **undo** or **redo** transactions depending on their commit status.