CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | |

Internal Assesment Test –II Answer Key
September 2024

Sub:	**Database Management System**						**Code:**	**22MCA21**	
Date:	**11/09/2024**	**Duration:**	90 mins	**Max Marks:**	50	**Sem:**	II	**Branch:**	MCA

Answer Any 5 **QUESTIONS**		Marks	OBE CO	OBE RBT
1)	**Explain the concept of an ER (Entity-Relationship) diagram in database design. Draw and describe an ER diagram for a university database that tracks students, courses, and instructors. Your diagramshould include the following:**	10	CO1, CO4	L2

Entities: Student, Course, Instructor
Relationships: Enrollment (between Student and Course), Teaching(between Instructor and Course)
Attributes:
For Student: Student_ID (primary key), Name, Age, Major For Course: Course_ID (primary key), Course_Name, Credits
For Instructor: Instructor_ID (primary key), Name, DepartmentFor Enrollment: Grade
Indicate any key attributes and participation constraints (total or partial). Use proper symbols for entities, attributes, and relationships.

ER Diagram Components

1. **Entities**:
   - **Student**: Represents the students enrolled in the university.
   - **Course**: Represents the courses offered by the university.
   - **Instructor**: Represents the instructors teaching the courses.
2. **Attributes**:
   - **Student**:
     - **Student_ID** (Primary Key)
     - Name
     - Age
     - Major
   - **Course**:
     - **Course_ID** (Primary Key)
     - Course_Name
     - Credits
   - **Instructor**:
     - **Instructor_ID** (Primary Key)
     - Name
     - Department
   - **Enrollment** (Relationship between Student and Course):
     - **Grade** (Attribute of Enrollment)
3. **Relationships**:
   - **Enrollment**: Connects Student and Course with a Grade attribute.

- **Teaching**: Connects Instructor and Course.

Participation Constraints

- A **Student** can enroll in multiple **Courses** (partial participation).
- A **Course** can have multiple **Students** enrolled (total participation).
- An **Instructor** can teach multiple **Courses** (partial participation).
- A **Course** must be taught by at least one **Instructor** (total participation).

Symbols

- **Entities** are represented by rectangles.
- **Attributes** are represented by ovals connected to their entity.
- **Relationships** are represented by diamonds connected to the related entities.

ER Diagram Description

1. Draw three rectangles labeled "Student," "Course," and "Instructor."
2. Connect the "Student" rectangle to a diamond labeled "Enrollment," which then connects to the "Course" rectangle.
   - Add an oval labeled "Grade" connected to the "Enrollment" diamond.
3. Connect the "Instructor" rectangle to a diamond labeled "Teaching," which connects to the "Course" rectangle.
4. For attributes, connect ovals to each entity rectangle:
   - **Student**: Connect ovals for "Student_ID," "Name," "Age," and "Major."
   - **Course**: Connect ovals for "Course_ID," "Course_Name," and "Credits."
   - **Instructor**: Connect ovals for "Instructor_ID," "Name," and "Department."

Diagram

OR

2) **Design an ER diagram for an online bookstore database. The systemshould manage information about customers, books, and orders.**
Ensure your diagram covers the following aspects:
Entities: Customer, Book, Order
Relationships: Places (between Customer and Order), Contains(between Order and Book)
Attributes:
For Customer: Customer_ID (primary key), Name, Email, PhoneFor Book: Book_ID (primary key), Title, Author, Price
For Order: Order_ID (primary key), Order_Date, Total_AmountIndicate key attributes and show participation constraints for the relationships. Also, specify the cardinalities of the relationships.

10

CO1, CO4

L2

ER Diagram Components

1. **Entities**:
   - **Customer**: Represents the customers purchasing books.
   - **Book**: Represents the books available for sale.

- **Order**: Represents the orders made by customers.

2. **Attributes**:
    - **Customer**:
        - **Customer_ID** (Primary Key)
        - Name
        - Email
        - Phone
    - **Book**:
        - **Book_ID** (Primary Key)
        - Title
        - Author
        - Price
    - **Order**:
        - **Order_ID** (Primary Key)
        - Order_Date
        - Total_Amount
3. **Relationships**:
    - **Places**: Connects Customer to Order.
    - **Contains**: Connects Order to Book.

Participation Constraints and Cardinalities

- A **Customer** can place multiple **Orders** (1-to-many relationship, total participation on the Order side).
- An **Order** is placed by exactly one **Customer** (1-to-1 relationship on the Customer side).
- An **Order** can contain multiple **Books** (1-to-many relationship).
- A **Book** can be included in multiple **Orders** (many-to-many relationship).

Symbols

- **Entities** are represented by rectangles.
- **Attributes** are represented by ovals connected to their respective entities.
- **Relationships** are represented by diamonds connected to the related entities.

ER Diagram Description

1. Draw three rectangles labeled "Customer," "Book," and "Order."
2. Connect the "Customer" rectangle to a diamond labeled "Places," which connects to the "Order" rectangle.
3. Connect the "Order" rectangle to a diamond labeled "Contains," which connects to the "Book" rectangle.
4. For attributes, connect ovals to each entity rectangle:
    - **Customer**: Connect ovals for "Customer_ID," "Name," "Email," and "Phone."
    - **Book**: Connect ovals for "Book_ID," "Title," "Author," and "Price."

Order: Connect ovals for "Order_ID," "Order_Date," and "Total_Amount."

Diagram

3)	**Discuss the importance of Entity-Relationship (ER) diagrams in database design. Explain the different types of attributes and relationships found in ER diagrams with examples. In your answer,include the following:** **1. Definition and purpose of ER diagrams in database systems.** **2. Types of attributes: Simple, Composite, Multivalued, andDerived (with examples).**	10	CO1, CO4	L1
	3. Types of relationships: One-to-One, One-to-Many, Many-to-Many (with examples). **4. The concept of key attributes and weak entities.** **1. Role of participation and cardinality constraints in ERdiagrams.** 1. An Entity-Relationship (ER) diagram is a visual tool used in database design to depict the structure of data, its entities (objects or things), attributes (characteristics), and relationships (associations) between entities. The primary purpose of an ER diagram is to help database designers and stakeholders understand the data requirements and relationships in a system, ensuring that the database structure is logically organized before implementation. This process leads to a well-structured, efficient, and normalized database design. 2. Types of Attributes Simple Attribute: These attributes cannot be divided further into smaller components. For example, Age or Employee_ID. Composite Attribute: These attributes consist of multiple components that can be broken down into simpler attributes. For example, Full Name can be split into First Name and Last Name. Multivalued Attribute: These attributes can hold multiple values for a single entity. For example, an entity Person can have a multivalued attribute Phone_Number, as a person may have more than one phone number. Derived Attribute: These attributes are not physically stored in the database but are derived from other attributes. For example, Age can be derived from the attribute Date_of_Birth. 3. Types of Relationships One-to-One (1:1): In this relationship, a single entity from one set is associated with a single entity from another set. For example, in a database of employees and their office locations, an Employee can be assigned to one Office and vice versa. One-to-Many (1): In this type, one entity from one set can be associated with multiple entities from another set. For example, a Customer can place many Orders, but each Order is placed by one Customer. Many-to-Many (M): Here, entities from both sets can have multiple associations. For instance,			

	in a university database, a Student can enroll in many Courses, and each Course can have many Students. 4. Key Attributes and Weak Entities Key Attributes: A key attribute is an attribute that uniquely identifies an entity in a set. For example, Student_ID uniquely identifies each student in a Student entity set. Weak Entities: A weak entity does not have sufficient attributes to form a primary key on its own and relies on a relationship with another (strong) entity. For example, in a system where Order is a weak entity, it depends on a Customer for identification, as Order_ID alone may not be unique. 5. Role of Participation and Cardinality Constraints in ER Diagrams Participation Constraints: This defines whether all or only some instances of an entity participate in a relationship. There are two types: Total Participation: All instances of an entity are involved in the relationship (represented by a double line). For example, every Employee must be assigned to a Department. Partial Participation: Some instances of the entity may not participate in the relationship (represented by a single line). For example, not every Student needs to register for a Club. Cardinality Constraints: These define the number of entities to which another entity can be associated through a relationship. 1:1 (One-to-One) 1 (One-to-Many) M (Many-to-Many) These constraints help database designers accurately represent real-world associations and ensure that relationships between entities are clearly understood during the database design process.			
OR				
4)	**Discuss any 5 different types of keys used in database management systems with example** In a Database Management System (DBMS), keys play a crucial role in defining how data is uniquely identified and organized. Here are five different types of keys commonly used in DBMS: 1. Primary Key A Primary Key is a column (or a set of columns) that uniquely identifies each record in a table. No two rows can have the same primary key value, and it cannot contain NULL values. Example: In an Employee table, Employee_ID can be the primary key as it uniquely identifies each employee.	10	CO5	L3

Employee_ID	Name	Department
101	Alice	HR
102	Bob	IT

Here, Employee_ID is the primary key.

2. Candidate Key
A Candidate Key is any column (or a combination of columns) that can be considered for the role of a primary key. Every candidate key is unique, but there can be multiple candidate keys in a table. One of them is selected as the primary key.

Example: In a Student table, both Student_ID and Email can serve as candidate keys because both are unique.

Student_ID	Name	Email
S001	John	john@gmail.com
S002	Sarah	sarah@yahoo.com

Here, both Student_ID and Email are candidate keys.

3. Foreign Key
A Foreign Key is a column (or set of columns) that creates a link between two tables. It refers to the primary key of another table, enforcing referential integrity between the tables.

Example: In an Order table, the Customer_ID can be a foreign key referencing the Customer_ID in the Customer table.

Customer Table:

Customer_ID	Name
C001	Alice
C002	Bob

Order Table:

Order_ID	Order_Date	Customer_ID
O101	2024-09-22	C001
O102	2024-09-23	C002

Here, Customer_ID in the Order table is a foreign key referencing the Customer_ID in the Customer table.

4. Super Key
A Super Key is any combination of columns that can uniquely identify each row in a table. A super key can consist of one or more columns and includes the primary key and candidate keys.

Example: In an Employee table, a combination of Employee_ID and Email could serve as a super key because together they uniquely identify each row.

Employee_ID Name Email

101	Alice	alice@company.com
102	Bob	bob@company.com

Here, both Employee_ID and Email together can form a super key.

5. Composite Key
A Composite Key is a key that consists of two or more columns that together uniquely identify a record in the table. It is used when no single column is unique by itself.

Example: In an Enrollment table that tracks students' enrollment in courses, a combination of Student_ID and Course_ID can form a composite key since no single column is unique.

Student_ID	Course_ID
S001	C101
S002	C102

Here, the combination of Student_ID and Course_ID uniquely identifies each row, forming a composite key.

Q. No.	Question	Marks	CO	Level
5)	**Define anomalies in the context of databases. Discuss the following types of anomalies with suitable examples:**	10	CO5	L2

1. **Insertion Anomaly**
2. **Update Anomaly**
3. **Deletion Anomaly**

In the context of databases, anomalies refer to problems that arise when performing operations (such as insertions, updates, or deletions) on a database that has not been properly normalized. These anomalies lead to data inconsistency, redundancy, and loss of information, especially in poorly designed tables.

1. Insertion Anomaly
An insertion anomaly occurs when inserting data into a database is problematic because some required data is not available, or inserting certain data requires additional, unrelated data to be entered unnecessarily.

Example:
Consider a table that stores information about Students and Courses in the same table:

Student_ID	Student_Name	Course_ID	Course_Name
S001	Alice	C101	Database
S002	Bob	C102	Networks

If a new course is introduced, say C103 - AI, and no students have yet enrolled in it, the database cannot store the course information unless we insert a dummy student record. This is an insertion anomaly because we cannot add the new course without a student enrolling in it, which is illogical.

Key Issue:

In a non-normalized database, inserting data may require unrelated or unnecessary data to be present, leading to inconsistent or invalid records.

2. Update Anomaly
An update anomaly occurs when data redundancy leads to multiple occurrences of the same information, making it necessary to update multiple rows when a single fact changes. If not all rows are updated, the database ends up with inconsistent information.

Example:
In the same table:

Student_ID	Student_Name	Course_ID	Course_Name
S001	Alice	C101	Database
S002	Bob	C101	Database

If the name of the course C101 (Database) changes to Advanced Database, the change must be applied to all rows where C101 is mentioned. If only one row is updated:

Student_ID	Student_Name	Course_ID	Course_Name
S001	Alice	C101	Advanced Database
S002	Bob	C101	Database

This inconsistency is caused by an update anomaly, where the information about the same course is not synchronized across the table.

Key Issue:
Without normalization, updating data in multiple places may lead to inconsistencies when not all related rows are updated.

3. Deletion Anomaly
A deletion anomaly occurs when deleting data inadvertently results in the loss of other important data, often because data that should be stored separately is combined in a single table.

Example:
Suppose a student Alice is removed from the course table:

Student_ID	Student_Name	Course_ID	Course_Name
S002	Bob	C101	Database

In this case, if Alice was the only student enrolled in C101 - Database, removing her data also deletes the course information. This results in the loss of data about the course, even though the course might still be relevant and offered to future students.

Key Issue:
A deletion anomaly causes unintended loss of related data when the deletion of a record accidentally removes additional, important

	information that should have been preserved.			
OR				
6)	**Define functional dependency in the context of a relational database. Explain the types of functional dependencies with examples** A functional dependency (FD) in a relational database is a constraint that describes the relationship between two attributes (or sets of attributes). In simpler terms, it expresses how one attribute (or a set of attributes) uniquely determines another attribute. It is denoted as: X $\rightarrow$ Y X→Y Where X is a determinant, and Y is the dependent. This means that for any two tuples (rows) in the relation, if the values of X are the same, then the values of Y must also be the same. Trivial Functional Dependency A functional dependency is said to be trivial if the dependent is a subset of the determinant. In other words, if Y is a part of X, then the dependency is trivial. Example: In the relation Student_ID, Name → Name, the dependency is trivial because Name is part of the left-hand side (determinant). Non-Trivial Functional Dependency A functional dependency is non-trivial if the dependent is not a subset of the determinant. In this case, X uniquely determines Y, but Y is not part of X. Example: In the relation Student_ID → Name, the dependency is non-trivial because Name is not a part of Student_ID. Full Functional Dependency A functional dependency is said to be a full functional dependency if removing any attribute from the determinant causes the dependency to break. In other words, the entire determinant is necessary for the dependency to hold. Example: In an Order table:	10	CO5	L3

Order_ID	Product_ID	Quantity
O001	P001	10
O002	P002	15

Here, the dependency Order_ID, Product_ID → Quantity is a full functional dependency because both Order_ID and Product_ID are required to determine Quantity. If you remove Product_ID, the quantity cannot be determined.

Partial Functional Dependency

A partial functional dependency occurs when a non-key attribute is functionally dependent on part of a composite primary key. It arises in cases where only part of the primary key is required to determine the dependent attribute.

Example: In an Enrollment table:

Student_ID	Course_ID	Instructor
S001	C101	Dr. Smith
S002	C102	Dr. Lee

Here, if Student_ID is part of a composite primary key, but Instructor only depends on Course_ID, it is a partial dependency because the full key (Student_ID, Course_ID) is not necessary to determine Instructor.

Transitive Functional Dependency

A transitive functional dependency occurs when a non-key attribute is dependent on another non-key attribute. This type of dependency arises indirectly, meaning one attribute is functionally dependent on a second, which is, in turn, functionally dependent on a third.

Example: In an Employee table:

Employee_ID	Department_ID	Department_Name
E001	D01	HR
E002	D02	IT

Here, Employee_ID → Department_ID and Department_ID → Department_Name. Hence, there is a transitive dependency Employee_ID → Department_Name through Department_ID.

Multivalued Dependency

A multivalued dependency occurs when one attribute in a table determines a set of values for another attribute, but this does not depend on a third attribute. This leads to redundancy because multiple rows may store the same values.

Example: In a Movie table:

Movie_ID	Actor	Genre
M001	Actor A	Action
M001	Actor B	Action
M001	Actor A	Adventure

	Here, Movie_ID → Actor and Movie_ID → Genre are independent of each other, leading to a multivalued dependency.			
7)	**Describe the process of normalization in detail up to the Third Normal Form (3NF).** Normalization is the process of organizing data in a relational database to reduce redundancy, improve data integrity, and ensure efficient data storage. It involves breaking down larger tables into smaller, related tables and defining relationships between them. The main objective of normalization is to eliminate data anomalies (insertion, update, and deletion anomalies) and ensure that the database structure supports consistency and accuracy of data. The process of normalization is typically carried out through a series of steps called normal forms. Each step builds on the previous one, progressively refining the database structure. First Normal Form (1NF) A relation (table) is said to be in First Normal Form (1NF) if it satisfies the following conditions: Atomicity: All the values in the table are atomic, meaning that each column contains indivisible values (no sets, lists, or arrays). Uniqueness of Rows: There are no duplicate rows in the table. Single-Valued Attributes: Each column must contain only a single value, i.e., no multivalued attributes. Example: Consider a table storing student information with courses: Student_ID Name Course_Enrolled S001 Alice Database, AI S002 Bob Networks Issues in 1NF: The Course_Enrolled column contains multiple values, which violates 1NF. Solution: Each student-course pair should be represented in a separate row: Student_ID Name Course_Enrolled S001 Alice Database S001 Alice AI S002 Bob Networks The table now satisfies 1NF as each attribute holds atomic values, and there are no multivalued attributes. Second Normal Form (2NF) A relation is in Second Normal Form (2NF) if: It is in 1NF. There is no partial dependency; that is, every non-prime attribute (an attribute that is not part of a candidate key) is fully functionally dependent on the primary key. This mainly applies to tables with composite primary keys.	10	CO5	L3

Example:
Consider a Student_Course table (already in 1NF):

Student_ID	Course_ID	Course_Name	Instructor
S001	C101	Database	Dr. Smith
S002	C102	Networks	Dr. Lee
S001	C102	Networks	Dr. Lee

Here, the composite key is (Student_ID, Course_ID). However, the attribute Course_Name and Instructor depend only on Course_ID, not on the full composite key, creating a partial dependency.

Solution: To eliminate partial dependencies, we split the table into two:

Student_Course:

Student_ID	Course_ID
S001	C101
S002	C102
S001	C102

Course_Details:

Course_ID	Course_Name	Instructor
C101	Database	Dr. Smith
C102	Networks	Dr. Lee

Now, the Student_Course table is in 2NF because there are no partial dependencies, and all non-prime attributes fully depend on the primary key.

Third Normal Form (3NF)
A relation is in Third Normal Form (3NF) if:

It is in 2NF.
There is no transitive dependency, meaning that no non-prime attribute is transitively dependent on the primary key. A transitive dependency occurs when one non-key attribute depends on another non-key attribute.
Example:
Consider the Course_Details table (in 2NF):

Course_ID	Course_Name	Instructor	Instructor_Department
C101	Database	Dr. Smith	CS
C102	Networks	Dr. Lee	IT

Here, Instructor_Department depends on Instructor, not directly on the primary key Course_ID, which creates a transitive dependency.

Solution: To eliminate transitive dependencies, we split the table again:

Course_Details:

Course_ID	Course_Name	Instructor
C101	Database	Dr. Smith
C102	Networks	Dr. Lee

Instructor_Details:

Instructor	Instructor_Department

	Dr. Smith CS Dr. Lee IT Now, the Course_Details table is in 3NF because there are no transitive dependencies, and every non-prime attribute depends only on the primary key.			
OR				
8)	***Student_Course(Student_ID, Student_Name, Course_ID, Course_Name, Instructor_ID, Instructor_Name)*** **Normalize above relation to BCNF. Explain each step in the process,identifying the keys, dependencies, and anomalies resolved at each stage.** Let's normalize the given relation Student_Course(Student_ID, Student_Name, Course_ID, Course_Name, Instructor_ID, Instructor_Name) step by step, from First Normal Form (1NF) to Boyce-Codd Normal Form (BCNF). We will identify the keys, functional dependencies, and anomalies at each stage and explain how we resolve them. Step 1: Unnormalized Relation The given relation is: Student_Course(Student_ID, Student_Name, Course_ID, Course_Name, Instructor_ID, Instructor_Name) Assumptions about Functional Dependencies: Student_ID → Student_Name: A student's ID uniquely determines the student's name. Course_ID → Course_Name, Instructor_ID, Instructor_Name: A course ID uniquely determines the course name, the instructor ID, and the instructor name. Instructor_ID → Instructor_Name: An instructor's ID uniquely determines the instructor's name. Step 2: First Normal Form (1NF)	10	CO5	L3

To satisfy 1NF, we need to ensure that all values are atomic and there are no repeating groups.

In this relation, there are no multi-valued attributes or repeating groups, so the table is already in 1NF.

Step 3: Second Normal Form (2NF)

A relation is in 2NF if it is in 1NF and all non-prime attributes are fully functionally dependent on the whole primary key. Partial dependencies need to be removed.

Identifying the Candidate Key:

The composite key here would be (Student_ID, Course_ID), since a student can enroll in multiple courses, and each course can have multiple students. Therefore, the combination of Student_ID and Course_ID uniquely identifies each record in the table.

Functional Dependencies:

Student_ID → Student_Name

Course_ID → Course_Name, Instructor_ID, Instructor_Name

Instructor_ID → Instructor_Name

Partial Dependencies:

Student_Name depends only on Student_ID (not on the full composite key Student_ID, Course_ID).

Course_Name, Instructor_ID, and Instructor_Name depend only on Course_ID (not on the full composite key Student_ID, Course_ID).

These are partial dependencies and violate 2NF.

Decomposition into 2NF:

We split the relation into smaller tables to remove partial dependencies:

Student Table:

Student(Student_ID, Student_Name)

This removes the partial dependency Student_ID → Student_Name.

Course Table:

Course(Course_ID, Course_Name, Instructor_ID)

This removes the partial dependency Course_ID → Course_Name, Instructor_ID.

Instructor Table (for instructor details):

Instructor(Instructor_ID, Instructor_Name)

This removes the partial dependency Instructor_ID → Instructor_Name.

Student_Course Table (for the relationship between students and courses):

Student_Course(Student_ID, Course_ID)

Let's normalize the given relation Student_Course(Student_ID, Student_Name, Course_ID, Course_Name, Instructor_ID, Instructor_Name) step by step, from First Normal Form (1NF) to Boyce-Codd Normal Form (BCNF). We will identify the keys, functional dependencies, and anomalies at each stage and explain how we resolve them.

Step 1: Unnormalized Relation

The given relation is:

scss

Copy code

Student_Course(Student_ID, Student_Name, Course_ID, Course_Name, Instructor_ID, Instructor_Name)

Assumptions about Functional Dependencies:

Student_ID → Student_Name: A student's ID uniquely determines the student's name.

Course_ID → Course_Name, Instructor_ID, Instructor_Name: A course ID uniquely determines the course name, the instructor ID, and the instructor name.

Instructor_ID → Instructor_Name: An instructor's ID uniquely determines the instructor's name.

Step 2: First Normal Form (1NF)

To satisfy 1NF, we need to ensure that all values are atomic and there are no repeating groups.

In this relation, there are no multi-valued attributes or repeating groups, so the table is already in 1NF.

Step 3: Second Normal Form (2NF)

A relation is in 2NF if it is in 1NF and all non-prime attributes are fully functionally dependent on the whole primary key. Partial dependencies need to be removed.

Identifying the Candidate Key:

The composite key here would be (Student_ID, Course_ID), since a student can enroll in multiple courses, and each course can have multiple students. Therefore, the combination of Student_ID and Course_ID uniquely identifies each record in the table.

Functional Dependencies:

Student_ID → Student_Name

Course_ID → Course_Name, Instructor_ID, Instructor_Name

Instructor_ID → Instructor_Name

Partial Dependencies:

Student_Name depends only on Student_ID (not on the full composite key Student_ID, Course_ID).

Course_Name, Instructor_ID, and Instructor_Name depend only on Course_ID (not on the full composite key Student_ID, Course_ID).

These are partial dependencies and violate 2NF.

Decomposition into 2NF:

We split the relation into smaller tables to remove partial dependencies:

Student Table:

scss

Copy code

Student(Student_ID, Student_Name)

This removes the partial dependency Student_ID → Student_Name.

Course Table:

scss

Copy code

Course(Course_ID, Course_Name, Instructor_ID)

This removes the partial dependency Course_ID → Course_Name, Instructor_ID.

Instructor Table (for instructor details):

scss

Copy code

Instructor(Instructor_ID, Instructor_Name)

This removes the partial dependency Instructor_ID → Instructor_Name.

Student_Course Table (for the relationship between students and courses):

scss

Copy code

Student_Course(Student_ID, Course_ID)

At this point, all relations are in 2NF because there are no partial dependencies.

Step 4: Third Normal Form (3NF)

A relation is in 3NF if it is in 2NF and there are no transitive dependencies.

Checking for Transitive Dependencies:

In the Course table, Instructor_ID → Instructor_Name is a transitive dependency because Course_ID → Instructor_ID and Instructor_ID → Instructor_Name. This violates 3NF since Instructor_Name depends on a non-key attribute (Instructor_ID), not directly on Course_ID.

Decomposition into 3NF:

To remove the transitive dependency, we split the Course table further:

Course Table (after removing Instructor_Name):

Course(Course_ID, Course_Name, Instructor_ID)

The Instructor Table remains the same:

Instructor(Instructor_ID, Instructor_Name)

Let's normalize the given relation Student_Course(Student_ID, Student_Name, Course_ID, Course_Name, Instructor_ID, Instructor_Name) step by step, from First Normal Form (1NF) to Boyce-Codd Normal Form (BCNF). We will identify the keys, functional dependencies, and anomalies at each stage and explain how we resolve them.

Step 1: Unnormalized Relation

The given relation is:

scss

Copy code

Student_Course(Student_ID, Student_Name, Course_ID, Course_Name, Instructor_ID, Instructor_Name)

Assumptions about Functional Dependencies:

Student_ID → Student_Name: A student's ID uniquely determines the student's name.

Course_ID → Course_Name, Instructor_ID, Instructor_Name: A course ID uniquely determines the course name, the instructor ID, and the instructor name.

Instructor_ID → Instructor_Name: An instructor's ID uniquely determines the instructor's name.

Step 2: First Normal Form (1NF)

To satisfy 1NF, we need to ensure that all values are atomic and there are no repeating groups.

In this relation, there are no multi-valued attributes or repeating groups, so the table is already in 1NF.

Step 3: Second Normal Form (2NF)

A relation is in 2NF if it is in 1NF and all non-prime attributes are fully functionally dependent on the whole primary key. Partial dependencies need to be removed.

Identifying the Candidate Key:

The composite key here would be (Student_ID, Course_ID), since a student can enroll in multiple courses, and each course can have multiple students. Therefore, the combination of Student_ID and Course_ID uniquely identifies each record in the table.

Functional Dependencies:

Student_ID → Student_Name

Course_ID → Course_Name, Instructor_ID, Instructor_Name

Instructor_ID → Instructor_Name

Partial Dependencies:

Student_Name depends only on Student_ID (not on the full composite key Student_ID, Course_ID).

Course_Name, Instructor_ID, and Instructor_Name depend only on Course_ID (not on the full composite key Student_ID, Course_ID).

These are partial dependencies and violate 2NF.

Decomposition into 2NF:

We split the relation into smaller tables to remove partial dependencies:

Student Table:

scss

Copy code

Student(Student_ID, Student_Name)

This removes the partial dependency Student_ID → Student_Name.

Course Table:

scss

Copy code

Course(Course_ID, Course_Name, Instructor_ID)

This removes the partial dependency Course_ID → Course_Name, Instructor_ID.

Instructor Table (for instructor details):

scss

Copy code

Instructor(Instructor_ID, Instructor_Name)

This removes the partial dependency Instructor_ID → Instructor_Name.

Student_Course Table (for the relationship between students and courses):

scss

Copy code

Student_Course(Student_ID, Course_ID)

At this point, all relations are in 2NF because there are no partial dependencies.

Step 4: Third Normal Form (3NF)

A relation is in 3NF if it is in 2NF and there are no transitive dependencies.

Checking for Transitive Dependencies:

In the Course table, Instructor_ID → Instructor_Name is a transitive dependency because Course_ID → Instructor_ID and Instructor_ID → Instructor_Name. This violates 3NF since Instructor_Name depends on a non-key attribute (Instructor_ID), not directly on Course_ID.

Decomposition into 3NF:

To remove the transitive dependency, we split the Course table further:

Course Table (after removing Instructor_Name):

scss

Copy code

Course(Course_ID, Course_Name, Instructor_ID)

The Instructor Table remains the same:

scss

Copy code

Instructor(Instructor_ID, Instructor_Name)

Now, all tables are in 3NF because there are no transitive dependencies.

Step 5: Boyce-Codd Normal Form (BCNF)

A relation is in BCNF if it is in 3NF and, for every functional

	dependency X → Y, X is a superkey (i.e., X must be a candidate key or a superset of a candidate key). Checking for BCNF Violations: Let's examine each table: Student Table: Functional dependency: Student_ID → Student_Name Student_ID is the primary key, so the table is in BCNF. Course Table: Functional dependency: Course_ID → Course_Name, Instructor_ID Course_ID is the primary key, so the table is in BCNF. Instructor Table: Functional dependency: Instructor_ID → Instructor_Name Instructor_ID is the primary key, so the table is in BCNF. Student_Course Table: Functional dependency: Student_ID, Course_ID → (no non-prime attributes) The composite key (Student_ID, Course_ID) is a superkey, so the table is in BCNF.			
9)	**Compare the process of normalizing a relation from 1NF to 3NF versus normalizing it to BCNF.**	10	CO5	L3

Normalizing a relation from **1NF (First Normal Form)** to **3NF (Third Normal Form)** and **BCNF (Boyce-Codd Normal Form)** both aim to reduce redundancy and maintain data integrity, but they differ in terms of their criteria and strictness.

1NF to 3NF Process

1. **1NF** (First Normal Form):
   - Eliminate any **repeating groups** or arrays, ensuring each column contains atomic values.
   - Ensure that each table has a **primary key** and every entry in the table is unique and non-null.
2. **2NF** (Second Normal Form):
   - Achieve 2NF by eliminating **partial dependencies**.
   - A partial dependency occurs when a non-prime attribute (non-key attribute) depends on a **part** of a composite key (only in cases where the primary key is composite).
   - Decompose the relation into smaller relations so that each non-prime attribute depends on the **whole key** and not just part of it.
3. **3NF** (Third Normal Form):
   - Achieve 3NF by eliminating **transitive dependencies**.
   - A transitive dependency exists when a non-prime attribute depends on another non-prime attribute, which in turn depends on the primary key.
   - In 3NF, all non-key attributes must depend directly on the **primary key** only and not on any other non-key attribute.
   - If a transitive dependency is found, the relation is split so that each non-key attribute depends directly on the key.

1NF to BCNF Process

1. **1NF**:
   - As with normalization to 3NF, begin by ensuring the relation is in 1NF by eliminating repeating groups and making sure that every column contains atomic values.
2. **2NF**:
   - Achieve 2NF by eliminating partial dependencies, just as in the process of normalizing to 3NF.
3. **BCNF** (Boyce-Codd Normal Form):
   - BCNF is a stricter form of 3NF.
   - A relation is in BCNF if, for every non-trivial functional dependency X→YX \to YX→Y, XXX must be a **superkey** (i.e., XXX contains a candidate key).
   - BCNF handles certain anomalies that can still exist in 3NF when the relation has more than one candidate key and a non-prime attribute depends on something other than the primary key.

OR

10) ## Explain Briefly about all 5 types of Normalization

10 | CO5 | L2

Normalization is the process of organizing a database to reduce redundancy and improve data integrity. There are five types of normalization, each building on the previous form:

1. First Normal Form (1NF):

- **Goal**: Eliminate repeating groups; ensure atomic values.

- **Requirements**:
  - Each column contains only **atomic values** (no multi-valued attributes).
  - Entries in columns are of the **same data type**.
  - Each record must be **unique**, identified by a primary key.
- **Example**: A table where each cell contains a single value and no sets of values (e.g., {apple, banana}) within a cell.

2. Second Normal Form (2NF):

- **Goal**: Eliminate partial dependencies (dependencies on part of a composite key).
- **Requirements**:
  - Must be in **1NF**.
  - All non-key attributes must depend on the **whole** primary key, not just part of it (if the key is composite).
- **Example**: If a table has a composite primary key, no non-key attribute should depend on just one part of the composite key.

3. Third Normal Form (3NF):

- **Goal**: Eliminate transitive dependencies (dependencies on non-prime attributes).
- **Requirements**:
  - Must be in **2NF**.
  - No non-key attribute should depend on another non-key attribute (i.e., no transitive dependencies).
- **Example**: If A→B→CA \to B \to CA→B→C, where AAA is the key, CCC should depend directly on AAA, not via BBB.

4. Boyce-Codd Normal Form (BCNF):

- **Goal**: Handle certain anomalies not covered by 3NF when there are multiple candidate keys.
- **Requirements**:
  - Must be in **3NF**.
  - For every functional dependency X→YX \to YX→Y, XXX must be a **superkey** (a candidate key or a set containing a candidate key).
- **Example**: In a relation where a non-prime attribute depends on a candidate key that is not the primary key, the relation must be decomposed to satisfy BCNF.

5. Fourth Normal Form (4NF):

- **Goal**: Eliminate multi-valued dependencies.
- **Requirements**:
  - Must be in **BCNF**.
  - No relation should have more than one **multi-valued dependency** (where one attribute determines a set of other attributes).
- **Example**: If a table has attributes where one attribute determines multiple independent values for another, it should be decomposed.

6. Fifth Normal Form (5NF) (also known as Project-Join Normal Form):

- **Goal**: Eliminate join dependencies.
- **Requirements**:

	◦ Must be in **4NF**. ◦ No table should have join dependencies that lead to redundant information. It ensures that data is reconstructed correctly without any anomalies after decomposition. • **Example**: A relation is decomposed in such a way that, after joining the decomposed relations, the original relation can be reconstructed without loss of information.			