ENSEM IMP DATABASE MANAGMENT SYSTEM UNIT -3

Q.1] What is the impact of insert, update & delete anomaly on overall design of database? How normalization is used to remove these anomalies?

ANS:

1. **Data Redundancy and Inconsistency: Without normalization, data redundancy is likely, leading to inconsistencies. For example, if you store the same information in multiple places, updating one instance might be forgotten in another, causing discrepancies.**
2. **Insert Anomaly: In a non-normalized database, inserting new data might be problematic as it could require adding redundant information or leave certain fields empty. This can lead to incomplete or inaccurate records.**
3. **Update Anomaly: Updating data in a non-normalized database can be error-prone. Modifying one piece of information might be overlooked in other related records, leading to inconsistencies in the database.**
4. **Delete Anomaly: Deleting data in a non-normalized database can unintentionally remove more information than intended. For instance, deleting a customer might also erase details about their orders if not properly handled.**

How Normalization Addresses These:

1. **Minimizing Redundancy: Normalization involves breaking down large tables into smaller ones and establishing relationships between them. This reduces redundancy by ensuring that data is stored in a normalized, efficient manner.**
2. **Ensuring Data Integrity: Normalization eliminates insertion anomalies by organizing data in a way that each piece of information is stored only once. This ensures that inserting new data is straightforward and doesn't require redundant information.**
3. **Consistent Updates: By organizing data into separate tables based on their relationships, normalization ensures that updates are consistent. You only need to update information in one place, reducing the risk of inconsistencies.**
4. **Preventing Delete Anomalies: Normalization prevents delete anomalies by ensuring that data is stored in a way that deleting a record only removes information about that specific entity and doesn't inadvertently erase data related to other entities.**

Q.2] Explain different features of good relational database design.

ANS:

1. **Normalization: The database should be normalized to minimize redundancy and dependency. This involves organizing data into related tables to avoid data duplication and ensure data integrity.**
2. **Consistency: A well-designed database maintains consistency in data across all tables. Changes or updates to data should be applied uniformly to prevent inconsistencies.**
3. **Data Integrity: The design should enforce data integrity constraints, such as primary and foreign keys, to ensure that data is accurate and reliable. This prevents the entry of invalid or inconsistent information.**
4. **Relationships: Establishing proper relationships between tables is crucial. Foreign keys should link tables based on logical connections between entities, supporting data retrieval and maintenance.**
5. **Scalability: The design should be scalable to accommodate future growth in data volume and user demands. It should handle increased data without significant performance degradation.**
6. **Performance: A good design considers performance optimization, including appropriate indexing, query optimization, and efficient use of storage. This ensures that database operations are executed quickly and with minimal resource consumption.**
7. **Flexibility: The design should be flexible enough to adapt to changing requirements without requiring major restructuring. This allows for easier modifications and updates as the needs of the system evolve.**
8. **Normalization Forms: Ensuring the database is in at least Third Normal Form (3NF) or Boyce-Codd Normal Form (BCNF) helps in eliminating data redundancy and dependency, contributing to a more efficient and reliable design.**
9. **Documentation: Thorough documentation of the database schema, relationships, constraints, and business rules is essential. This aids in understanding and maintaining the database over time.**
10. **Security: Implementing proper security measures, including user authentication and authorization, helps protect sensitive data from unauthorized access or modifications.**

Q.3] Explain following Codd's rules with suitable examples : Guaranteed Access Rule, Comprehensive Data Sub-Language Rule, High-Level Insert, Update, and Delete Rule

ANS:

1. **Guaranteed Access Rule:**
    - **This rule ensures that each data value in the database is accessible by specifying a table name, primary key value, and column name.**
    - **Example: In a relational database, if you have a table called "Employees" with columns like "EmployeeID," "Name," and "Salary," the Guaranteed Access Rule ensures that you can retrieve specific information like the salary of an employee with a known EmployeeID.**
2. **Comprehensive Data Sub-Language Rule:**
    - **This rule states that the DBMS must support a language that can express all the logical capabilities of the relational database.**
    - **Example: SQL (Structured Query Language) is a comprehensive data sub-language that allows you to perform various operations like selecting data (SELECT), inserting data (INSERT), updating data (UPDATE), and deleting data (DELETE). This language enables you to manipulate the data in a relational database comprehensively.**
3. **High-Level Insert, Update, and Delete Rule:**
    - **This rule states that the DBMS must provide high-level operations for inserting, updating, and deleting data, rather than just basic low-level operations.**
    - **Example: Instead of manually managing the details of how data is inserted, updated, or deleted, you can use high-level SQL statements. For instance, to insert a new record into a table, you can use the INSERT INTO statement, specifying the table and values. Similarly, the UPDATE and DELETE statements provide high-level ways to modify and remove data from the database.**

Q.4] Justify the impact of normalization on database? Explain 2nd normal form, 3rd normal form and BCNF with example

ANS: Normalization is a crucial process in database design that helps reduce data redundancy, dependency, and improves data integrity. Let's delve into the impact of normalization and explore the concepts of 2nd normal form (2NF), 3rd normal form (3NF), and Boyce-Codd Normal Form (BCNF) with examples:

Impact of Normalization:

1. **Reduction of Redundancy:**
   - **Normalization eliminates duplicate data by breaking down tables into smaller, related tables. This reduces data redundancy and ensures that each piece of information is stored in only one place.**
2. **Minimization of Data Dependency:**
   - **Normalization helps in reducing data dependency, ensuring that changes in one part of the database don't create anomalies or inconsistencies in other parts.**
3. **Enhanced Data Integrity:**
   - **By organizing data systematically, normalization enhances data integrity, reducing the chances of errors and inconsistencies in the database.**
4. **Simplified Data Maintenance:**
   - **Normalized databases are generally easier to maintain and update. The modular structure allows for changes to be made more efficiently without affecting the entire database.**

Now, let's discuss the normal forms with examples:

2nd Normal Form (2NF):

- **A table is in 2NF if it is in 1NF and all non-prime attributes are fully functionally dependent on the primary key.**

Example: Consider a table "Employee_Projects" with columns (EmployeeID, ProjectID, EmployeeName, ProjectName, HoursWorked).

- **In 1NF, there are no repeating groups, but the table has a composite primary key (EmployeeID, ProjectID).**
- **To achieve 2NF, we can create two tables: "Employees" (EmployeeID, EmployeeName) and "Projects" (ProjectID, ProjectName, HoursWorked).**
- **Now, both tables are in 2NF, and data redundancy is reduced.**

3rd Normal Form (3NF):

- **A table is in 3NF if it is in 2NF and all transitive dependencies are removed.**

Example: Consider a table "Student_Courses" with columns (StudentID, CourseID, StudentName, CourseInstructor).

- **In 2NF, the table is already decomposed. However, there's a transitive dependency: StudentName depends on StudentID.**
- **To achieve 3NF, we can create a "Students" table (StudentID, StudentName) and a "Courses" table (CourseID, CourseInstructor).**
- **Now, both tables are in 3NF, and transitive dependencies are eliminated.**

Boyce-Codd Normal Form (BCNF):

- **A table is in BCNF if it is in 3NF and for every non-trivial functional dependency, the left-hand side is a superkey.**

Example: Consider a table "Book_Authors" with columns (ISBN, AuthorID, AuthorName).

- **In 3NF, the table is decomposed, but there's a non-trivial dependency: AuthorName depends on AuthorID.**
- **To achieve BCNF, we can create a "Authors" table (AuthorID, AuthorName) and a "Books" table (ISBN, AuthorID).**
- **Now, both tables are in BCNF, and all non-trivial dependencies are satisfied.**

Q.5] Elaborate the significance of codd's rule. Explain 12 rules proposed by codd's.
ANS: Edgar F. Codd's rules, often referred to as Codd's Twelve Commandments, are a set of principles that define the requirements for a database management system (DBMS) to be considered a relational database. These rules play a crucial role in ensuring the integrity, efficiency, and consistency of relational databases. Let's dive into the significance of Codd's rules and briefly explore each of the 12 rules:

1. **Information Rule:**
   - ***Significance:*** **Ensures that all data in the database is represented in a consistent and structured manner, avoiding redundancy and confusion.**
2. **Guaranteed Access Rule:**
   - ***Significance:*** **Ensures that each piece of data is accessible through a well-defined method, promoting clarity and reliability in data retrieval.**
3. **Systematic Treatment of Null Values:**
   - ***Significance:*** **Allows for the representation of missing or undefined information, preventing ambiguity and supporting a more flexible data model.**
4. **Dynamic Online Catalog Based on The Relational Model:**
   - ***Significance:*** **Ensures that the database schema, including metadata, is stored in a relational format, promoting a unified and organized approach to database management.**
5. **Comprehensive Data Sub-Language Rule:**
   - ***Significance:*** **Requires support for a comprehensive language (like SQL) to express all logical capabilities, enabling powerful and standardized data manipulation.**
6. **View Updating Rule:**
   - ***Significance:*** **Ensures that all views that are theoretically updatable can be updated by the system, maintaining consistency between different representations of the data.**
7. **High-Level Insert, Update, and Delete Rule:**
   - ***Significance:*** **Calls for support for high-level operations, simplifying data manipulation and reducing the need for low-level, error-prone interactions.**
8. **Physical Data Independence:**
   - ***Significance:*** **Allows changes in the physical storage or structure of the database without affecting the application, promoting adaptability and ease of maintenance.**
9. **Logical Data Independence:**
   - ***Significance:*** **Permits changes to the logical structure of the database without impacting applications, enhancing flexibility in database design.**
10. **Integrity Independence:**
    - ***Significance:*** **Separates the definition and management of integrity constraints from application programs, ensuring data integrity without compromising system efficiency.**
11. **Distribution Independence:**
    - ***Significance:*** **Enables the reorganization and distribution of the database without affecting applications, supporting scalability and distributed computing.**

12. Non-Subversion Rule:

- ***Significance:*** **Prevents the use of low-level languages to bypass integrity rules, ensuring that the database's integrity constraints are maintained.**

Q.6] Explain entity and referential integrity constraints used in SQL
ANS:
1.Entity Integrity Constraint:

- ***Definition:*** **Entity integrity ensures that each row (or record) in a table has a unique and non-null primary key value.**
- ***Usage:*** **In a table, the primary key is a column or a combination of columns that uniquely identifies each record. The entity integrity constraint ensures that the primary key value for each record is unique and not null.**
- ***Example:*** **Consider a "Students" table with a primary key column "StudentID." The entity integrity constraint ensures that every student has a unique and non-null StudentID.**

```
CREATE TABLE Students (
   StudentID INT PRIMARY KEY,
   FirstName VARCHAR(50),
   LastName VARCHAR(50)
);
```

2.Referential Integrity Constraint:

- ***Definition:*** **Referential integrity ensures the consistency between tables by enforcing relationships between them. It typically involves a foreign key that references a primary key in another table.**
- ***Usage:*** **In a relationship between two tables, the referential integrity constraint ensures that values in the foreign key column of one table correspond to values in the primary key column of another table.**
- ***Example:*** **Consider a "Orders" table with a foreign key "CustomerID" that references the "Customers" table's primary key "CustomerID."**

```
CREATE TABLE Customers (
   CustomerID INT PRIMARY KEY,
   CustomerName VARCHAR(50)
);

CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,
   CustomerID INT,
   OrderDate DATE,
   FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

In this example, the referential integrity constraint ensures that every "CustomerID" in the "Orders" table corresponds to an existing "CustomerID" in the "Customers" table.

Q.7] Define 3NF. Explain with example, how to bring the relation in 3NF?
ANS:
Third Normal Form (3NF):
Third Normal Form is a level of database normalization that builds on the concepts of 1st Normal Form (1NF) and 2nd Normal Form (2NF). A relation is in 3NF if it is in 2NF and there are no transitive dependencies. Transitive dependencies occur when a non-prime attribute is functionally dependent on another non-prime attribute, rather than on the primary key.
Example: Let's consider a table called "Employee_Departments" with the following columns: (EmployeeID, EmployeeName, DepartmentID, DepartmentName, Location).

- **1st Normal Form (1NF): Ensure that there are no repeating groups, and each cell contains atomic values. This is typically achieved by splitting the data into separate tables.**

Employee_Departments:

EmployeeID	EmployeeName	DepartmentID	DepartmentName	Location
1	John	101	HR	NY
2	Jane	102	IT	SF

2nd Normal Form (2NF): Ensure that all non-prime attributes are fully functionally dependent on the primary key. In our case, the primary key is (EmployeeID, DepartmentID).
Employees:

EmployeeID	EmployeeName
1	John
2	Jane

Departments:

DepartmentID	DepartmentName	Location
101	HR	NY
102	IT	SF

Employee_Departments:

EmployeeID	DepartmentID
1	101
2	102

3rd Normal Form (3NF): Eliminate transitive dependencies. In our case, the transitive dependency is Location on DepartmentName.
Employees:

EmployeeID	EmployeeName
1	John
2	Jane

Departments:

DepartmentID	DepartmentName
101	HR
102	IT

Department_Locations:

DepartmentID	Location
101	NY
102	SF

Employee_Departments:

EmployeeID	DepartmentID
1	101
2	102

Q.8] Explain following Codd's rules with suitable examples : i) Physical Data Independence ii) Integrity Independence iii) Systematic Treatment of NULL Values

ANS:

i) Physical Data Independence:

- **Rule: Changes in the physical storage or structure of the database should not affect the application.**
- **Example:**
  - **Suppose you have a table named "Employees" with columns (EmployeeID, FirstName, LastName, Salary). The physical storage might initially be a single disk, and the data is stored in a specific way.**
  - **Achieving physical data independence means that you can later move the data to a different disk, reorganize the storage, or change the underlying storage technology without impacting the application or the way users interact with the data.**

ii) Integrity Independence:

- **Rule: Integrity constraints must be definable and manageable separately from application programs.**
- **Example:**
  - **Consider a scenario where you have a constraint that ensures that the "Salary" column in the "Employees" table should not be negative.**
  - **Achieving integrity independence means that you can define and manage this constraint separately from the application code. You can set up the constraint within the database itself, and it will be enforced regardless of how different applications interact with the database.**

iii) Systematic Treatment of NULL Values:

- **Rule: The DBMS must allow each field to remain null, representing missing or undefined information.**
- **Example:**
  - **In a "Students" table, you may have a column named "MiddleName." However, not all students may have a middle name.**
  - **Systematic treatment of NULL values allows you to represent the absence of a middle name for certain students. Instead of inserting a placeholder value like "N/A" or an empty string, you can use NULL to indicate that the information is missing or undefined.**

Q.9] Explain 3NF and BCNF and give its example. Also enlist their differences.
ANS:

1. **3NF (Third Normal Form):**
    - **In 3NF, a relation/table is in second normal form (2NF) and further eliminates transitive dependencies.**
    - **It means that all the columns in a table are dependent only on the primary key and not on any other column.**
2. **BCNF (Boyce-Codd Normal Form):**
    - **BCNF is a higher level of normalization where every non-trivial functional dependency is a superkey. In simpler terms, it ensures that there are no non-trivial functional dependencies of attributes on anything other than a superkey.**

Example: Let's consider a table for a library with columns:

- **BookID (Primary key)**
- **Title**
- **Author**
- **Genre**

In this table, suppose the functional dependencies are:

- **BookID -> Title, Author**
- **Title -> Genre**

This table is in 2NF because each non-prime attribute depends on the whole primary key. However, it's not in 3NF or BCNF due to transitive dependencies.
Bringing it to 3NF: To achieve 3NF, we need to break the transitive dependency. We split the table into two:
Table 1:

- **BookID (Primary key)**
- **Title**
- **Author**

Table 2:

- **Title (Primary key)**
- **Genre**

This separation resolves the transitive dependency and brings the table to 3NF.
BCNF: For BCNF, the original example is almost there. However, since Title -> Genre and Title is not a superkey, to achieve BCNF, we'd need to split the table further:
Table 1:

- **BookID (Primary key)**
- **Title**
- **Author**

Table 2:

- **Title (Primary key)**
- **Genre**

This separation satisfies BCNF as all non-trivial functional dependencies are based on superkeys (BookID in the first table and Title in the second).
Differences between 3NF and BCNF:

- **3NF deals with eliminating transitive dependencies while ensuring data is stored in a way that reduces redundancy.**

- **BCNF goes a step further, ensuring that every non-trivial functional dependency is based on a superkey, optimizing the table for minimal redundancy and anomalies.**

Both are about improving data integrity and reducing anomalies in a database schema, but BCNF is a stricter form of normalization.

Q.10] What are the desirable properties of decomposition? Explain it with example.

ANS: here are the desirable properties of decomposition explained in simple points along with an example:

1. **Reduced Complexity: Decomposition breaks down a complex system into simpler, more manageable parts. Each part focuses on a specific aspect, making it easier to understand and work with.**
2. **Modularity: Decomposed parts should be relatively independent modules that can be developed, tested, and modified separately without affecting other parts of the system. This modularity enhances flexibility and maintainability.**
3. **Abstraction: Each part should hide unnecessary details, providing a higher-level view that facilitates understanding and reasoning about the system. Abstraction allows developers to focus on essential aspects while ignoring irrelevant details.**
4. **Reusability: Well-decomposed parts can be reused in different contexts or integrated into other systems. Reusability saves time and effort by leveraging existing components rather than reinventing them.**
5. **Scalability: Decomposition enables scaling a system by adding or modifying parts without disrupting the entire system. This scalability is crucial for accommodating growth or adapting to changing requirements.**

Example:

Consider a software application for managing a bookstore. Instead of building a monolithic system to handle all tasks, decomposition can break it down into smaller, more manageable components:

1. **Inventory Management Module: This module tracks the bookstore's inventory, including books in stock, quantities, and locations. It handles tasks such as adding new books, updating quantities, and generating reports.**
2. **Sales and Customer Management Module: This module deals with customer interactions, including sales transactions, customer information, and order processing. It manages tasks such as processing orders, generating invoices, and maintaining customer records.**
3. **Payment Processing Module: This module handles payment processing, integrating with payment gateways to facilitate secure transactions. It manages tasks such as authorizing payments, capturing funds, and handling refunds.**
4. **User Interface Module: This module provides the user interface for interacting with the bookstore application. It includes components for browsing books, searching for titles, placing orders, and managing user accounts.**

Q.11] Explain partial and transitive dependencies with example.

ANS: let's break it down into simple points:

1. **Dependency in Databases: In databases, dependencies refer to relationships between attributes (columns) in a table.**
2. **Functional Dependency: It's a concept where one attribute determines the value of another attribute. For example, in a table of employees, the attribute "Employee ID" determines the "Employee Name" uniquely. This is denoted as Employee ID -> Employee Name.**
3. **Partial Dependency: It occurs when an attribute is functionally dependent on only a part of a composite primary key (multi-attribute primary key). In simpler terms, if an attribute depends on only one part of a composite key, it's a partial dependency.**
4. **Example of Partial Dependency: Let's say we have a table called "Order Details" with attributes (columns) like Order ID, Product ID, Product Name, and Quantity. If Product Name depends only on Product ID and not on Order ID, it's a partial dependency. Because Product Name should be uniquely determined by Product ID, not by both Order ID and Product ID.**
5. **Transitive Dependency: This occurs when an attribute depends on another attribute, which in turn depends on a third attribute. In other words, if A -> B and B -> C, then A -> C. This is known as transitive dependency.**
6. **Example of Transitive Dependency: Consider a table called "Student Grades" with attributes Student ID, Course ID, and Professor Name. Here, if Professor Name depends on Course ID, and Course ID depends on Student ID, then Professor Name depends transitively on Student ID. Because Professor Name is indirectly determined by Student ID.**
7. **Illustration of Transitive Dependency:**
    - **Student ID -> Course ID (Each student is enrolled in specific courses)**
    - **Course ID -> Professor Name (Each course is taught by a specific professor)**
    - **Therefore, Student ID -> Professor Name (Professor Name is determined by the courses taken by the student)**
8. **Normalization to Resolve Dependencies: To deal with these dependencies and ensure data integrity, databases are often normalized. Normalization involves organizing the attributes of a table into separate tables to reduce redundancy and dependency issues.**
9. **Conclusion: Partial and transitive dependencies are essential concepts in database design. Understanding them helps in organizing data efficiently and maintaining data integrity. Normalization techniques are applied to address these dependencies and optimize database structures.**

Q.12] Explain why database normalization is required for good relational database design? Explain with example requirements of different normal forms like 1NF, 2 NFand 3NF.

ANS: Sure, I'll break it down into simple points:

Why Database Normalization is Required:

1. **Minimize Redundancy: Normalization helps in reducing redundancy by organizing data efficiently. Redundancy can lead to inconsistencies and increase storage requirements.**
2. **Data Integrity: By eliminating redundant data, normalization ensures data integrity, meaning each piece of data is accurate and consistent throughout the database.**
3. **Flexibility: Normalized databases are more flexible and adaptable to changes in requirements or updates. It's easier to add, delete, or modify data without affecting the entire structure.**
4. **Efficient Queries: Properly normalized databases tend to perform better for query operations, as they require less processing to retrieve and manipulate data.**

Requirements of Different Normal Forms:

1. **First Normal Form (1NF):**
   - **Eliminate duplicate columns from the same table.**
   - **Create separate tables for related data and identify each row with a unique identifier or primary key.**
   - **Example: Suppose you have a table for storing customer orders. Instead of having columns like "Order ID," "Customer Name," and "Customer Address" in the same table, create a separate table for customers with unique IDs and link it to the orders table using the customer ID.**
2. **Second Normal Form (2NF):**
   - **Meet the requirements of 1NF.**
   - **Remove partial dependencies by ensuring that non-key attributes are fully functionally dependent on the primary key.**
   - **Example: Continuing with the customer orders example, if the orders table includes columns like "Product ID" and "Product Name," where "Product Name" depends only on "Product ID" and not on the entire primary key, you would move the product-related information to a separate table linked by the product ID.**
3. **Third Normal Form (3NF):**
   - **Meet the requirements of 2NF.**
   - **Eliminate transitive dependencies by ensuring that non-key attributes are not dependent on other non-key attributes.**
   - **Example: In the customer orders scenario, if there's a column like "Supplier Address" dependent on "Product ID" rather than directly on the primary key, you would create a separate table for products, including supplier information, and link it to the orders table using the product ID.**

Q.13] What is anomaly in relational model. Explain how normalization can be used to reduce the anomalies.

ANS: Anomalies in the relational model refer to inconsistencies or errors that can occur when storing, updating, or deleting data in a database table. These anomalies can be categorized into three types: insertion anomalies, update anomalies, and deletion anomalies.

Here's a simple and easy-to-understand explanation of each type of anomaly and how normalization can help reduce them:

1. **Insertion Anomalies:**
   - **Occur when you can't add a new record to the database without also adding additional unrelated data.**
   - **Example: In a table storing information about students and their courses, if a student hasn't taken any courses yet, you can't add them to the database because there's no course information associated with them.**
   - **Normalization Solution: Break down the original table into smaller, related tables. In this case, you could have separate tables for students and courses. Then, you can add students even if they haven't taken any courses yet.**
2. **Update Anomalies:**
   - **Arise when updating one piece of data leads to inconsistencies or errors in other related data.**
   - **Example: If a student changes their address, you have to update multiple records for each course they're enrolled in, leading to the risk of inconsistencies if you forget to update all instances.**
   - **Normalization Solution: By breaking down the original table into smaller, related tables and properly linking them with keys, you minimize the chances of inconsistencies. In this case, you would have separate tables for students and addresses, linked by a unique identifier like a student ID.**
3. **Deletion Anomalies:**
   - **Happen when deleting a piece of data unintentionally removes other related data that you want to keep.**
   - **Example: If you delete a student's record from a table containing both student and course information, you might unintentionally lose information about the course itself if the student is the only one enrolled in it.**
   - **Normalization Solution: Again, by breaking down the original table into smaller, related tables, you can prevent deletion anomalies. Each piece of data is stored in its own table, reducing the risk of inadvertently losing unrelated data. In this case, having separate tables for students and courses would help avoid deletion anomalies.**

Q.14] Explain 2NF and 3NF and BCNF with example.
ANS: Sure, let's break it down into simple points:
2NF (Second Normal Form):

1. **Eliminate Partial Dependencies: Every non-prime attribute (attributes not part of any candidate key) must be fully functionally dependent on the primary key.**
2. **Example: Let's say we have a table called "Employee_Details" with columns: Employee_ID (PK), Employee_Name, Department_ID (FK), and Department_Name.**
3. **Here, Employee_Name is fully dependent on Employee_ID (PK), but Department_Name depends only on Department_ID, not on Employee_ID (PK).**
4. **To make it 2NF, we split the table into two: "Employee" (Employee_ID, Employee_Name, Department_ID) and "Department" (Department_ID, Department_Name).**

3NF (Third Normal Form):

1. **Eliminate Transitive Dependencies: No non-prime attribute should depend on another non-prime attribute.**
2. **Example: Using the "Employee" and "Department" tables from 2NF example.**
3. **If we add a new column in "Department" called "Manager_Name", it might create a transitive dependency because "Manager_Name" depends on "Department_ID", not directly on the Employee.**
4. **To achieve 3NF, we create a new table "Manager" with columns: "Department_ID" and "Manager_Name".**

BCNF (Boyce-Codd Normal Form):

1. **Ensure All Dependencies are Superkey Dependencies: Every determinant (a column that determines the value of another column) must be a candidate key.**
2. **Example: Continuing with the "Employee" and "Department" tables from 3NF example.**
3. **If we add a new column in "Employee" called "Salary", where "Salary" depends on "Employee_ID" and not just "Department_ID".**
4. **To make it BCNF, we would need to further normalize the tables, possibly separating out salary information into its own table, ensuring all dependencies are based on superkeys.**

Q.15] What are relational integrity constraints. Explain with example Domain constraints, Referential-Integrity and enterprise constraints.
ANS: Sure, let's break down relational integrity constraints into three main types: domain constraints, referential integrity, and enterprise constraints, and explain each with simple examples:

1. **Domain Constraints:**
    - **Definition: These constraints ensure that only valid data is entered into a database by restricting the values that can be stored in a column.**
    - **Example: Suppose you have a database table for storing ages of people. A domain constraint might specify that the age column can only contain integer values between 0 and 120, excluding negative values and outliers.**

2. **Referential Integrity:**
    - **Definition: This constraint maintains the consistency and integrity of relationships between tables by ensuring that foreign key values in one table match primary key values in another table.**
    - **Example: Let's say you have two tables: "Students" and "Courses." The "Students" table has a foreign key "course_id" referencing the primary key "course_id" in the "Courses" table. Referential integrity ensures that if a course is deleted from the "Courses" table, all corresponding records in the "Students" table referencing that course are also deleted or appropriately handled.**

3. **Enterprise Constraints:**
    - **Definition: These constraints involve rules or conditions that span multiple tables or the entire database and are typically specific to the organization's business rules or policies.**
    - **Example: Consider a hospital database where scheduling surgeries requires availability of both surgeons and operating rooms. An enterprise constraint might enforce that a surgery cannot be scheduled unless both a surgeon and an operating room are available at the specified time.**