UNIT- 2

CONSTRAINTS & NORMALIZATION

2.1 BASIC CONCEPTS

1. **Entity:** An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. Or Any real-world object can be represented as an entity about which data can be stored in a database For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.
2. **An entity set** is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.
3. **Attributes:** Entities are represented by means of their properties, called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes. attributes gives identity to entity.
4. **Domain:** or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc. Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).
5. **Tuple**: A single row of a table, which contains a single record for that relation is called a tuple.

2.2 CODD's RULES

Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, suggested twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database. These rules can be applied on any database system that manages stored data using only its relational capabilities.

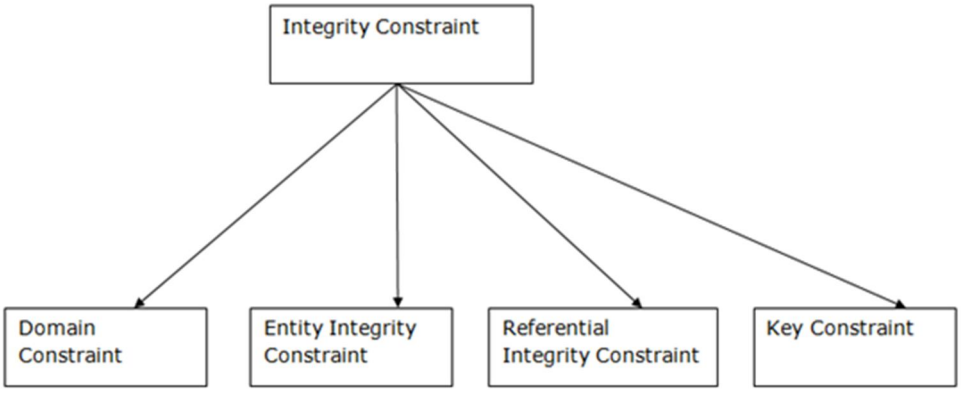
- **Rule 1: Information Rule:** The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.
- **Rule 2: Guaranteed Access Rule:** Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.
- **Rule 3: Systematic Treatment of NULL Values:** The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following – data is missing, data is not known, or data is not applicable.
- **Rule 4: Active Online Catalog:** The structure description of the entire database must be stored in an online catalog, known as data dictionary, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.
- **Rule 5: Comprehensive Data Sub-Language Rule:** A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and

transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

- **Rule 6: View Updating Rule:** All the views of a database, which can theoretically be updated, must also be updatable by the system.
- **Rule 7: High-Level Insert, Update, and Delete Rule:** A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.
- **Rule 8: Physical Data Independence:** The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.
- **Rule 9: Logical Data Independence:** The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.
- **Rule 10: Integrity Independence:** A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.
- **Rule 11: Distribution Independence:** The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.
- **Rule 12: Non-Subversion Rule:** If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

2.3 INTEGRITY CONSTRAINTS

Integrity constraints are a set of rules. It is used to maintain the quality of information. It ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected. Thus, is used to guard against accidental damage to the database. Various types of Integrity Constraint are



1. Domain constraints: Domain constraints can be defined as the definition of a valid set of values for an attribute. The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.
Attributes have specific values in real-world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

2. Entity integrity constraints: The entity integrity constraint states that primary key value can't be null. This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows. A table can contain a null value other than the primary key field.

Example:

EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

3. Key constraints: Keys are the entity set that is used to identify an entity within its entity set uniquely. An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

Example:

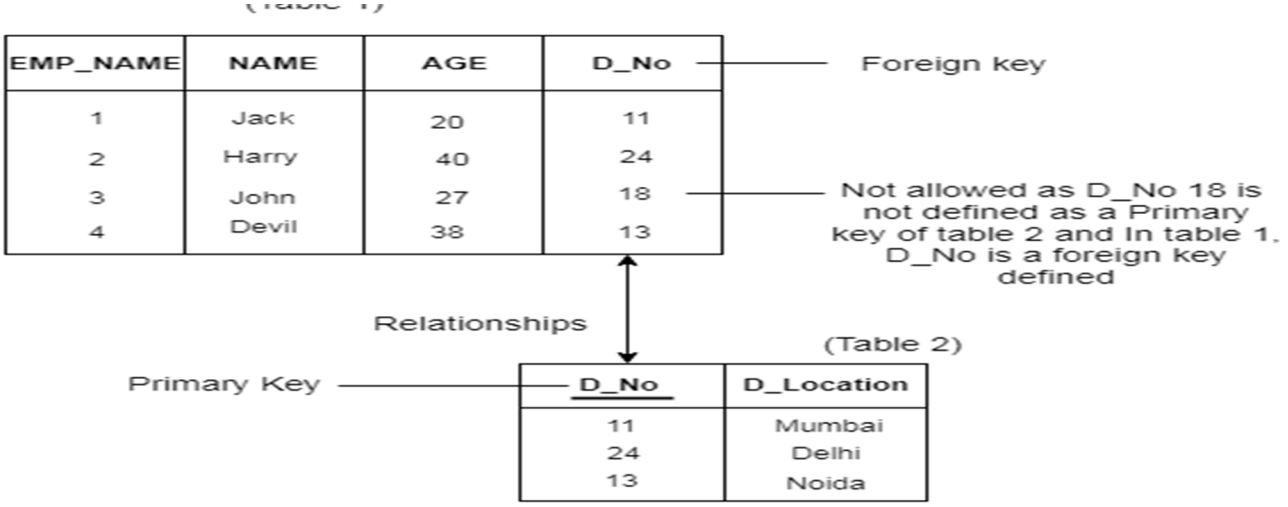
ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

4. Referential Integrity Constraints: A referential integrity constraint is specified between two tables. In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2. So we can say Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other

relation. Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

Example:



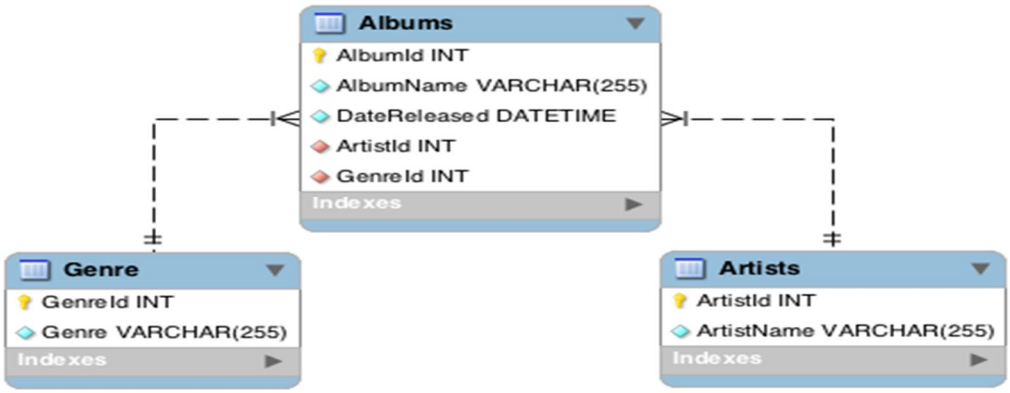
2.4 Enterprise Constraints: Enterprise constraints – sometimes referred to as semantic constraints – are additional rules specified by users or database administrators and can be based on multiple tables. Here are some examples.

- A class can have a maximum of 30 students.
- A teacher can teach a maximum of four classes per semester.
- An employee cannot take part in more than five projects.
- The salary of an employee cannot exceed the salary of the employee's manager.

2.5 DATABASE SCHEMA

A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.



Above is a simple example of a schema diagram. It shows three tables, along with their data types, relationships between the tables, as well as their primary keys and foreign keys.

2.6 FUNCTIONAL DEPENDENCY

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

$$X \rightarrow Y$$

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example: Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address. Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it. Functional dependency can be written as:

$$Emp\_Id \rightarrow Emp\_Name$$

We can say that Emp_Name is functionally dependent on Emp_Id.

2.6.1 Types of Functional dependency

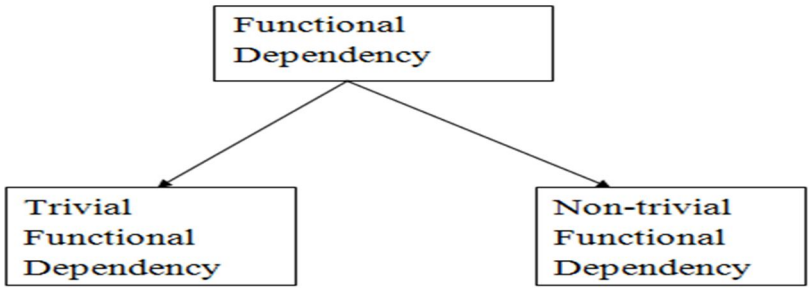


Figure: Types of Functional Dependency

1. Trivial functional dependency: A Dependency is said to be Trivial if in any dependency A -> B, B is a entity which is subset of entity set A. The dependencies like: $A \rightarrow A$, $B \rightarrow B$ are also trivial

Example: Consider a table with two columns Employee_Id and Employee_Name.

$$\{Employee\_id, Employee\_Name\} \rightarrow Employee\_Id$$

It is a trivial functional dependency as Employee_Id is a subset of {Employee_Id, Employee _Name}. Also following are trivial dependencies too.

$$Employee\_Id \rightarrow Employee\_Id \text{ and } Employee\_Name \rightarrow Employee\_Name$$

2. Non-trivial functional dependency: A Dependency is said to be Non Trivial if in any dependency A -> B, B is a entity which is not subset of entity set A. When A intersection B is NULL, then $A \rightarrow B$ is called as complete non-trivial.

Example: **1.** ID → Name, 2. Name → DOB

2.7 ANOMALIES: If a database design is not perfect, it may contain anomalies that is impurity, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

- **Update anomalies** – If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.
- **Deletion anomalies** – We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.
- **Insert anomalies** – We tried to insert data in a record that does not exist at all.

2.8 RELATIONAL DECOMPOSITION:

Decomposition is the process of splitting the tables in two or more tables. When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required. In a database, it breaks the table into multiple tables. If the relation has no proper decomposition, then it may lead to problems like loss of information. Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

2.8.1 Types of Decomposition

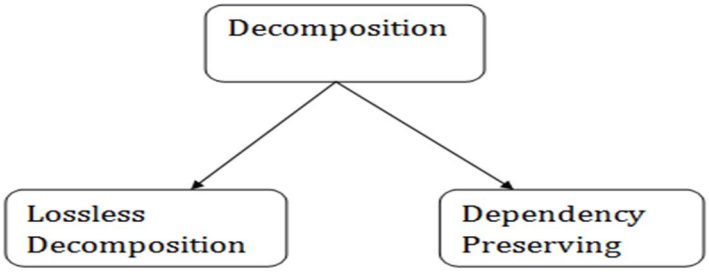


Figure: Types of Decomposition

1. Lossless Decomposition: If the information is not lost from the relation that is decomposed, then the decomposition will be lossless. The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed. The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

Example: **Table: Employee_Department**

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

The above relation is decomposed into two relations Employee and Department

Table: Employee

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY
22	Denim	28	Mumbai
33	Alina	25	Delhi
46	Stephan	30	Bangalore
52	Katherine	36	Mumbai
60	Jack	40	Noida

Table: Department

DEPT_ID	EMP_ID	DEPT_NAME
827	22	Sales
438	33	Marketing
869	46	Finance
575	52	Production
678	60	Testing

Now, when these two relations are joined on the common column "EMP_ID", then the resultant relation will look like: Employee X Department (Joint of Both Tables)

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

Hence, the decomposition is Lossless join decomposition.

2. Dependency Preserving: It is an important constraint of the database. In the dependency preservation, at least one decomposed table must satisfy every dependency. If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2. For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

2.9 MULTIVALUED DEPENDENCY:

Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute. A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

Example: Suppose there is a bike manufacturer company which produces two colors (white and black) of each model every year.

BIKE_MODEL	MANUF_YEAR	COLOR
M2001	2008	Black
M3001	2013	White
M3001	2013	Black
M4006	2017	White
M4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other. In this case, these two columns can be called as multivalued dependent on BIKE_MODEL. The representation of these dependencies is shown below:

1. BIKE_MODEL → → MANUF_YEAR
2. BIKE_MODEL → → COLOR

So to remove this dependency the above table need to be decomposed in to two tables as bellow.

BIKE_MODEL	MANUF_YEAR
M2001	2008
M3001	2013
M3001	2013
M4006	2017
M4006	2017

BIKE_MODEL	COLOR
M2001	Black
M3001	White
M3001	Black
M4006	White
M4006	Black

This can be read as "BIKE_MODEL multi determined MANUF_YEAR" and "BIKE_MODEL multi determined COLOR".

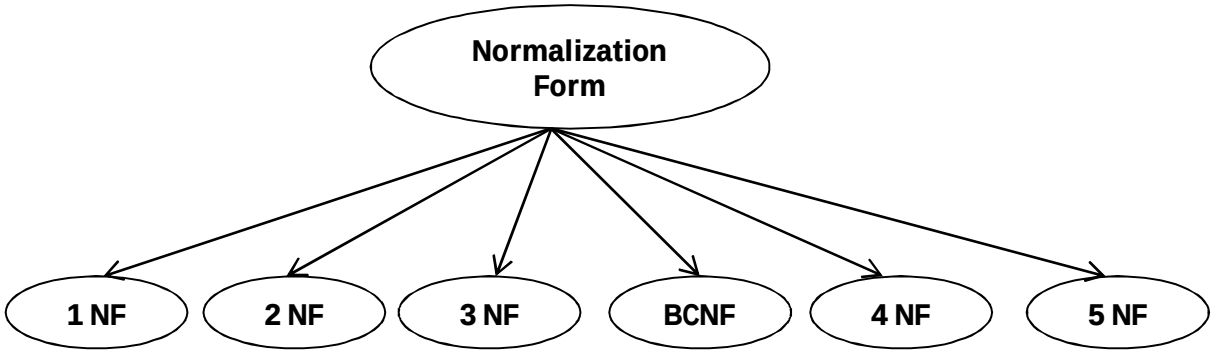
2.10 JOIN DEPENDENCY

Join decomposition is a further generalization of Multivalued dependencies. If the join of R1 and R2 over C is equal to relation R, then we can say that a join dependency (JD) exists. Where R1 and R2 are the decompositions R1(A, B, C) and R2(C, D) of a given relations R (A, B, C, D). Alternatively, R1 and R2 are a lossless decomposition of R. A JD ⋈ {R1, R2,...,Rn} is said to hold over a relation R if R1, R2,....., Rn is a lossless-join decomposition. The *(A, B, C, D), (C, D) will be a JD of R if the join of join's attribute is equal to the relation R. Here, *(R1, R2, R3) is used to indicate that relation R1, R2, R3 and so on are a JD of R.

2.11 NORMALIZATION

Normalization is the process of organizing the data in the database. It is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization divides the larger table into the smaller table and links them using relationship. The normal form is used to reduce redundancy from the database table. So we can say It is a method to remove all these anomalies and bring the database to a consistent state.

Types of Normal Forms: There are the six types of normal forms



2. First Normal Form (1NF): A relation will be 1NF if it contains an atomic value. It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute. First normal form disallows the multi-valued attribute, composite attribute, and their combinations. First normal form also disallows the empty field for any row or column
Example: Relation Employee is not in 1NF because of multi-valued attribute EMP_PHONE.

Table: Not in 1NF Before Decomposition

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	
12	Sam	7390372389, 8589830302	Punjab

Table: 1NF after Decomposition

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

The decomposition of the Employee table into 1NF has been shown Above:

3. Second Normal Form (2NF): For the 2NF, relational must be in 1NF. In the second normal form, all non-key attributes are fully functional dependent on the primary key**.**
Example: Let's assume a Student Table can store the data of student and his project as follow.

Table: In 1NF But not in 2NF

Student_ID	Project_ID	Student_Name	Project_Name
25	P1	John	ABC
26	P2	Gems	XYZ
47	P3	Harry	PQR
83	P4	Sam	LMN
25	P5	John	STU

In the given table, non-prime attribute Project_Name is dependent on Project_ID as well Student_ID which is a proper subset of a candidate key, i.e Project_Name can be identified by Student_ID as well Project_ID. That's why it violates the rule for 2NF. To convert the given table into 2NF, we decompose it into two tables:

Table: student_Detail

Student_ID	Student_Name
25	John
26	Gems
47	Harry
83	Sam

Table: Project_Detail

Project_ID	Project_Name
P1	ABC
P2	XYZ
P3	PQR
P4	LMN
P5	STU

4. Third Normal Form (3NF): A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency. 3NF is used to reduce the data duplication. It is also used to achieve the data integrity. If there is no transitive dependency for non-prime (Non Primary key) attributes, then the relation must be in third normal form.
If any attribute A,B, & C from your table hold the functional dependency relation like A -> B, A -> C Then we can conclude that there will be A -> C which is called as transitive dependency, it violates the rule for 3NF

Example **Table: Employee_Detail in 2NF But Not in 3NF**

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

In above table all attributes except EMP_ID are non-prime. Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form. That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

Table: EMPLOYEE in 3NF

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

Table: EMPLOYEE_ZIP in 3NF

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

5. Boyce Codd normal form (BCNF): BCNF is the advance version of 3NF. It is stricter than 3NF. A table is in BCNF if every functional dependency $X \rightarrow Y$, X is the super key of the table. For BCNF, the table should be in 3NF, and for every FD, LHS is super key. Should have single super key. Example: Let's assume there is a company where employees work in more than one department.

Table EMPLOYEE in 3NF but Not in BCNF

EMP_ID	EMP_DEPT	DEPT_MEMBER	EMP_DEPT_NO
264	Designing	394	283
264	Testing	194	300
364	Stores	583	232
364	Developing	283	549

In given table it contains two super keys namely EMP_ID and EMP_DEPT_NO which is violating the rule of BCNF so to convert the given table into BCNF, we decompose it into two tables:

Table: EMP_DEPT

EMP_DEPT	DEPT_MEMBER	EMP_DEPT_NO
Designing	394	283
Testing	194	300
Stores	583	232
Developing	283	549

Table: EMP_DEPT_MAPPING

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

6. Fourth normal form (4NF): A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency. For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

Table: STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY. In the STUDENT relation, a student with STU_ID 21 contains Two courses, Computer and Math and two hobbies Dancing and Singing. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data. So to make the above table into 4NF, we can decompose it into two tables:

Table: STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology

Table: STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket

7. Fifth normal form (5NF): A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless. 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy. 5NF is also known as Project-join normal form (PJ/NF).

Example

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data. Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank. So to make the above table into 5NF, we can decompose it into three relations R1, R2 & R3:

Table: R1.

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

Table: R2.

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

Table: R3.

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

2.11.1 Purpose of Normalization: Normalization is the aim of well design Relational Database Management System (RDBMS). It is step by step set of rules by which data is put in its simplest forms. We normalize the relational database management system because of the following reasons:

- Minimize data redundancy i.e. no unnecessarily duplication of data.
- To make database structure flexible i.e. it should be possible to add new data values and rows without reorganizing the database structure.
- Data should be consistent throughout the database i.e. it should not suffer from following anomalies.

- Insert Anomaly - Due to lack of data i.e., all the data available for insertion such that null values in keys should be avoided. This kind of anomaly can seriously damage a database
- Update Anomaly - It is due to data redundancy i.e. multiple occurrences of same values in a column. This can lead to inefficiency.
- Deletion Anomaly - It leads to loss of data for rows that are not stored else where. It could result in loss of vital data.
- To Make queries required by the user easy to handle.

The resulting relations (tables) obtained on normalization should possess the properties such as each row must be identified by a unique key, no repeating groups, homogenous columns, each column is assigned a unique name etc.