Unit 2 Database Design

Database Design

- Functional Dependencies
- Inference Rules
- Functional Dependency Closure
- Minimal Cover
- Decomposition Properties
- Need of Normalization
- Normal Forms: 1NF, 2NF, 3NF and BCNF, Multi-valued Dependency, 4NF

- **Database design is the process of creating a structured plan for the database that outlines how data will be stored, organized, and accessed. It involves designing the database schema, which defines the structure and relationships between data elements and determines the most appropriate data types and storage mechanisms for the data.**

- **There are several key considerations in database design, including:**
- **Data requirements:** Identify the types of data that need to be stored and the relationships between different data elements.
-
- **Data normalization:** Divide the data into smaller, related tables to eliminate redundancy and improve data integrity.
-
- **Indexes:** Determine which data elements need to be indexed to improve query performance.
-
- **Data types:** Choose appropriate data types for each data element to ensure efficient storage and retrieval of data.

Functional Dependencies

- **Functional Dependency (FD)** is a **constraint that determines the relation of one attribute to another attribute** in a Database Management System (DBMS).
- **The attributes of a table is said to be dependent on each other when an attribute of a table uniquely identifies another attribute of the same table.**
- For example: Suppose we have a student table with attributes: **Stu_Id, Stu_Name, Stu_Age**. Here **Stu_Id attribute uniquely identifies the Stu_Name attribute of student table** because if we know the student id we can tell the student name associated with it. **This is known as functional dependency and can be written as Stu_Id->Stu_Name or in words we can say Stu_Name is functionally dependent on Stu_Id.**
- **Formally**:
  If column A of a table uniquely identifies the column B of same table then it can represented as A->B (Attribute B is functionally dependent on attribute A)
- **A->B The left side of FD is known as a determinant, the right side of the production is known as a dependent.**

Functional Dependencies

- **Types of Functional Dependencies in DBMS**
- There are mainly four types of Functional Dependency in DBMS. Following are the types of Functional Dependencies in DBMS:
- **Multivalued Dependency**
- **Trivial Functional Dependency**
- **Non-Trivial Functional Dependency**
- **Transitive Dependency**

Functional Dependencies

- **Multivalued Dependency:**
- Multivalued dependency occurs in the **situation where there are multiple independent multivalued attributes in a single table**.
- A multivalued dependency **is a complete constraint between two sets of attributes in a relation.** It requires that certain tuples be present in a relation.
  - In this example, **maf_year and color are independent of each other but dependent on car_model.** In this example, **these two columns are said to be multivalue dependent on car_model.**
  - This dependence can be represented like this:
  - **car_model -> maf_year**
  - **car_model-> colour**

Car_model	Maf_year	Color
H001	2017	Metallic
H001	2017	Green
H005	2018	Metallic
H005	2018	Blue
H010	2015	Metallic
H033	2012	Gray

Functional Dependencies

Emp_id	Emp_name
AS555	Harry
AS811	George

- **Trivial Functional Dependency in DBMS**
- The Trivial dependency is a set of attributes which are called a trivial if the set of attributes are included in that attribute.
- So, X -> Y is a trivial functional dependency if Y is a subset of X.
- Consider this table of with two columns Emp_id and Emp_name.
- {Emp_id, Emp_name} -> Emp_id is a trivial functional dependency as Emp_id is a subset of {Emp_id,Emp_name}.

- **Non Trivial Functional Dependency in DBMS**
- Functional dependency which also known as a nontrivial dependency occurs **when A->B holds true where B is not a subset of A.**
- In a relationship, **if attribute B is not a subset of attribute A, then it is considered as a non-trivial dependency.**
- (Company} -> {CEO} (if we know the Company, we knows the CEO name)
- But CEO is not a subset of Company, and hence it's non-trivial functional dependency.

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Apple	Tim Cook	57

Functional Dependencies

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Apple	Tim Cook	57

- **Transitive Dependency in DBMS**
- **A Transitive Dependency is a type of functional d pens when "t" is indirectly formed by two functional dependencies.**
- **{Company} -> {CEO} (if we know the company, we know its CEO's name)**
- **{CEO } -> {Age} If we know the CEO, we know the Age**
- **Therefore according to the rule of rule of transitive dependency:**
- **{ Company} -> {Age} should hold, that makes sense because if we know the company name, we can know his age.**
- Note: You need to remember that transitive dependency can only occur in a relation of three or more attributes.

Functional Dependencies

- Some key terms for Functional Dependency in Database:

Key Terms	Description
Axiom	**Axioms is a set of inference rules** used to infer all the functional dependencies on a relational database.
Decomposition	**It is a rule that suggests if you have a table that appears to contain two entities which are determined by the same primary key then you should consider breaking them up into two different tables.**
Dependent	It is **displayed on the right side** of the functional dependency diagram.
Determinant	It is **displayed on the left side** of the functional dependency Diagram.
Union	It suggests that if two tables are separate, and the PK is the same, you should consider putting them together

Functional Dependencies

- **Inference Rules of Functional Dependencies**
- Below are the Three most important rules for Functional Dependency in Database:
- **Reflexive rule –. If X is a set of attributes and Y is_subset_of X, then X holds a value of Y. For example {STU_ID, NAME} →NAME is valid reflexive relation.**
- **Augmentation rule: When x -> y holds, and c is attribute set, then ac -> bc also holds. That is adding attributes which do not change the basic dependencies. For example {STU_ID, NAME} →{ DEPT_BUILDING} is valid then {STU_ID, NAME,DEPT_NAME} →{ DEPT_BUILDING,DEPT_NAME} is also valid.**
- **Transitivity rule: This rule is very much similar to the transitive rule in algebra if x -> y holds and y -> z holds, then x -> z also holds.** X -> y is called as functionally that determines y. **For example, if STU_ID→CLASS, CLASS→LECTURE_HALL holds true then according to the axiom of transitivity, STU_ID→LECTURE_HALL will also hold true.**

Functional Dependencies

- **Inference Rules of Functional Dependencies**
- **Union Rule (IR4)**
- This rule is also known as the additive rule. if X determines Y and X determines Z, then X also determines both Y and Z.
- If X $\rightarrow$ Y and X $\rightarrow$ Z then X $\rightarrow$ YZ
- **For example, STU_ID $\rightarrow$ STU_NAME, STU_ID$\rightarrow$COURSE then STU_ID$\rightarrow$ {STU_NAME, COURSE} holds true.**
- Proof:
- 1. X $\rightarrow$ Y (given)
- 2. X $\rightarrow$ Z (given)
- 3. X $\rightarrow$ XY (using IR2 on 1 by augmentation with X)
- 4. XY $\rightarrow$ YZ (using IR2 on 2 by augmentation with Y)
- 5. X $\rightarrow$ YZ (using IR3 on 3 and 4)

Functional Dependencies

- Inference Rules of Functional Dependencies
- Decomposition Rule (IR5)
- This rule is the reverse of the Union rule and is also known as the project rule.
- if X determines Y and Z together, then X determines Y and Z separately
- For example STU_ID→ {STU_NAME, COURSE} then STU_ID → STU_NAME, STU_ID→COURSE holds true.
- If X → YZ then X → Y and X → Z
- Proof:
- 1. X → YZ (given)
- 2. YZ → Y (using IR1 Rule)
- 3. X → Y (using IR3 on 1 and 2)
- Pseudo transitive Rule (IR6)
- In the pseudo transitive rule, if X determines Y, and YZ determines W, then XZ also determines W.
- If X → Y and YZ → W then XZ → W
- Proof:
- 1. X → Y (given)
- 2. WY → Z (given)
- 3. WX → WY (using IR2 on 1 by augmenting with W)
- 4. WX → Z (using IR3 on 3 and 2)

- Simple FD Roll_No : Name One attribute determines another
- Composite FD {First_Name, Last_Name} : Emp_ID Combination determines Emp_ID
- Transitive FD Roll_No : Course, Course : Dept Indirect dependency
- Trivial FD {Roll_No, Name} : Name RHS is part of LHS
- Non-Trivial FD Roll_No : Course RHS is not part of LHS

Functional Dependency Closure

- Attribute Closure of an attribute set is defined as a **set of all attributes that can be functionally determined from it.**
- Closure of an attribute **x is the set of all attributes that are functional dependencies on X with respect to F.** It is denoted by $\underline{X^{\pm}}$ which means what X can determine.
- The **closure of an attribute is represented as** $^{+}$

- **Finding Closure of an attribute set**
- You can follow the steps to find the Closure of an attribute set:

1. **Determine A^{+}, the Closure of A** under functional dependency set F.
2. **A^{+}: = will contain A itself**; **For example**, if we need to find the closure of an attribute X, the closure will incorporate the X itself and the other attributes that the X attribute can determine.
3. **Repeat the process as**
4. **old A^{+}: = A Closure;**
5. **for each FB X → Y in the FD set, do**
6. **if X Closure is a subset of X, then A Closure:= A Closure U Y;**
7. **Repeat until (A^{+}= old A^{+});**

Functional Dependency Closure

- **Step-1 : Add the attributes which are present on Left Hand Side in the original functional dependency.**
- **Step-2 : Now, add the attributes present on the Right Hand Side of the functional dependency.**
- **Step-3 : With the help of attributes present on Right Hand Side, check the other attributes that can be derived from the other given functional dependencies. Repeat this process until all the possible attributes which can be derived are added in the closure.**
- **Example-1 : Consider the table student_details having (Roll_No, Name,Marks, Location) as the attributes and having two functional dependencies.**
- **FD1 : Roll_No -> Name, Marks**
- **FD2 : Name -> Marks, Location**
- **Now, We will calculate the closure of all the attributes present in the relation using the three steps.**

Functional Dependency Closure

- **Example: Calculate closure of Roll_No**
- **Step-1 : Add attributes present on the LHS of the first functional dependency to the closure.**
- **$\{Roll\_no\}^+ = \{Roll\_No\}$**
- **Step-2 : Add attributes present on the RHS of the original functional dependency to the closure.**
- **$\{Roll\_no\}^+ = \{Roll\_No, Marks\}$**
- **Step-3 : Add the other possible attributes which can be derived using attributes present on the RHS of the closure. So Roll_No attribute cannot functionally determine any attribute but Name attribute can determine other attributes such as Marks and Location using 2nd Functional Dependency(Name -> Marks, Location).**
- **Therefore, complete closure of Roll_No will be :**
- **$\{Roll\_no\}^+ = \{Roll\_No, Marks, Name, Location\}$**

Functional Dependency Closure

- Similarly, we can calculate closure for other attributes too i.e “Name”.
- Step-1 : Add attributes present on the LHS of the functional dependency to the closure.
- $\{Name\}^+ = \{Name\}$
- Step-2 : Add the attributes present on the RHS of the functional dependency to the closure.
- $\{Name\}^+ = \{Name, Marks, Location\}$
- Step-3 : Since, we don’t have any functional dependency where “Marks or Location” attribute is functionally determining any other attribute , we cannot add more attributes to the closure. Hence complete closure of Name would be :
- $\{Name\}^+ = \{Name, Marks, Location\}$
- Calculate Closure for Marks and Location attribute:
- We don’t have any Functional dependency where marks and location can functionally determine any attribute. Hence, for those attributes we can only add the attributes themselves in their closures. Therefore,
- $\{Marks\}^+ = \{Marks\}$ and $\{Location\}^+ = \{ Location\}$

- Example:

Let's say we have a relation R(A, B, C, D) with the following functional dependencies:

A -> B, B -> C, and C -> D.

- Now, let's find the closure of the attribute set {A}.
- Start with the initial set: {A}.
- Apply the first dependency A -> B: We can add B to the set, so it becomes {A, B}.
- Apply the second dependency B -> C: We can add C to the set, so it becomes {A, B, C}.
- Apply the third dependency C -> D: We can add D to the set, so it becomes {A, B, C, D}.
- Since no further dependencies can be applied, the closure of {A} is {A, B, C, D}.

- Example :
- Consider a relation R(A, B, C, D, E) with the following functional dependencies: A -> BC, CD -> E, B -> D, and E -> A.
- Now, let's find the closure of the attribute set {A}.
- Start with the initial set: {A}.
- Apply A -> BC: {A, B, C}.
- Apply B -> D: {A, B, C, D}.
- Apply CD -> E: {A, B, C, D, E}.
- Apply E -> A: The set remains {A, B, C, D, E} as A is already present.
- No further dependencies can be applied, so the closure of {A} is {A, B, C, D, E}.

- **Example:**
- Given a Relation R(A, B, C, D, E) {AB -> CD, D -> E, A -> C,B -> D}.
- we can determine A^+ as:
- Add A to the set S = {A}.
- get all the attribute which are derived from A
    - A->C (From given Functional Dependencies)
    - Add C to the set. Now S will contain {A,C}
- No other attribute can be derived from A and C hence
- **A^+= {A,C}.**
- we can determine B^+ as:
- Add B to the set S = {B}.
- get all the attribute which are derived from B
    - B->D (From given Functional Dependencies)
    - Add D to the set. Now S will contain {B,D}
- get all the attribute which are either derived by D or the combination of BD
    - D->E
    - Add E to the Set. Now S will contain {B,D,E}
- No other attribute can be derived from B,D,E or with their combination hence
- **B^+= {B,D,E}.**

Closure Of Functional Dependency : Calculating Candidate Key

- A Candidate Key of a relation is an attribute or set of attributes that can determine the whole relation or contains all the attributes in its closure.“
- Example: Consider the relation R(A,B,C) with given functional dependencies :
- FD1 : A B
- FD2 : B C
- Now, calculating the closure of the attributes as :
- $\{A\}^+ = \{A, B, C\}$
- $\{B\}^+ = \{B, C\}$
- $\{C\}^+ = \{C\}$
- Clearly, “A” is the candidate key as, its closure contains all the attributes present in the relation “R”.

Minimal Cover /Canonical Cover of Functional Dependency

- In any relational model, there exists a set of functional dependencies. These functional dependencies when closely observed might contain redundant attributes. **The ability of removing these redundant attributes without affecting the capabilities of the functional dependency is known as "Canonical Cover** of Functional Dependency".
- Canonical cover of functional dependency is **sometimes also referred to as "Minimal Cover".** Canonical cover of functional dependency is denoted usi**ng "M_c".**
- A canonical cover or irreducible a set of functional dependencies FD **is a simplified set of FD that has a similar closure as the original set FD.**
- **Important terms:**

1. **Extraneous attributes:** If we can delete an attribute of a functional dependency without modifying the closure of the set of functional dependencies, it is said to be unnecessary.
2. **Canonical cover:** Canonical cover Fc is a set of functional dependencies F such that the following properties are not violated:

- 1) All dependencies in Fc are logically implied by F.
- 2) All dependencies in F are logically implied by Fc.
- 3) There are no extraneous attributes in Fc's functional dependencies.
- 4) Each functional dependency's left side in Fc is distinct. There are no two dependencies $\alpha1 \rightarrow \beta1$ and $\alpha2 \rightarrow \beta2$ in such that $\alpha1 \rightarrow \alpha2$.

Minimal Cover /Canonical Cover of Functional Dependency

- **Algorithm for Canonical Cover as follows:**
- **Step 1:** First use union rules to replace any functional dependence such as A1 -> B1 and A1 -> B2 with A1 -> B1B2.
- **Step 2:** Find functional dependency **Fd** A -> B with an irrelevant attribute in A or B.
- **Step 3:** If any irrelevant attribute is found delete it from A -> B until functional dependency does not change.
- **Example**
- Consider a given relation(X, Y, Z, W) having some attributes, and below are mentioned functional dependencies as follows.
- FD: Y -> X
- FD: XW -> Z
- FD: Z ->XYW

Minimal Cover /Canonical Cover of Functional Dependency

- FD: Y -> X
- FD: XW -> Z
- FD: Z ->XYW
- **Step 1:**
- First, **decompose all functional dependencies using the decomposition rule. That means a single attribute on the right-hand side.** Decomposition of functional dependence as follows.
- **FD 1:** Y -> X
- **FD 2:** XW -> Z
- **FD 3:** Z -> X
- **FD 4:** Z -> Y
- **FD 5:** Z -> W

Minimal Cover /Canonical Cover of Functional Dependency

- **FD 1:** Y -> X
- **FD 2: XW -> Z**
- **FD 3:** Z -> X
- **FD 4:** Z -> Y
- **FD 5:** Z -> W
- **Step 2:**
- In the second step, **we remove all irrelevant attributes from the left-hand side of functional dependencies by finding the closure as follows.**
- Only one functional dependency has two or more attributes that means XW -> Z.
- **$\{X\}^+ = \{X\}$**
- **$\{W\}^+ = \{W\}$**
- **In above both cases X can only determine X and W can only determine W, so no irrelevant attributes are present in the above functional dependencies** and functional dependencies will remain constant and will not be removed.

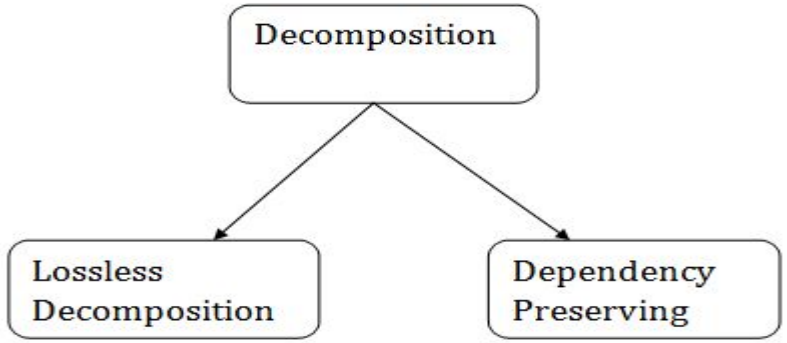
Minimal Cover /Canonical Cover of Functional Dependency

- **Step 3:**
- Remove all functional dependencies that having transitive relation.
- **FD 1:** Y -> X
- **FD 2:** XW -> Z
- **FD 3: Z -> X**
- **FD 4:** Z -> Y
- **FD 5:** Z -> W
- After removing transitive relation the functional dependencies look as follows.
- **FD 1:** Y -> X
- **FD 2:** Z -> Y
- **FD 3:** XW -> Z
- **FD 4:** Z -> W
- We combine 2 and 4 functional dependencies together **now the canonical cover of the above relation R(X, Y, Z, W) is {Y -> X, Z ->YW, XW -> Z}**
- With the help of normal form, we can implement canonical cover.

- **Example**
  Minimize {A->C, AC->D, E->H, E->AD}
- **Step 1**: {A->C, AC->D, E->H, E->A, E->D}
- **Step 2**: {A->C, AC->D, E->H, E->A}
  Here Redundant FD : {E->D}
- **Step 3**: {AC->D}
  {A}+ = {A,C}
  Therefore C is extraneous and is removed.
  {A->D}
- Minimal Cover = {A->C, A->D, E->H, E->A}
- **Example**
  Minimize {AB->C, D->E, AB->E, E->C}
- **Step 1**: {AB->C, D->E, AB->E, E->C}
- **Step 2**: {D->E, AB->E, E->C}
  Here Redundant FD = {AB->C}
- **Step 3**: {AB->E}
  {A}+ = {A}
  {B}+ = {B}
  There is no extraneous attribute.
- Therefore, Minimal cover = {D->E, AB->E, E->C}

Decomposition Properties

- When a relation in the relational model is not appropriate normal form then the decomposition of a relation is required.
- In a database, **breaking down the table into multiple tables termed as decomposition.**
- 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.**
- Types of Decomposition



Decomposition Properties

- **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.

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

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

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

Decomposition Properties

- **Dependency Preserving**
- In the dependency preservation, **at least one decomposed table must satisfy every dependency.**
- If each functional dependency X->Y specified in F appears directly in one of the relation schemas Ri in the decomposition D or could be inferred from the dependencies that appear in some Ri. This is the Dependency Preservation.
- If a decomposition is not dependency preserving some dependency is lost in decomposition. To check this condition, take the JOIN of 2 or more relations in the decomposition.
- For example:
- R = (A, B, C)
- F = {A ->B, B->C}
- **Key = {A}**
- R is not in BCNF.
- **Decomposition R1 = (A, B), R2 = (B, C)**

Decomposition Properties

- **Attribute Preservation:**
- Using functional dependencies the algorithms decompose the universal relation schema R in a set of relation schemas **D = { R1, R2, ….. Rn } relational database schema, where 'D' is called the Decomposition of R.**
- The **attributes in R will appear in at least one relation schema Ri in the decomposition, i.e., no attribute is lost.**
- This is called the *Attribute Preservation* condition of decomposition.
- **Non Additive Join Property:**
- Another **property of decomposition is that D should possess is the *Non Additive Join Property*, which ensures that no spurious/Extra tuples are generated when a NATURAL JOIN operation is applied to the relations** resulting from the decomposition.

What is Normalization?

- Normalization is the **process of organizing the data in the database.**
- Normalization is **used to minimize the redundancy** from a relation or set of relations.
- It is also used to **eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.**
- **Normalization divides the larger table into smaller** and links them using relationships.
- Most commonly used normal forms:
  - **First normal form(1NF)**
  - **Second normal form(2NF)**
  - **Third normal form(3NF)**
  - **Boyce & Codd normal form (BCNF)**
  - **Fourth Normal Form(4NF)**

Why do we need Normalization?

- The **main reason for normalizing the relations is removing these anomalies**. **Failure to eliminate anomalies leads to data redundancy** and can cause data integrity and other problems as the database grows.
- **Update anomalies –** 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.
- **Normalization is a method to remove all these anomalies** and bring the database to a consistent state.
- Normalization consists of a series of guidelines that helps **to guide you in creating a good database structure.**

First normal form(1NF)

- A relation will be 1NF if it contains an **atomic value.**
- It means, A relation is said to be in "1NF" if, every attribute in a relation is has **“Single Valued”** tuple.
- Also, the **domain of the attributes must remain same throughout the table.** For example : E_ID attribute should contain only Employee ID and not any other value.
- Each column in a table should have a **unique name.**
- **First normal form disallows the multi-valued attribute, composite attribute, and their combinations.**

- **Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.**

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

- **The decomposition of the EMPLOYEE table into 1NF has been shown below:**

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

Second normal form(2NF)

- A relation / table to be in the Second Normal Form,

1. **It should be in the First Normal form.**
2. **And, it should not have Partial Dependency.**

- **Partial Dependency:** It is a type of functional dependency that occurs when non-prime attributes are partially dependent on part of Candidate keys.

Employee Table (Before 2NF)

EMP_ID	PROJECT_ID	MANAGER
1	23	Mr.X
1	67	Mr.Z
2	45	Mr.X
3	78	Mr.Y
3	23	Mr.X
4	23	Mr.X
5	78	Mr.Y
5	67	Mr.Z

- In the employee table, **if manager details are to be fetched for an employee, multiple results are returned when searched with EMP_ID, in order to fetch one result, EMP_ID and PROJECT_ID together are considered as the Candidate Keys.**
- **Here the manager depends on PROJECT_ID and not on EMP_ID, this creates a partial dependency.**
- There are multiple ways to eliminate this partial dependency and reduce the table to its 2nd normal form, one such method is adding the Manager information to the project table .

After 2NF

Employee Table

EMP_ID	PROJECT_ID
1	23
1	67
2	45
3	78
3	23
4	23
5	78
5	67

Project Table

PROJECT_ID	PROJECT	MANAGER
23	Extract	Mr.X
67	Load	Mr.Z
45	Extract	Mr.X
78	Transform	Mr.Y
23	Extract	Mr.X
23	Extract	Mr.X
78	Transform	Mr.Y
67	Load	Mr.Z

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 attributes**, then relation must be 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

EMPLOYEE_DETAIL table:

- **Super key in the given table is:**
- **{EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on**
- **Candidate key: {EMP_ID}**
- **Non-prime attributes: In the given 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.**

Third normal form(3NF)

- To convert relation in 3NF,**we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.**

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

EMPLOYEE table:

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

EMPLOYEE_ZIP table:

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 → 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.**
- **Example:** Let's assume there is a company where employees work in more than one department.
- The table is **not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.**

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

EMPLOYEE table

- In the given table Functional dependencies are as follows:
- **EMP_ID → EMP_COUNTRY**
- **EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}**
- **Candidate key: {EMP-ID, EMP-DEPT}**

Boyce & Codd normal form (BCNF)

- **To convert the given table into BCNF, we decompose it into three tables:**

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_COUNTRY table

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT table

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

EMP_DEPT_MAPPING table

- Functional dependencies:
- **EMP_ID → EMP_COUNTRY**
- **EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}**
- Candidate keys:
- For the first table: **EMP_ID**
- For the second table: **EMP_DEPT**
- For the third table**: {EMP_ID, EMP_DEPT}**

- Now, this is in BCNF because left side part of both the functional dependencies is a key.

Fourth Normal Form(4NF)

- A table is said to be in the Fourth Normal Form when,

1. It is in the Boyce-Codd Normal Form.
2. And, it doesn't have Multi-Valued Dependency.

s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
2	C#	Cricket
2	Php	Hockey

- As you can see in the table above, student with s_id 1 has opted for two courses, Science and Maths, and has two hobbies, Cricket and Hockey.

- **You must be thinking what problem this can lead to, right?**
- **Well the two records for student with s_id 1, will give rise to two more records, as shown below, because for one student, two hobbies exists, hence along with both the courses, these hobbies should be specified.**

s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
1	Science	Hockey
1	Maths	Cricket

- And, in the table above, **there is no relationship between the columns course and hobby. They are independent of each other.**
- So there is **multi-value dependency, which leads to un-necessary repetition of data and other anomalies as well.**

Fourth Normal Form(4NF)

- To make the above relation satify the 4th normal form, we can decompose the table into 2 tables.

s_id	course
1	Science
1	Maths
2	C#
2	Php

CourseOpted Table

s_id	hobby
1	Cricket
1	Hockey
2	Cricket
2	Hockey

Hobbies Table
Table

- Now this **relation satisfies the fourth normal form.**
- A table can also have functional dependency along with multi-valued dependency. In that case, the functionally dependent columns are moved in a separate table and the multi-valued dependent columns are moved to separate tables.

Normal Form	Description
1NF	If a table has no repeated groups, it is in 1NF.
2NF	If a table is in 1NF and every non-key attribute is fully dependent on the primary key, then it is in 2NF.
3NF	If a table is in 2NF and has no transitive dependencies, it is in 3NF.
BCNF	If a table is in 3NF and every non-prime attribute fully dependent on the candidate keys, then it is in BCNF.
4NF	If a table is in BCNF and has no multi-valued dependencies, it is in 4NF.