**Functional Dependency**

A functional dependency is a constraint that specifies the relationship between two sets of attributes where one set can accurately determine the value of other sets. It is denoted as **X → Y**, where X is a set of attributes that is capable of determining the value of Y. The attribute set on the left side of the arrow, **X**is called **Determinant**, while on the right side, **Y**is called the **Dependent**. Functional dependencies are used to mathematically express relations among database entities and are very important to understand advanced concepts in Relational Database System.

**Example:**

| roll\_no | name | dept\_name | dept\_building |
| --- | --- | --- | --- |
| 42 | abc | CO | A4 |
| 43 | pqr | IT | A3 |
| 44 | xyz | CO | A4 |
| 45 | xyz | IT | A3 |
| 46 | mno | EC | B2 |
| 47 | jkl | ME | B2 |

* roll\_no → { name, dept\_name, dept\_building },→  Here, roll\_no can determine values of fields name, dept\_name and dept\_building, hence a valid Functional dependency
* roll\_no → dept\_name , Since, roll\_no can determine whole set of {name, dept\_name, dept\_building}, it can determine its subset dept\_name also.
* dept\_name → dept\_building ,  Dept\_name can identify the dept\_building accurately, since departments with different dept\_name will also have a different dept\_building
* More valid functional dependencies: roll\_no → name, {roll\_no, name} ⇢ {dept\_name, dept\_building}, etc.

**Armstrong’s axioms/properties of functional dependencies:**

1. **Reflexivity:**If Y is a subset of X, then X→Y holds by reflexivity rule  
   For example, {roll\_no, name} → name is valid.
2. **Augmentation:** If X → Y is a valid dependency, then XZ → YZ is also valid by the augmentation rule.  
   For example, If {roll\_no, name} → dept\_building is valid, hence {roll\_no, name, dept\_name} → {dept\_building, dept\_name} is also valid.→
3. **Transitivity**: If X → Y and Y → Z are both valid dependencies, then X→Z is also valid by the Transitivity rule.  
   For example, roll\_no → dept\_name & dept\_name → dept\_building, then roll\_no → dept\_building is also valid.
4. **1. Trivial Functional Dependency**
5. In **Trivial Functional Dependency**, a dependent is always a subset of the determinant.  
   i.e. If **X → Y** and **Y is the subset of X**, then it is called trivial functional dependency
6. **For example,**

| roll\_no | name | age |
| --- | --- | --- |
| 42 | abc | 17 |
| 43 | pqr | 18 |
| 44 | xyz | 18 |

1. Here, **{roll\_no, name} → name** is a trivial functional dependency, since the dependent **name** is a subset of determinant set **{roll\_no, name}**  
   Similarly, **roll\_no → roll\_no**is also an example of trivial functional dependency.
2. **2. Non-trivial Functional Dependency**

In **Non-trivial functional dependency**, the dependent is strictly not a subset of the determinant.  
i.e. If **X → Y**and **Y** **is not a subset of X**, then it is called Non-trivial functional dependency.

1. **For example,**

| **roll\_no** | name | age |
| --- | --- | --- |
| 42 | abc | 17 |
| 43 | pqr | 18 |
| 44 | xyz | 18 |

Here, **roll\_no → name** is a non-trivial functional dependency, since the dependent **name** is **not a subset of**determinant**roll\_no**  
Similarly, **{roll\_no, name} → age** is also a non-trivial functional dependency, since **age** is**not a subset of {roll\_no, name}**

**Multivalued Functional Dependency**

In **Multivalued functional dependency**, entities of the dependent set are **not dependent** **on each other.**  
i.e. If **a → {b, c}** and there exists **no functional dependency** between **b and c**, then it is called a **multivalued functional dependency.**

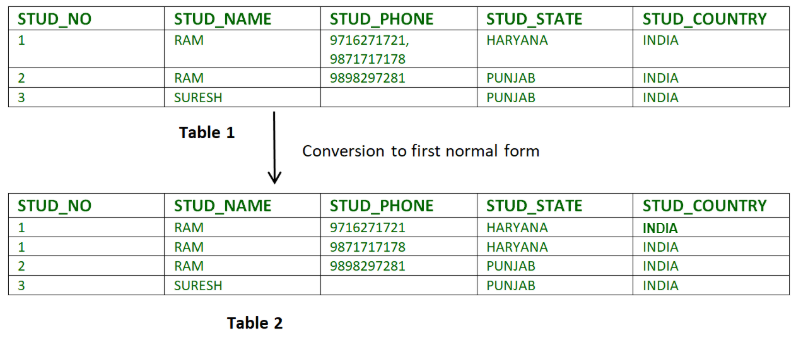
# Normal Forms in DBMS

**Normalization** is the process of minimizing **redundancy** from a relation or set of relations. Redundancy in relation may cause insertion, deletion, and update anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.

**1. First Normal Form –**

If a relation contain composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is **singled valued attribute**.

* **Example 1 –** Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD\_PHONE. Its decomposition into 1NF has been shown in table 2.



**Second Normal Form (2NF):**  
Second Normal Form (2NF) is based on the concept of full functional dependency. Second Normal Form applies to relations with composite keys, that is, relations with a primary key composed of two or more attributes. A relation with a single-attribute primary key is automatically in at least 2NF. A relation that is not in 2NF may suffer from the update anomalies.

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has No Partial Dependency, i.e., no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

In other words,

*A relation that is in First Normal Form and every non-primary-key attribute is fully functionally dependent on the primary key, then the relation is in Second Normal Form (2NF).*

The [normalization](https://www.geeksforgeeks.org/database-normalization-normal-forms/) of 1NF relations to 2NF involves the **removal of partial dependencies**. If a partial dependency exists, we remove the partially dependent attribute(s) from the relation by placing them in a new relation along with a copy of their determinant.

**Example-1:**  
Consider table as following below.

STUD\_NO COURSE\_NO COURSE\_FEE

1 C1 1000

2 C2 1500

1 C4 2000

4 C3 1000

4 C1 1000

2 C5 2000

{Note that, there are many courses having the same course fee. }

Here,  
COURSE\_FEE cannot alone decide the value of COURSE\_NO or STUD\_NO;  
COURSE\_FEE together with STUD\_NO cannot decide the value of COURSE\_NO;  
COURSE\_FEE together with COURSE\_NO cannot decide the value of STUD\_NO;  
Hence,  
COURSE\_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD\_NO, COURSE\_NO} ;  
But, COURSE\_NO -> COURSE\_FEE, i.e., COURSE\_FEE is dependent on COURSE\_NO, which is a proper subset of the candidate key. Non-prime attribute COURSE\_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.

To convert the above relation to 2NF,  
we need to split the table into two tables such as :  
Table 1: STUD\_NO, COURSE\_NO  
Table 2: COURSE\_NO, COURSE\_FEE

**Table 1** **Table 2**

STUD\_NO COURSE\_NO COURSE\_NO COURSE\_FEE

1 C1 C1 1000

2 C2 C2 1500

1 C4 C3 1000

4 C3 C4 2000

4 C1 C5 2000

2 C5

**Note –** 2NF tries to reduce the redundant data getting stored in memory. For instance, if there are 100 students taking C1 course, we dont need to store its Fee as 1000 for all the 100 records, instead once we can store it in the second table as the course fee for C1 is 1000.

**Third Normal Form (3NF):**  
A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency X –> Y:

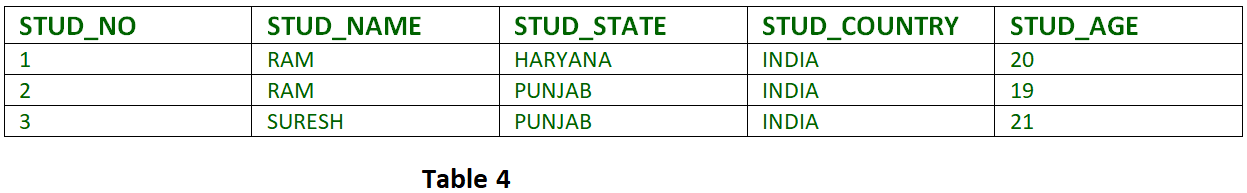
1. X is a super key.
2. Y is a prime attribute (each element of Y is part of some candidate key).

In other words,

The [normalization](https://www.geeksforgeeks.org/database-normalization-normal-forms/) of 2NF relations to 3NF involves the removal of transitive dependencies. If a transitive dependency exists, we remove the transitively dependent attribute(s) from the relation by placing the attribute(s) in a new relation along with a copy of the determinant.

Consider the examples given below.

**Example-1:**  
In relation STUDENT given in Table 4,



FD set:  
{STUD\_NO -> STUD\_NAME, STUD\_NO -> STUD\_STATE, STUD\_STATE -> STUD\_COUNTRY, STUD\_NO -> STUD\_AGE}

Candidate Key:  
{STUD\_NO}

For this relation in table 4, STUD\_NO -> STUD\_STATE and STUD\_STATE -> STUD\_COUNTRY are true. So STUD\_COUNTRY is transitively dependent on STUD\_NO. It violates the third normal form. To convert it in third normal form, we will decompose the relation STUDENT (STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_COUNTRY\_STUD\_AGE) as:

STUDENT (STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_AGE)

STATE\_COUNTRY (STATE, COUNTRY)

Example: Consider relation R(A, B, C, D, E)

A -> BC,

CD -> E,

B -> D,

E -> A

All possible candidate keys in above relation are {A, E, CD, BC} All attribute are on right sides of all functional dependencies are prime.

**Note –**  
Third Normal Form (3NF) is considered *adequate* for normal relational database design because most of the 3NF tables are free of insertion, update, and deletion anomalies. Moreover, 3NF *always ensures functional dependency preserving and lossless*.

# **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.

**EMPLOYEE table:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 |

**In the above table Functional dependencies are as follows:**

1. EMP\_ID  →  EMP\_COUNTRY
2. EMP\_DEPT  →   {DEPT\_TYPE, EMP\_DEPT\_NO}

**Candidate key: {EMP-ID, EMP-DEPT}**

**Example-1:**  
Find the highest normal form of a relation R(A, B, C, D, E) with FD set as:

{ BC->D, AC->BE, B->E }

**Explanation:**

* **Step-1:** As we can see, (AC)+ ={A, C, B, E, D} but none of its subset can determine all attribute of relation, So AC will be candidate key. A or C can’t be derived from any other attribute of the relation, so there will be only 1 candidate key {AC}.
* **Step-2:** Prime attributes are those attribute which are part of candidate key {A, C} in this example and others will be non-prime {B, D, E} in this example.
* **Step-3:** The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute.

The relation is in 2nd normal form because BC->D is in 2nd normal form (BC is not a proper subset of candidate key AC) and AC->BE is in 2nd normal form (AC is candidate key) and B->E is in 2nd normal form (B is not a proper subset of candidate key AC).

The relation is not in 3rd normal form because in BC->D (neither BC is a super key nor D is a prime attribute) and in B->E (neither B is a super key nor E is a prime attribute) but to satisfy 3rd normal for, either LHS of an FD should be super key or RHS should be prime attribute. So the highest normal form of relation will be 2nd Normal form.

Inclusion Dependency

* Multivalued dependency and join dependency can be used to guide database design although they both are less common than functional dependencies.
* Inclusion dependencies are quite common. They typically show little influence on designing of the database.
* The inclusion dependency is a statement in which some columns of a relation are contained in other columns.
* The example of inclusion dependency is a foreign key. In one relation, the referring relation is contained in the primary key column(s) of the referenced relation.
* Suppose we have two relations R and S which was obtained by translating two entity sets such that every R entity is also an S entity.
* Inclusion dependency would be happen if projecting R on its key attributes yields a relation that is contained in the relation obtained by projecting S on its key attributes.
* In inclusion dependency, we should not split groups of attributes that participate in an inclusion dependency.

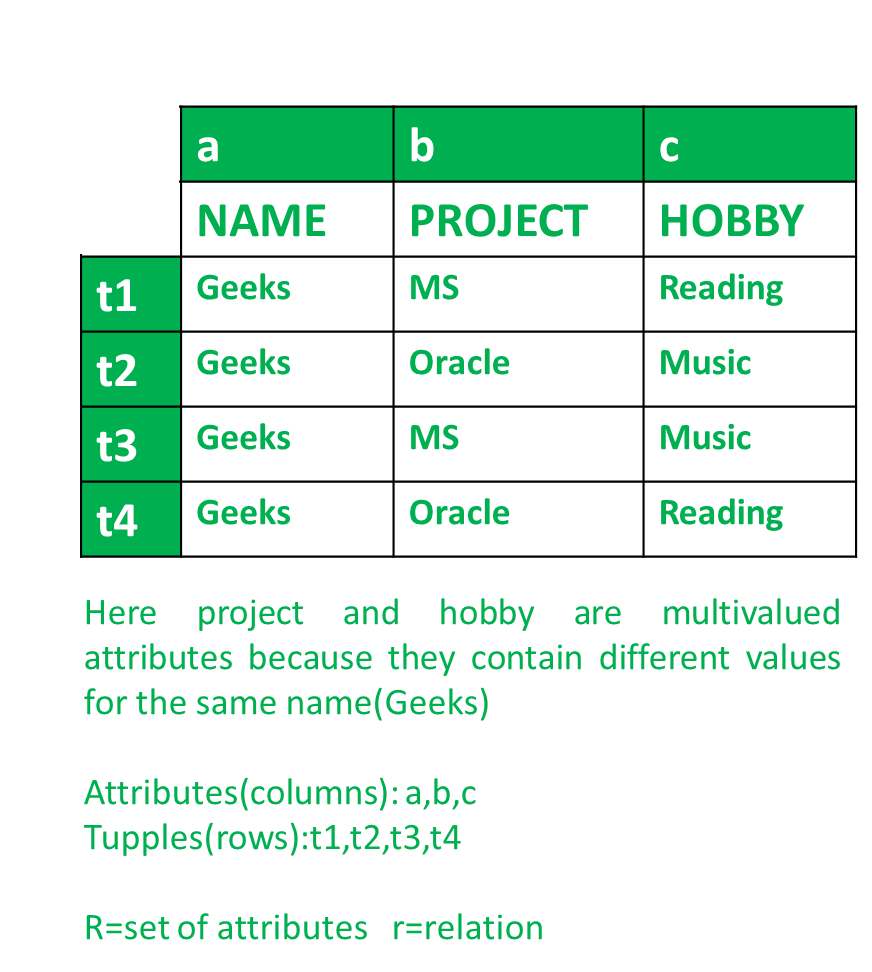
**Multivalued Dependency (MVD) in DBMS**

MVD or multivalued dependency means that for a single value of attribute ‘a’ multiple values of attribute ‘b’ exist. We write it as,

a --> --> b

It is read as: a is multi-valued dependent on b.

Suppose a person named Geeks is working on 2 projects Microsoft and Oracle and has 2 hobbies namely Reading and Music. This can be expressed in a tabular format in the following way.



Project and Hobby are multivalued attributes as they have more than one value for a single person i.e., Geeks.

**Multi Valued Dependency (MVD) :**  
We can say that multivalued dependency exists if the following conditions are met.

**Conditions for MVD :**  
Any attribute say **a** multiple define another attribute b; if any legal relation r(R), for all pairs of tuples t1 and t2 in r, such that,

t1[a] = t2[a]

Then there exists t3 and t4 in r such that.

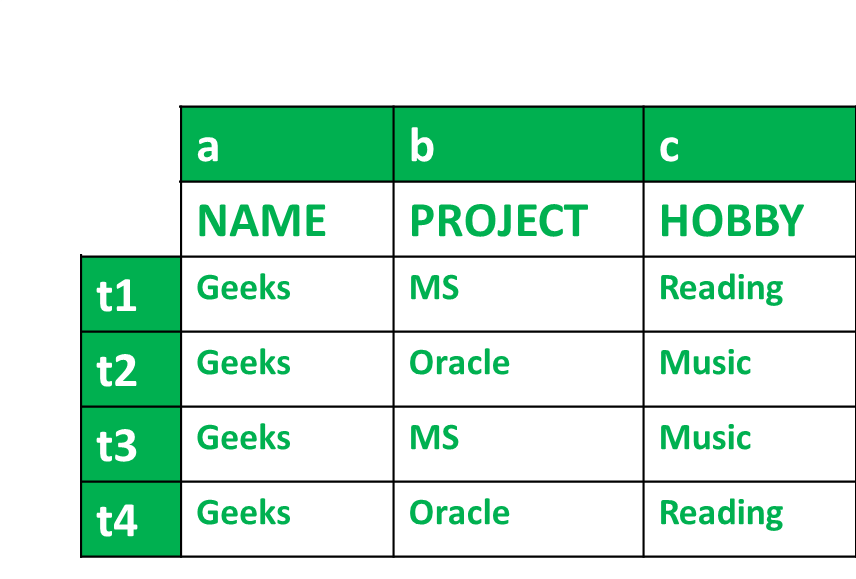
t1[a] = t2[a] = t3[a] = t4[a]

t1[b] = t3[b]; t2[b] = t4[b]

t1 = t4; t2 = t3

Then multivalued (MVD) dependency exists.

To check the MVD in given table, we apply the conditions stated above and we check it with the values in the given table.



**Condition-1 for MVD –**

t1[a] = t2[a] = t3[a] = t4[a]

Finding from table,

t1[a] = t2[a] = t3[a] = t4[a] = Geeks

So, condition 1 is Satisfied.

**Condition-2 for MVD –**

t1[b] = t3[b]

And

t2[b] = t4[b]

Finding from table,

t1[b] = t3[b] = MS

And

t2[b] = t4[b] = Oracle

So, condition 2 is Satisfied.

**Condition-3 for MVD –**

∃c ∈ R-(a ∪ b) where R is the set of attributes in the relational table.

t1 = t4

And

t2=t3

Finding from table,

t1 = t4 = Reading

And

t2 = t3 = Music

So, condition 3 is Satisfied.

All conditions are satisfied, therefore,

a --> --> b

According to table we have got,

name --> --> project

And for,

a --> --> C

We get,

name --> --> hobby

Hence, we know that MVD exists in the above table and it can be stated by,

name --> --> project

name --> --> hobby

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.

**Armstrong’s Axioms in Functional Dependency in DBMS**

The term Armstrong axioms refer to the sound and complete set of inference rules or axioms, introduced by William W. Armstrong, that is used to test the logical implication of **functional dependencies**. If F is a set of functional dependencies then the closure of F, denoted as , is the set of all functional dependencies logically implied by F. Armstrong’s Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

**Axioms –**

Inference Rule (IR):

* The Armstrong's axioms are the basic inference rule.
* Armstrong's axioms are used to conclude functional dependencies on a relational database.
* The inference rule is a type of assertion. It can apply to a set of FD(functional dependency) to derive other FD.
* Using the inference rule, we can derive additional functional dependency from the initial set.

The Functional dependency has 6 types of inference rule:

1. Reflexive Rule (IR1)

In the reflexive rule, if Y is a subset of X, then X determines Y.

1. If X ⊇ Y then X  →    Y

**Example:**

1. X = {a, b, c, d, e}
2. Y = {a, b, c}

2. Augmentation Rule (IR2)

The augmentation is also called as a partial dependency. In augmentation, if X determines Y, then XZ determines YZ for any Z.

1. If X    →  Y then XZ   →   YZ

**Example:**

1. For R(ABCD),  **if** A   →   B then AC  →   BC

3. Transitive Rule (IR3)

In the transitive rule, if X determines Y and Y determine Z, then X must also determine Z.

1. If X   →   Y and Y  →  Z then X  →   Z

4. Union Rule (IR4)

Union rule says, if X determines Y and X determines Z, then X must also determine Y and Z.

1. If X    →  Y and X   →  Z then X  →    YZ

**Proof:**

1. X → Y (given)  
2. X → Z (given)  
3. X → XY (using IR2 on 1 by augmentation with X. Where XX = X)  
4. XY → YZ (using IR2 on 2 by augmentation with Y)  
5. X → YZ (using IR3 on 3 and 4)

5. Decomposition Rule (IR5)

Decomposition rule is also known as project rule. It is the reverse of union rule.

This Rule says, if X determines Y and Z, then X determines Y and X determines Z separately.

1. 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)

6. Pseudo transitive Rule (IR6)

In Pseudo transitive Rule, if X determines Y and YZ determines W, then XZ determines W.

1. If X   →   Y and YZ   →   W then XZ   →   W

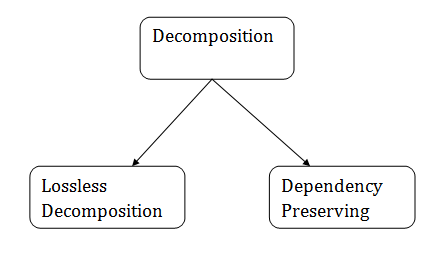
**Proof:**

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)

Relational Decomposition

* 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.

Types of Decomposition



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:**

**EMPLOYEE\_DEPARTMENT table:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **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 | |

**EMPLOYEE table:**

**DEPARTMENT table**

|  |  |  |
| --- | --- | --- |
| **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 ⋈ 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 |

Hence, the decomposition is Lossless join decomposition.

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).