UNIT-III Relational Database Design

1. Relational database design

Relational database design (RDD) models information and data into a set of tables with rows and columns. Each row of a relation/table represents a record, and each column represents an attribute of data. The Structured Query Language (SQL) is used to manipulate relational databases.Relational database is based on the relational model. This database consists of various components based on the relational model. These include:

- **Relation :** Two-dimensional table used to store a collection of data elements.
- **Tuple :** Row of the relation, depicting a real-world entity.
- **Attribute/Field :** Column of the relation, depicting properties that define the relation.
- **Attribute Domain :** Set of pre-defined atomic values that an attribute can take i.e., it describes the legal values that an attribute can take.
- **Degree :** It is the total number of attributes present in the relation.
- **Cardinality :** It specifies the number of entities involved in the relation i.e., it is the total number of rows present in the relation. Read more about Cardinality in DBMS.
- **Relational Schema :** It is the logical blueprint of the relation i.e., it describes the design and the structure of the relation. It contains the table name, its attributes, and their types:

```
TABLE_NAME(ATTRIBUTE_1 TYPE_1, ATTRIBUTE_2 TYPE_2, ...)
```

For our Student relation example, the relational schema will be:

```
STUDENT(ROLL_NUMBER INTEGER, NAME VARCHAR(20), CGPA FLOAT)
```

- **Relational Instance:** It is the collection of records present in the relation at a given time.
- **Relation Key:** It is an attribute or a group of attributes that can be used to uniquely identify an entity in a table or to determine the relationship between two tables. Relation keys can be of 6 different types:
    1. Candidate Key
    2. Super Key
    3. Composite Key
    4. Primary Key
    5. Alternate Key
    6. Foreign Key

Highlights:

1. A Relation is a collection of rows (tuples) and columns (attributes).
2. In a relation, the tuples depicts real-world entity, while the attributes are the properties that define the relation.

3. Structure of the relation is described by the relational schema.
4. Relational keys are used to uniquely identify a row in a table or to determine the relationship between two tables.

Anomalies in Relational Model

When we notice any unexpected behavior while working with the relational databases, there may be a presence of too much redundancy in the data stored in the database. If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.
This can cause anomalies in the DBMS and it can be of various types such as:

Insertion Anomalies: It is the inability to insert data in the database due to the absence of other data. For example: Suppose we are dividing the whole class into groups for a project and the Group Number attribute is defined so that null values are not allowed. If a new student is admitted to the class but not immediately assigned to a group then this student can't be inserted into the database.

Example

We use the same table in the previous example with modified data

stu_id	stu_name	stu_address	stu_club
220	Annamalai	Kerala	yoga
220	Muthu	Kerala	Music
231	Mukesh	Mumbai	Crypto
232	Muni	Karnataka	Public Speaking
232	Muni	Karnataka	Arts

For example, in the above table if a new student named Nanda has joined the college and he has no department affiliation as the club allows intake of students only from second year. Then we can't insert the data of Nanda into the table since the stu_club field cannot accept null values.

Deletion Anomalies - It is the accidental loss of data in the database upon deletion of any other data element. For example: Suppose, we have an employee relation that contains the details of the employee along with the department they are working in. Now, if a department has one employee working in it and we remove the information of this employee from the table, there will be the loss of data related to the department also. This can lead to data inconsistency.

Example In this example, we use modified data from the previous example

stu_id	stu_name	stu_address	stu_club
120	Nanthu	Maharasthra	yoga
122	Nanthu	Maharashthra	Music

stu_id	stu_name	stu_address	stu_club
131	Mukesh	Mumbai	Crypto
132	Muni	Karnataka	Public Speaking
132	Muni	Karnataka	Arts

Suppose, for instance, the college at some point closes the club crypto, then deleting the rows that contain s_club as crypto would also delete the information of student Mukesh since he belongs only to this department.

Modification/Update Anomalies - It is the data inconsistency that arises from data redundancy and partial updation of data in the database. For example: Suppose, while updating the data into the database duplicate entries were entered. Now, if the user does not realize that the data is stored redundantly after updation, there will be data inconsistency in the database.

Consider a college database that keeps student information in a table called student, which contains four columns: stu_id for the student's id, stu_name for the student's name, stu_address for the student's address, and stu_club for the student's club. Eventually, the table will appear as follows:

stu_id	stu_name	stu_address	stu_club
330	Muthu	Rajasthan	Literature
330	Muthu	Rajasthan	Finance
331	Mukesh	Mumbai	Crypto
332	Nanda	Karnataka	Public Speaking
332	Nanda	Karnataka	Arts

For student Muthu, we have two columns in the above table as he belongs to two clubs at the college. If we want to change Muthu's address, we must update it twice otherwise the data will be inconsistent.

When the correct address gets updated in one club but not in another, Muthu would possess two different addresses, which is not acceptable and could result in inconsistent data.

All these anomalies can lead to unexpected behavior and inconvenience for the user. These anomalies can be removed with the help of a process known as normalization.

Highlights:

Any unexpected behavior in a relational database can be caused by an anomaly.
Anomaly occurs mainly due to the presence of data redundancy in the database.
Anomalies are of 3 types i.e., Insertion, Updation, and Deletion anomaly.

Codd Rules in DBMS

Edgar F. Codd, the creator of the relational model proposed 13 rules known as Codd Rules that states: For a database to be considered as a perfect relational database, it must follow the following rules:

1. Foundation Rule - The database must be able to manage data in relational form.
2. Information Rule - All data stored in the database must exist as a value of some table cell.
3. Guaranteed Access Rule - Every unique data element should be accessible by only a combination of the table name, primary key value, and the column name.
4. Systematic Treatment of NULL values - Database must support NULL values.
5. Active Online Catalog - The organization of the database must exist in an online catalog that can be queried by authorized users.
6. Comprehensive Data Sub-Language Rule - Database must support at least one language that supports: data definition, view definition, data manipulation, integrity constraints, authorization, and transaction boundaries.
7. View Updating Rule - All views should be theoretically and practically updatable by the system.
8. Relational Level Operation Rule - The database must support high-level insertion, updation, and deletion operations.
9. Physical Data Independence Rule - Data stored in the database must be independent of the applications that can access it i.e., the data stored in the database must not depend on any other data or an application.
10. Logical Data Independence Rule - Any change in the logical representation of the data (structure of the tables) must not affect the user's view.
11. Integrity independence - Changing the integrity constraints at the database level should not reflect any change at the application level.
12. Distribution independence - The database must work properly even if the data is stored in multiple locations or is being used by multiple end-users.
13. Non-subversion Rule - Accessing the data by low-level relational language should not be able to bypass the integrity rules and constraints expressed in the high-level relational language.

Highlights:

Codd Rules are 13 sets of constraints that a perfect relational database must follow.

Codd Rules were introduced by Edgar F. Codd to resolve the database standardization problem.

Advantages of using the relational model

The advantages and reasons due to which the relational model in DBMS is widely accepted as a standard are:

Simple and Easy To Use - Storing data in tables is much easier to understand and implement as compared to other storage techniques.

Manageability - Because of the independent nature of each relation in a relational database, it is easy to manipulate and manage. This improves the performance of the database.

Query capability - With the introduction of relational algebra, relational databases provide easy access to data via high-level query language like SQL.

Data integrity - With the introduction and implementation of relational constraints, the relational model can maintain data integrity in the database.

Highlights:

Relational databases are simple to use, easy to manage, provide data integrity, and are query capable.

All the advantages of relational databases are because of the use of tables and constraints.

Disadvantages of using the relational model

The main disadvantages of relational model in DBMS occur while dealing with a huge amount of data as:
The performance of the relational model depends upon the number of relations present in the database.
Hence, as the number of tables increases, the requirement of physical memory increases.
The structure becomes complex and there is a decrease in the response time for the queries.
Because of all these factors, the cost of implementing a relational database increase.

Highlights:

Relational databases work perfectly well for a limited number of relations.

Increasing the amount of data can lead to performance and storage issues with relational databases.

2. Domain and Data dependency

Domain and data dependency are two critical aspects of database management system (DBMS). Domain dependency is the relationship between two tables in a database which are related to each other by the same attribute and data dependency is the relationship between two tables in a database which are related to each other by the same data.

Domain dependency is a type of data dependency which states that the domain of the value of an attribute in one table must be the same as the domain of the values of the corresponding attribute in the other table. In other words, the domain of an attribute in one table must match the domain of the corresponding attribute in the other table. This ensures that the data in each table is consistent and that the data in one table can be related to the data in the other table.

Data dependency is a type of data dependency which states that the data in one table must be related to the data in another table. This ensures that the data in each table is consistent and can be related to the data in the other table. This also ensures that the data in the two tables can be used to create a query which returns the desired results.

In summary, domain and data dependency are two critical aspects of database management system (DBMS). Domain dependency states that the domain of an attribute in one table must match the domain of the corresponding attribute in the other table and data dependency states that the data in one table must be related to the data in the other table. These two dependencies ensure that the data in each table is consistent and can be used to create queries which will return the desired results.

3. Armstrong's axioms

Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

1. Reflexive Rule (IR_1)

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

If $X \supseteq Y$ then $X \rightarrow Y$

Example:

$X = \{a, b, c, d, e\}$

$Y = \{a, b, c\}$

2. Augmentation Rule (IR_2)

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

If $X \rightarrow Y$ then $XZ \rightarrow YZ$

Example:

For R(ABCD), if $A \rightarrow B$ then $AC \rightarrow BC$

3. **Transitive Rule (IR_3)**
   In the transitive rule, if X determines Y and Y determine Z, then X must also determine Z.
   If X → Y and Y → Z then X → Z

Secondary Rules –
These rules can be derived from the above axioms.

4. **Union Rule (IR_4)**
   Union rule says, if X determines Y and X determines Z, then X must also determine Y and Z.
   If X → Y and X → Z then X → YZ

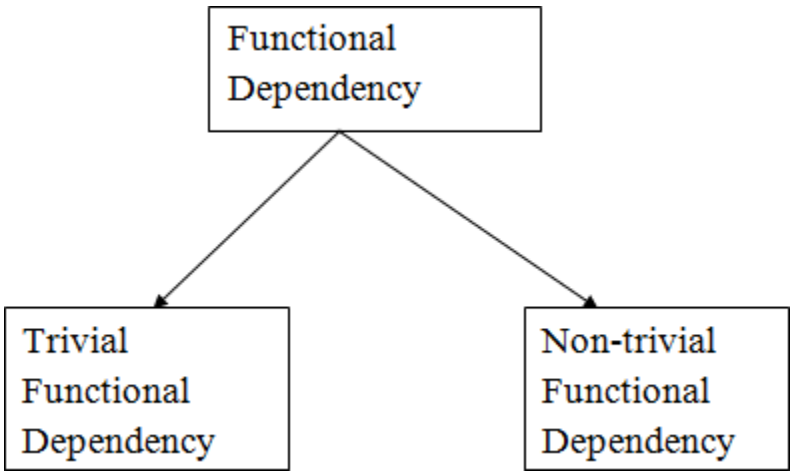
   Proof:
   1. X → Y (given)
   2. X → Z (given)
   3. X → XY (using IR_2 on 1 by augmentation with X. Where XX = X)
   4. XY → YZ (using IR_2 on 2 by augmentation with Y)
   5. X → YZ (using IR_3 on 3 and 4)

5. **Decomposition Rule (IR_5)**
   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.
   If X → YZ then X → Y and X → Z
   Proof:
   1. X → YZ (given)
   2. YZ → Y (using IR_1 Rule)
   3. X → Y (using IR_3 on 1 and 2)
6. **Pseudo transitive Rule (IR_6)**
   In Pseudo transitive Rule, if X determines Y and YZ determines W, then XZ determines W.
   If X → Y and YZ → W then XZ → W
   Proof:
   1. X → Y (given)
   2. YZ → W (given)
   3. XZ → YZ (using IR_2 on 1 by augmenting with Z)
   4. XZ → W (using IR_3 on 3 and 2)

4. Functional Dependencies

Functional Dependency is a constraint between two sets of attributes in relation to a database. A functional dependency is denoted by an arrow (→). If an attribute A functionally determines B, then it is written as A → B.

For example, employee_id → name means employee_id functionally determines the name of the employee. As another example in a timetable database, {student_id, time} → {lecture_room}, student ID and time determine the lecture room where the student should be.



What does functionally dependent mean?

A function dependency A → B means for all instances of a particular value of A, there is the same value of B. For example in the below table A → B is true, but B → A is not true as there are different values of A for B = 3.

A	B
---	---
1	3
2	3
4	0
1	3
4	0

Trivial Functional Dependency
X → Y is trivial only when Y is a subset of X.

Examples

ABC → AB

ABC → A

ABC → ABC

Non Trivial Functional Dependencies
X → Y is a non-trivial functional dependency when Y is not a subset of X.

X → Y is called completely non-trivial when X intersect Y is NULL.

Example:
Id → Name,

Name → DOB

Semi Non Trivial Functional Dependencies
X → Y is called semi non-trivial when X intersect Y is not NULL.

Examples:

AB → BC,

AD → DC

5. Normal forms

Database normalization is the process of organizing the attributes of the database to reduce or eliminate **data redundancy (having the same data but at different places)**.
Problems because of data redundancy: Data redundancy unnecessarily increases the size of the database as the same data is repeated in many places. Inconsistency problems also arise during insert, delete and update operations. **Normal forms** are used to eliminate or reduce redundancy in database tables.

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.

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	INDIA
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

Example 2 –

ID Name Courses

---

1 A c1, c2

2 E c3

3 M C2, c3

In the above table Course is a multi-valued attribute so it is not in 1NF.

Below Table is in 1NF as there is no multi-valued attribute

ID Name Course

---

1 A c1

1 A c2

2 E c3

3 M c2

3 M c3

2. Second Normal Form –

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.

Partial Dependency – If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

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 don't 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.

- **Example 2 –** Consider following functional dependencies in relation R (A, B , C, D)

- AB -> C [A and B together determine C]

  BC -> D [B and C together determine D]

  In the above relation, AB is the only candidate key and there is no partial dependency, i.e., any proper subset of AB doesn't determine any non-prime attribute.

3. Third Normal Form –
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).

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

Table 4

Transitive dependency – If A->B and B->C are two FDs then A->C is called transitive dependency.

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 2 – 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 attributes are on right sides of all functional dependencies are prime.

4. Boyce-Codd Normal Form (BCNF) –
A relation R is in BCNF if R is in Third Normal Form and for every FD, LHS is super key. A relation is in BCNF iff in every non-trivial functional dependency X –> Y, X is a super key.

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}
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 attributes that 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.

Example 2 –For example consider relation R(A, B, C)
A -> BC,
B ->
A and B both are super keys so above relation is in BCNF.
Key Points –

1. BCNF is free from redundancy.
2. If a relation is in BCNF, then 3NF is also satisfied.
3. If all attributes of relation are prime attribute, then the relation is always in 3NF.
4. A relation in a Relational Database is always and at least in 1NF form.
5. Every Binary Relation (a Relation with only 2 attributes) is always in BCNF.
6. If a Relation has only singleton candidate keys(i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF(because no Partial functional dependency possible).
7. Sometimes going for BCNF form may not preserve functional dependency. In that case go for BCNF only if the lost FD(s) is not required, else normalize till 3NF only.
8. There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it's generally not required to go beyond BCNF.

Exercise 1: Find the highest normal form in R (A, B, C, D, E) under following functional dependencies.

ABC --> D

CD --> AE

Important Points for solving above type of question.
1) It is always a good idea to start checking from BCNF, then 3 NF, and so on.
2) If any functional dependency satisfied a normal form then there is no need to check for lower normal form. For example, ABC –> D is in BCNF (Note that ABC is a superkey), so no need to check this dependency for lower normal forms.

Candidate keys in the given relation are {ABC, BCD}

BCNF: ABC -> D is in BCNF. Let us check CD -> AE, CD is not a super key so this dependency is not in BCNF. So, R is not in BCNF.
3NF: ABC -> D we don't need to check for this dependency as it already satisfied BCNF. Let us consider CD -> AE. Since E is not a prime attribute, so the relation is not in 3NF.
2NF: In 2NF, we need to check for partial dependency. CD is a proper subset of a candidate key and it determines E, which is non-prime attribute. So, given relation is also not in 2 NF. So, the highest normal form is 1 NF.

6. Dependency preservation &Lossless design

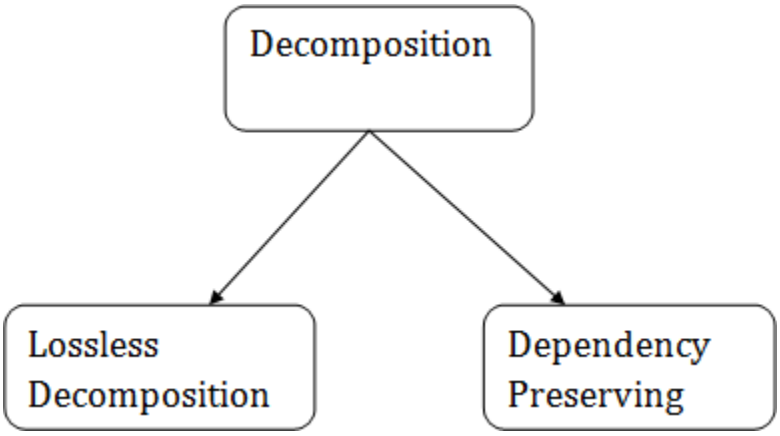
When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.

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

EMPLOYEE table:

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

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).

Decomposition of a relation is done when a relation in relational model is not in appropriate normal form. Relation R is decomposed into two or more relations if decomposition is lossless join as well as dependency preserving.

Lossless Join Decomposition
If we decompose a relation R into relations R1 and R2,

- Decomposition is lossy if $R1 \bowtie R2 \supset R$
- Decomposition is lossless if $R1 \bowtie R2 = R$

To check for lossless join decomposition using FD set, following conditions must hold:

1. Union of Attributes of R1 and R2 must be equal to attribute of R. Each attribute of R must be either in R1 or in R2.
   Att(R1) U Att(R2) = Att(R)

2. Intersection of Attributes of R1 and R2 must not be NULL.
   Att(R1) ∩ Att(R2) ≠ Φ

3. Common attribute must be a key for at least one relation (R1 or R2)
   Att(R1) ∩ Att(R2) -> Att(R1) or Att(R1) ∩ Att(R2) -> Att(R2)

For Example, A relation R (A, B, C, D) with FD set{A->BC} is decomposed into R1(ABC) and R2(AD) which is a lossless join decomposition as:

1. First condition holds true as Att(R1) U Att(R2) = (ABC) U (AD) = (ABCD) = Att(R).
2. Second condition holds true as Att(R1) ∩ Att(R2) = (ABC) ∩ (AD) ≠ Φ
3. Third condition holds true as Att(R1) ∩ Att(R2) = A is a key of R1(ABC) because A->BC is given.

Dependency Preserving Decomposition

If we decompose a relation R into relations R1 and R2, All dependencies of R either must be a part of R1 or R2 or must be derivable from combination of FD's of R1 and R2.

For Example, A relation R (A, B, C, D) with FD set{A->BC} is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of R1(ABC).