MODULE 4- Normalization

- Different anomalies in designing a database, The idea of normalization, Functional dependency,

Armstrong's Axioms (proofs not required), Closures and their computation, Equivalence of

Functional Dependencies (FD), Minimal Cover (proofs not required). First Normal Form (1NF),

Second Normal Form (2NF), Third Normal Form (3NF), Boyce Codd Normal Form (BCNF),

Lossless join and dependency preserving decomposition, Algorithms for checking Lossless Join

(LJ) and Dependency Preserving (DP) properties.

Different anomalies in designing a database

- If a table is not properly normalized and have data redundancy then it will not only eat up extra memory space but will also make it difficult to handle and update the database, without facing data loss.
- Insertion, Updation and Deletion Anomalies are very frequent if database is not normalized.

Student table

rollno	name	branch	hod	office_tel
401	Akon	CSE	Mr. X	53337
402	Bkon	CSE	Mr. X	53337
403	Ckon	CSE	Mr. X	53337
404	Dkon	CSE	Mr. X	53337

- In the table above, we have data of 4 Computer Sci. students.
- As we can see, data for the fields branch, hod(Head of Department) and office_tel is repeated for the students who are in the same branch in the college, this is **Data Redundancy**.

Insertion Anomaly

- Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to set the branch information as **NULL**.
- Also, if we have to insert data of 100 students of same branch, then the branch information will be repeated for all those 100 students.
- These scenarios are nothing but **Insertion anomalies**.

Updation Anomaly

- What if Mr. X leaves the college? or is no longer the HOD of computer science department?
- In that case all the student records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency.
- This is Updation anomaly.

Deletion Anomaly

- In our **Student** table, two different informations are kept together, Student information and Branch information.
- Hence, at the end of the academic year, if student records are deleted, we will also lose the branch information.
- This is Deletion anomaly.

What is Normalization?

- **Normalization** is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization rules divides larger tables into smaller tables and links them using relationships.
- The purpose of Normalisation in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.

Functional dependencies in DBMS

- 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.
- **X** is called **Determinant**,
- **Y** is called the **Dependent**.

- Example

Name	Rollno	CGPA
A	R1	7.6
B	R2	5.5
C	R3	9.2
A	R4	9.1
B	R5	8.7

- Rollno can uniquely determine CGPA.
  Rollno → CGPA (functional dependency)
- Rollno can also determine name.
  Rollno → Name. (functional dependency)
- But CGPA cannot be uniquely determined by Name (CGPA of A → ? - 2 A's we have)
- In general, if A → B, attribute B is functionally dependent on A and A can uniquely determine B.

roll_no	name	dept_name	dept_building
42	abc	CO	A4
43	pqr	IT	A3the
44	xyz	CO	A4

valid functional dependencies:

- 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

invalid functional dependencies:

- name → dept_name
  - Students with the same name can have different dept_name, hence this is not a valid functional dependency.
- dept_building → dept_name
  - There can be multiple departments in the same building, For example, in the above table departments ME and EC are in the same building B2,
  - hence dept_building → dept_name is an invalid functional dependency.

Armstrong's axioms/properties of functional dependencies:

- **Reflexivity:** If Y is a subset of X, then X→Y holds by reflexivity rule
  - For example, {roll_no, name} → name is valid.
- **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.→
- **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.

Types of Functional dependencies in DBMS:

- Fully functional dependency
- Partial Functional Dependency
- Trivial functional dependency
- Non-Trivial functional dependency
- Multivalued functional dependency
- Transitive functional dependency

Fully functional Dependency

- If $X \rightarrow Y$, then Y is said to be fully functional dependent, if Y cannot be determined by any of the proper subset of X.

$ABC \rightarrow D$

$\Downarrow$

fully functional dependent, if D cannot be determined by any of the subset of ABC

$BC \rightarrow D$ ✗

$C \rightarrow D$ ✗

$A \rightarrow D$ ✗

$B \rightarrow D$ ✗

Partial Functional Dependency

→ An FD, $X \rightarrow Y$ is said to be partially functional-dependent, if Y can be determined by any of the proper subset of X.

$AB \rightarrow C$

⇓

partially dependent if C can by determined by A or B.

$A \rightarrow C$

$B \rightarrow C$

Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
  - {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

- 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 conside
  - (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.

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.

- Car(carmodel,mafyear,colour)
  - 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

Transitive Dependency in DBMS

Transitive Functional Dependency

- An FD, $X \rightarrow Y$ is said to be transitive, if there exists a set of attribute Z, such that $X \rightarrow Z$ & $Z \rightarrow Y$ holds.

$$\left.\begin{array}{l} X \rightarrow Z \\ Z \rightarrow Y \end{array}\right\} \Rightarrow X \rightarrow Y$$

- Company} -> {CEO} (if we know the compay, we know its CEO's name)
- {CEO } -> {Age} If we know the CEO, we know the Age
- Therefore according to the rule of transitive dependency:
- { Company} -> {Age} should hold, that makes sense because if we know the company name, we can know his age.

Inference Rules (Axioms) for FDs:

$[X, Y, Z \subseteq R]$

1. Reflexive rule
   - If $Y \subseteq X$, then $X \rightarrow Y$ state that a set of attributes always determines itself or any of its subsets.
   - $X \rightarrow Y$ is trivial if $Y \subseteq X$, otherwise it is not trivial.
2. Augmentation rule
   - If $X \rightarrow Y$, then $XZ \rightarrow YZ$
     - $\Rightarrow$ X, Y, Z denotes the set of attributes over a relation.
3. Transitive rule
   - If $X \rightarrow$ [redacted] Y and $Y \rightarrow Z$, then $X \rightarrow Z$

4. Additive/Union rule

- If $X \to Y$ and $X \to Z$, then $X \to YZ$.

5. Pseudo transitive rule

- If $X \to Y$ and $YZ \to W$, then $XZ \to W$

$$\left\{ \underset{\substack{\downarrow \\ X \\ (X \to Y)}}{\underline{Y}}Z \to W \Rightarrow XZ \to W \right\}$$

6. Productive/decomposition rule.

- If $X \to YZ$, then $X \to Y$ and $X \to Z$.

Closure of an Attribute

- **Closure of an Attribute:** Closure of an Attribute can be defined as a set of attributes that can be functionally determined from it.
- Closure of an attribute X is X+

Find the closure of A,B,C,D given R(A,B,C,D),FD : {A->B,B->D,C->B}

- A+=A->B->D
  - A+=ABD
- B+=B->D
- =BD
- C+=C->B->D
- =CBD
- D+=D

Closure of attribute A-

- Consider a relation R (A , B , C , D , E , F , G) with the functional dependencies-
- $A \rightarrow BC$
- $BC \rightarrow DE$
- $D \rightarrow F$
- $CF \rightarrow G$
-
- $A^+ = \{ A \}$
- $= \{ A , B , C \}$ (Using $A \rightarrow BC$)
- $= \{ A , B , C , D , E \}$ (Using $BC \rightarrow DE$)
- $= \{ A , B , C , D , E , F \}$ (Using $D \rightarrow F$)
- $= \{ A , B , C , D , E , F , G \}$ (Using $CF \rightarrow G$)
- Thus,
- $\mathbf{A^+ = \{ A , B , C , D , E , F , G \}}$

- $A \rightarrow BC$
- $BC \rightarrow DE$
- $D \rightarrow F$
- $CF \rightarrow G$
-

- $D^+ = \{ D \}$
- $= \{ D , F \}$ (Using $D \rightarrow F$)

- $A \rightarrow BC$
- $BC \rightarrow DE$
- $D \rightarrow F$
- $CF \rightarrow G$
-

- $\{ B , C \}^+ = \{ B , C \}$
- $= \{ B , C , D , E \}$ (Using $BC \rightarrow DE$)
- $= \{ B , C , D , E , F \}$ (Using $D \rightarrow F$)
- $= \{ B , C , D , E , F , G \}$ (Using $CF \rightarrow G$)
- Thus,
- **$\{ B , C \}^+ = \{ B , C , D , E , F , G \}$**
-

Given relational schema **R(P Q R S T U V)** having following attribute P Q R S T U and V, also there is a set of functional dependency denoted by **FD = { P->Q, QR->ST, PTV->V }.** Determine Closure of **$(QR)^+$ and $(PR)^+$**

- QR+ = QR FD QR→ST
- =QRST

- PR + = PR → **P** → **Q**
- **=PRQ** →ST
- =PRQST

Given relational schema R(P Q R S T) having following attributes P Q R S and T, also there is a set of functional dependency denoted by FD = { P->QR, RS->T, Q->S, T-> P }. Determine Closure of $(T)^+$

- $T^+ = T \rightarrow P \rightarrow QR \rightarrow S$
- $= TPQRS$

Different kinds of keys

candidate key

- A candidate key may be defined as-
  - A set of minimal attribute(s) that can identify each tuple uniquely in the given relation is called as a candidate key.
- **OR**
  - A minimal super key is called as a candidate key.

- Consider the following Student schema-
- **Student (roll , name , sex , age , address , class , section)**
-
- Given below are the examples of candidate keys-
- (class , section , roll)
- (name , address)
-
- These are candidate keys because each set consists of minimal attributes required to identify each student uniquely in the Student table.
-

Finding Candidate key from a given set of FDs.

Q1) R(A, B, C, D, E, F)

FD : { $A \rightarrow C$,
$C \rightarrow D$,
$D \rightarrow B$,
$E \rightarrow F$ }

Find all the possible candidate keys.

- In the above given set of FDs, C, D, B & F can be determined somehow.
- The attributes that cannot be determined is present in the R.H.S and those attributes will be present in the candidate key set.

2018

MAY 201
W M T W T F S S
18 1 2 3 4 5 6
19 7 8 9 10 11 12 1
20 14 15 16 17 18 19 2
21 21 22 23 24 25 26 2
22 28 29 30 31 - - -

- In the above FD, attributes not present in the R.H.S are A and E
- So find the closure of AE.

$(AE)^+ = AECDBF$

⇒ With AE, we can determine all the other attributes in the relation.

⇒ So AE is the candidate key.

- Let R = (A, B, C, D, E, F) be a relation scheme with the following dependencies-
- $C \rightarrow F$
- $E \rightarrow A$
- $EC \rightarrow D$
- $A \rightarrow B$

- Determine all essential attributes of the given relation.
- Essential attributes of the relation are- C and E.
- So, attributes C and E will definitely be a part of every candidate key.
-

- $C \rightarrow F$
- $E \rightarrow A$
- $EC \rightarrow D$
- $A \rightarrow B$

- So, we have-
- $\{ CE \}^+$
- $= \{ C , E \}$
- $= \{ C , E , F \}$ (Using $C \rightarrow F$)
- $= \{ A , C , E , F \}$ (Using $E \rightarrow A$)
- $= \{ A , C , D , E , F \}$ (Using $EC \rightarrow D$)
- $= \{ A , B , C , D , E , F \}$ (Using $A \rightarrow B$)
- We conclude that CE can determine all the attributes of the given relation.
- So, CE is the only possible candidate key of the relation.

- Let R = (A, B, C, D, E) be a relation scheme with the following dependencies-
- $AB \rightarrow C$
- $C \rightarrow D$
- $B \rightarrow E$
- Determine the total number of candidate keys

- So, we have-
- $\{ AB \}^+$
- $= \{ A , B \}$
- $= \{ A , B, C \}$ (Using $AB \rightarrow C$)
- $= \{ A , B, C, D \}$ (Using $C \rightarrow D$)
- $= \{ A , B, C, D, E \}$ (Using $B \rightarrow E$)
- We conclude that AB can determine all the attributes of the given relation.
- So, AB is the only possible candidate key of the relation

Q2) R(A, B, C, D, E, F, G, H)

FD: $\{ CH \rightarrow G$

$A \rightarrow BC$

$B \rightarrow CFH$

$E \rightarrow A$

$F \rightarrow EG \}$

Find all the possible candidate keys.

- Attribute not present in R.H.S → D

$D^+ = D$ (Take combinations of D)

$(DA)^+ = DABCFHEG$ ✓

$(DB)^+ = DBCFHEGA$ ✓

$(DC)^+ = DC$ ✗

$(DE)^+ = DEABCFHG$

or

$\underline{DE}A$

↓ already in candidate key list, so no need to check other attributes

$(DF)^+ = DFE$ ✓

$(DG)^+ = DG$ ✗

$(DH)^+ \rightarrow DH$ ✗

$(DCH)^+ = DCHG$ ✗

⋮

Candidate key list = { DA, DB, DE, DF }

Closures of a set of functional dependencies

- A **Closure** is a set of FDs is a set of all possible FDs that can be derived from a given set of FDs.
- It is also referred as a **Complete** set of FDs.
- If F is used to donate the set of FDs for relation R, then a closure of a set of FDs implied by F is denoted by F^+.

Q) Consider the relation, $R(A, B, C, D, E)$

$$F = \{ A \rightarrow D,\ D \rightarrow B,\ B \rightarrow C,\ E \rightarrow B \}$$

find A^+, $(BD)^+$, $(AE)^+$. find the candidate key of R.

Normalization Rule

- Normalization rules are divided into the following normal forms:
- First Normal Form
- Second Normal Form
- Third Normal Form
- BCNF
- Fourth Normal Form

- Database Normalization is a technique of organizing the data in the database.
- Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies.
- It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.
- Normalization is used for mainly two purposes,
  - Eliminating redundant(useless) data.
  - Ensuring data dependencies make sense i.e data is logically stored.

First Normal Form (1NF)

- For a table to be in the First Normal Form, it should follow the following 4 rules:
  - It should only have single(atomic) valued attributes/columns.
  - Values stored in a column should be of the same domain
  - All the columns in a table should have unique names.
  - And the order in which data is stored, does not matter.

roll_no	name	subject
101	Akon	OS, CN
103	Ckon	Java
102	Bkon	C, C++

- Our table already satisfies 3 rules out of the 4 rules, as all our column names are unique, we have stored data in the order we wanted to and we have not inter-mixed different type of data in columns.
- But out of the 3 different students in our table, 2 have opted for more than 1 subject.
- And we have stored the subject names in a single column. But as per the 1st Normal form each column must contain atomic value.

How to solve this Problem?

roll_no	name	subject
101	Akon	OS
101	Akon	CN
103	Ckon	Java
102	Bkon	C
102	Bkon	C++

- By doing so, although a few values are getting repeated but values for the subject column are now atomic for each record/row.
- Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

Second Normal Form

- For a table to be in the Second Normal Form, it must satisfy two conditions:
  - The table should be in the First Normal Form.
  - There should be no Partial Dependency.

What is Dependency?

- Let's take an example of a **Student** table with
  columns student_id, name, reg_no branch a nd address .
- In this table, student_id is the primary key and will be unique for every row, hence we can use student_id to fetch any row of data from this table
- Even for a case, where student names are same, if we know the student_id we can easily fetch the correct record.

student_id	name	reg_no	branch	address
10	Akon	07-WY	CSE	Kerala
11	Akon	08-WY	IT	Gujarat

- Hence we can say a **Primary Key** for a table is the column or a group of columns(composite key) which can unique
- if I ask for name of student with student_id **10** or **11**, I will get it.
- So all I need is student_id and every other column **depends** on it, or can be fetched using it.
- This is **Dependency** and we also call it **Functional Dependency**.

Partial Dependency

- For a simple table like Student, a single column like student_id can uniquely identfy all the records in a table.
- But this is not true all the time.
- Let's create another table for **Subject**, which will have subject_id and subject_name fields and subject_id will be the primary key.

Subject

subject_id(Primary Key)	subject_name
1	Java
2	C++
3	Php

- Now we have a **Student** table with student information and another table **Subject** for storing subject information.
- Let's create another table **Score**, to store the **marks** obtained by students in the respective subjects.
- We will also be saving **name of the teacher** who teaches that subject along with marks.

Score

score_id	student_id	subject_id	marks	teacher
1	10	1	70	Java Teacher
2	10	2	75	C++ Teacher
3	11	1	80	Java Teacher

- In the score table we are saving the **student_id** to know which student's marks are these and **subject_id** to know for which subject the marks are for.
- Together, student_id + subject_id forms a **Candidate Key** for this table, which can be the **Primary key**
- Now if you look at the **Score** table, we have a column names teacher which is only dependent on the subject, for Java it's Java Teacher and for C++ it's C++ Teacher & so on.
- primary key for this table is a composition of two columns which is student_id & subject_id but the teacher's name only depends on subject, hence the subject_id, has nothing to do with student_id.
- This is **Partial Dependency**, where an attribute in a table depends on only a part of the primary key and not on the whole key.

How to remove Partial Dependency

- There can be many different solutions for this, but out objective is to remove teacher's name from Score table.
- The simplest solution is to remove columns teacher from Score table and add it to the Subject table. Hence, the Subject table will become:

The simplest solution is to remove columns teacher from Score table and add it to the Subject table. Hence, the Subject table will become:

id	subject_name	teacher
1	Java	Java Teacher
2	C++	C++ Teacher
3	Php	Php Teacher

And our Score table is now in the second normal form, with no partial dependency

score_id	student_id	subject_id	marks
1	10	1	70
2	10	2	75
3	11	1	80

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 the relation must be in third normal form.

Employee Table

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

- **Super key in the table above:**
  - {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EM
- **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.(a→b,b→c a→c)
  - So need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

Employee Table

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007

EMPLOYEE_ZIP table

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago

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.

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

- **In the above table Functional dependencies are as follows:**
  - EMP_ID → EMP_COUNTRY
  - EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}
- **Candidate key: {EMP-ID, EMP-DEPT}**
- The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.
- To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

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

EMP_DEPT_MAPPING table

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

- **Candidate keys:**
  - **For the first table:** EMP_ID
    For the second table: EMP_DEPT
    For the third table: {EMP_ID, EMP_DEPT}
- **Functional dependencies:**
  - EMP_ID → EMP_COUNTRY
  - EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}
- Now, this is in BCNF because left side part of both the functional dependencies is a key.
-

Fourth Normal Form (4NF)

- For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:
  - It should be in the **Boyce-Codd Normal Form**.
  - And, the table should not have any **Multi-valued Dependency**.

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing

- 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:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing

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.

- **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 $R1 \bowtie R2 = R$

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

DEPARTMENT table

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

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

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

- 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.
  - $Att(R1) \cup Att(R2) = Att(R)$
- Intersection of Attributes of R1 and R2 must not be NULL.
  - $Att(R1) \cap Att(R2) \neq \Phi$
- Common attribute must be a key for at least one relation (R1 or R2)
  - $Att(R1) \cap Att(R2) \rightarrow Att(R1)$ or $Att(R1) \cap Att(R2) \rightarrow Att(R2)$

A relation R (A, B, C, D) with FD set{A->BC}.Perform decomposition and check whether it is lossy or lossless ?

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

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).

Thank you