Relational Database design

Functional Dependency

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

$$X \rightarrow Y$$

- The left side of FD is known as a determinant, the right side of the production is known as a dependent.
- Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.
  - Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Emp_Id → Emp_Name

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

Valid Functional Dependency

roll_no → { name, dept_name, dept_building }

roll_no can determine values of fields name, dept_name and dept_building, hence a valid Functional dependency

roll_no → dept_name

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

roll_no → name

{roll_no, name} ⇢ {dept_name, 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.

name → roll_no

{name, dept_name} → roll_no

dept_building → roll_no

Armstrong's axioms/properties of functional dependencies:

Reflexivity: If Y is a subset of X, then X→Y holds by reflexivity rule

{roll_no, name} → name is valid.

Augmentation: If X → Y is a valid dependency, then XZ → YZ is also valid by the augmentation rule.

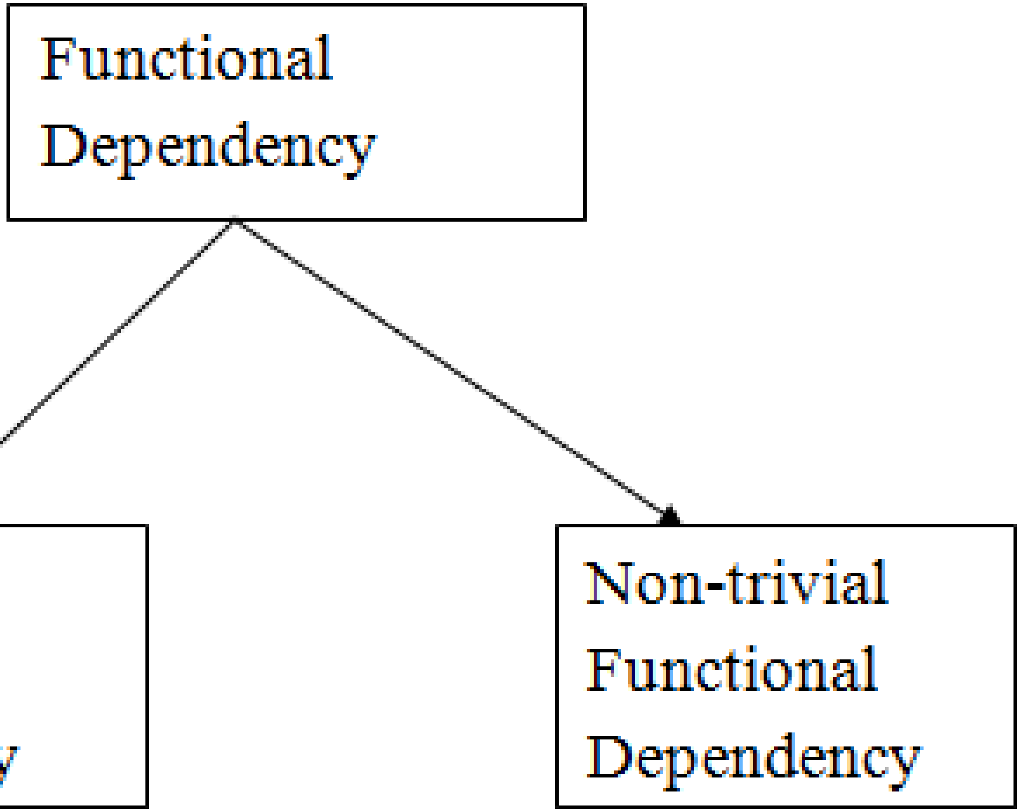
{roll_no, name} → dept_building is valid

{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.

roll_no → dept_name & dept_name → dept_building, then roll_no → dept_building is also valid.

Types of Functional dependency



Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B
  - Consider a table with two columns Employee_Id and Employee_Name.
  - {Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as
  - Employee_Id is a subset of {Employee_Id, Employee_Name}.
  - Also, Employee_Id → Employee_Id and Employee_Name →Employee_Name are trivial dependencies too.

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

{roll_no, name} → name

is a trivial functional dependency, since the dependent **name** is a subset of determinant set **{roll_no, name}**

roll_no → roll_no

is also an example of trivial functional dependency.

Non-trivial functional dependency

- A → B has a non-trivial functional dependency if B is not a subset of A.
- When A intersection B is NULL, then A → B is called as complete non-trivial.
  - ID → Name,
  - Name → DOB

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

roll_no → name

is a non-trivial functional dependency, since the dependent **name** is **not a subset of** determinant **roll_no**

{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.**
  - If **a → {b, c}** and there exists **no functional dependency** between **b and c**, then it is called a **multivalued functional dependency.**

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

roll_no → {name, age}

is a multivalued functional dependency, since the dependents **name** & **age** are **not dependent** on each other(i.e. **name → age** or **age → name doesn't exist !**)

Transitive Functional Dependency

- In transitive functional dependency, dependent is indirectly dependent on determinant.
  - If **a** $\rightarrow$ **b** & **b** $\rightarrow$ **c**, then according to axiom of transitivity, **a** $\rightarrow$ **c**. This is a **transitive functional dependency**

roll_ no	na me	dept_na me	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

enrol_no → dept and **dept → building_no**

according to the axiom of transitivity, **enrol_no → building_no** is a valid functional dependency. This is an indirect functional dependency, hence called Transitive functional dependency.

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^+.

Example

- **F = {A -> B, B -> C, C -> D}**
  - from F, it is possible to derive following dependencies.
    - A -> A ...By using Rule-4, Self-Determination.
    - A -> B ...Already given in F.
    - A -> C ...By using rule-3, Transitivity.
    - A -> D ...By using rule-3, Transitivity.

    - it is possible to derive A^+ -> ABCD

- 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 $(PR)^+$**

    - Now as per algorithm look into a set of FD that complete the left side of any FD contains either Q, R, or QR since in FD QR→ST has complete QR.

Hence QR+ = QRST
PR+ = 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+ = TPQRS

Consider the relation $X(P, Q, R, S, T, U)$ with the following set of functional dependencies

$$F = \{\ \{P, R\} \rightarrow \{S, T\},\ \ \{P, S, U\} \rightarrow \{Q, R\}\ \}$$

Which of the following is the trivial functional dependency in F^+, where F^+ is closure to F?

A. $\{P, R\} \rightarrow \{S, T\}$
B. $\{P, R\} \rightarrow \{R, T\}$
C. $\{P, S\} \rightarrow \{S\}$
D. $\{P, S, U\} \rightarrow \{Q\}$

Attribute Closure

- An attribute set can be defined as set of attributes which can be functionally determined from it.
- **How to find attribute closure of an attribute set?**
  - Add elements of attribute set to the result set.
  - Recursively add elements to the result set which can be functionally determined from the elements of the result set.

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

(STUD_NO)+ = {STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY, STUD_AGE}

(STUD_STATE)+ = {STUD_STATE, STUD_COUNTRY}

Normalization

- “Database Normalization” is a process or technique to reduce the attribute redundancy and functional dependency within the set of tables present in any database.
- Redundancy needs to be eliminated because of its undesirable ability to generate multiple issues in the whole database.
- Redundancy can be a major cause of concern while Inserting, Deleting and Updating the data in the tables and these issues are commonly known as “Anomalies” i.e. “Insertion Anomaly, Deletion Anomaly and Updation Anomaly”.

“Anomaly” means “Inconsistency” in data.

Employee_ID	Name	Department	Student_Group
123	J. Longfellow	Accounting	Beta Alpha Psi
234	B. Rech	Marketing	Marketing Club
234	B. Rech	Marketing	Management Club
456	A. Bruchs	CIS	Technology Org.
456	A. Bruchs	CIS	Beta Alpha Psi

Normalization is the process of **splitting relations into well structured relations** that allow users to insert, delete, and update tuples without introducing database

- An **update anomaly** is a data inconsistency that results from data redundancy and a partial update. If A. Bruchs' department is an error it must be updated at least 2 times or there will be inconsistent data in the database.
- A **deletion anomaly** is the unintended loss of data due to deletion of other data. For example, if the student group Beta Alpha Psi disbanded and was deleted from the table above, J. Longfellow and the Accounting department would cease to exist.
- An **insertion anomaly** is the inability to add data to the database due to absence of other data. If a new employee is hired but not immediately assigned to a Student_Group then this employee could not be entered into the database.

Is There Any Solution?

- **Without applying any solution to anomalies, database normalization cannot be achieved.**

For this, we can **split or decompose the whole relation.**

Properties of Decomposition

- No information is lost from the original relation during **decomposition**.
- When the **sub relations are joined back**, the same relation is obtained that was decomposed.

Dependency preservation ensures:

- None of the functional dependencies that holds on the original relation are lost.
- The sub relations still hold or satisfy the functional dependencies of the original relation.

• Lossy Decomposition

- Consider there is a relation R which is decomposed into sub relations R_1 , R_2 , …. , R_n.
- This decomposition is called lossy join decomposition **when the join of the sub relations does not result in the same relation R** that was decomposed.
- Lossy join decomposition is also known as **careless decomposition**.

$$R_1 \bowtie R_2 \bowtie R_3 \ldots\ldots\ldots \bowtie R_n \supset R$$

where ⋈ is a natural join operator

Id	Fname	lname
1	Naisargi	Shah
2	Nishtha	Prajapati
3	aman	Verma
4	Payal	Gohil
5	Purnab	Goswami
6	aman	deva

Student1(id,fname)

Id	Fname
1	Naisargi
2	Nishtha
3	aman
4	Payal
5	Purnab
6	aman

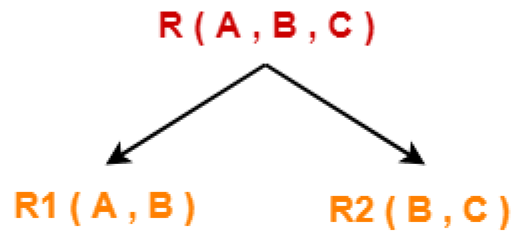
Studen2(fname,lname)

Fname	lname
Naisargi	Shah
Nishtha	Prajapati
aman	Verma
Payal	Gohil
Purnab	Goswami
aman	deva

Types of Decomposition

- **Lossless Join Decomposition**
- Now, let us check whether this decomposition is lossless or not.
- For lossless decomposition, we must have-
- $R_1 \bowtie R_2 = R$

Student(id,fname,lname)



Id	Fname	lname
1	Naisargi	Shah
2	Nishtha	Prajapati
3	aman	Verma
4	Payal	Gohil
5	Purnab	Goswami
6	aman	deva

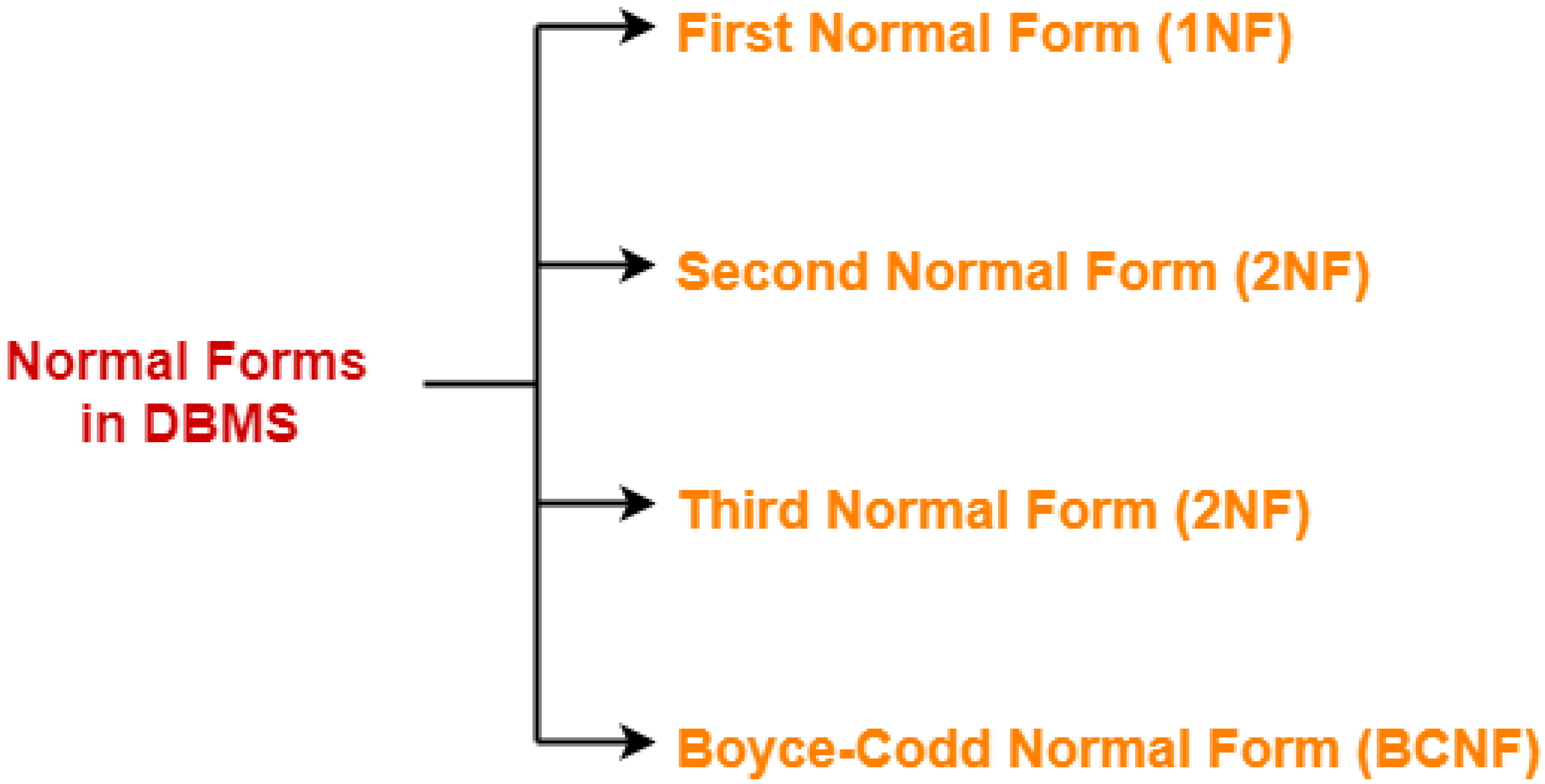
Student1(id,fname)

Id	Fname
1	Naisargi
2	Nishtha
3	aman
4	Payal
5	Purnab
6	aman

Student2(id,lname)

Id	lname
1	Shah
2	Prajapati
3	Verma
4	Gohil
5	Goswami
6	deva

https://www.gatevidyalay.com/decomposition-of-a-relation/


Normal Forms in DBMS
First Normal Form (1NF)
Second Normal Form (2NF)
Third Normal Form (2NF)
Boyce-Codd Normal Form (BCNF)

First Normal Form (1NF)

- A given relation is called in First Normal Form (1NF) **if each cell of the table contains only an atomic value**

Student_id	Name	Subjects
100	Akshay	Computer Networks, Designing
101	Aman	Database Management System
102	Anjali	Automata, Compiler Design

Student_id	Name	Subjects
100	Akshay	Computer Networks
100	Akshay	Designing
101	Aman	Database Management System
102	Anjali	Automata
102	Anjali	Compiler Design

Second Normal Form (2NF)

A given relation is called in Second Normal Form (2NF) if and only if-

1. Relation already exists in 1NF.
2. **No partial dependency exists in the relation**.

In this example

employee_No-> employee_name

dept_No- > dept_name

For ex: A->B, C->D but if you break relation(A,B,C,D) into R1(A,B) and R2(C,D) then it will be lossy decomposition?

That's why R1(A,B,C) and R2(C,D)

Employee No	Department No	Employee Name	Department
1	101	Amit	OBIEE
2	102	Divya	COGNOS
3	101	Rama	OBIEE

Employee No	Department No	Employee Name
1	101	Amit
2	102	Divya
3	101	Rama

Table 2:Department table

Department No	Department
101	OBIEE
102	COGNOS

Third Normal Form (3NF)

In this example
employee_No-> Salary_SlipNo
Salary_SlipNo-> Salary

For ex: A->B, B->C   means A->C

The database is in Third normal form if it satisfies following conditions:

- It is in Second normal form
- There is **no transitive functional dependency** for non-prime attributes, then the relation must be in third normal form.

Employee No	Salary Slip No	Employee Name	Salary
1	0001	Amit	50000
2	0002	Divya	40000
3	0003	Rama	57000

Employee No	Salary Slip No	Employee Name
1	0001	Amit
2	0002	Divya
3	0003	Rama

Salary Table:

Salary Slip No	Salary
0001	50000
0002	40000
0003	57000

Following are 2 Advantages of 3rd normal form:
1. Amount of **data duplication is removed** because **transitive dependency is removed in third normal form.**
2. Achieved **Data integrity**

Employee Details 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
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

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

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

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.
- Super Key = “A superkey is a **combination of columns** that **uniquely identifies any row** within a relational database management system (RDBMS) 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

These all the possible candidate key of above table

1.EMP_ID → EMP_COUNTRY
2.EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Super key: {EMP-ID, EMP-DEPT}

EMP_ID	EMP_COUNTRY
264	India
364	UK

Employee Country

To achieve the BCNF we have to split the table in three table

Employee Department

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

EMP_ID	EMP_DEPT
264	283
264	300
364	232
364	549

Employee Department _ Mapping

1. EMP_ID → EMP_COUNTRY
2. 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}

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and **has no multi-valued dependency.**
- For a dependency $A \rightarrow B$, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

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

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.

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

Student _ Course

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Student _ Hobby

Fifth normal form (5NF)

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

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

John takes both **Computer and Math class** for **Semester 1** but he doesn't take **Math class for Semester 2**. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as **Semester 3** but **do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL**. But all **three columns together acts as a primary key**, so we can't leave other two columns blank.

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

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

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

Chapter 5 Complete