MODULE 4

Normalization
If a database design is not perfect, it may contain anomalies, managing a database with anomalies is not possible. There are three types of anomalies:

- **Update anomalies** − If data items are scattered and are not linked to each other properly, then it could lead to strange situations.

  For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.

- **Deletion anomalies** − We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.

- **Insert anomalies** − We tried to insert data in a record that does not exist at all.

Normalization

Normalization is a method to remove all these anomalies and bring the database to a consistent state.

1. First Normal Form

First Normal Form is defined in the definition of relations (tables) itself. This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units.

Course	Content
Programming	Java, c++
Web	HTML, PHP, ASP

We re-arrange the relation (table) as below, to convert it to First Normal Form.

Course	Content
Programming	Java
Programming	c++
Web	HTML
Web	PHP
Web	ASP

Each attribute must contain only a single value from its pre-defined domain.

2. Second Normal Form

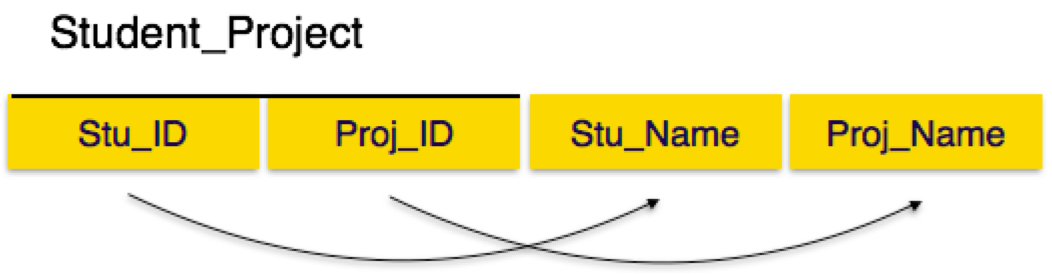
Prime attribute − An attribute, which is a part of the candidate-key, is known as a prime attribute.

Non-prime attribute − An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.
In second normal form, a relation must be in first normal form and relation must not contain any partial dependency.

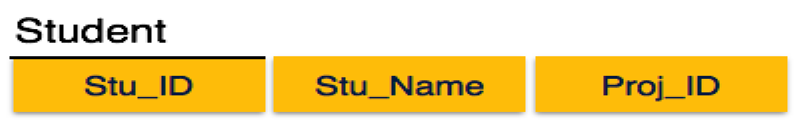
A relation is in 2NF if it has **No Partial Dependency,** i.e.**,** no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

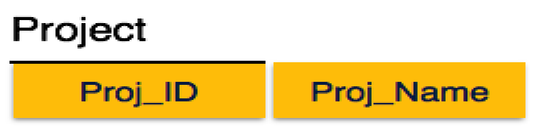
Partial Dependency – If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

Every non-prime attribute should be fully functionally dependent on prime key attribute



Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non-key attributes, i.e. Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attribute individually. But we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called partial dependency, which is not allowed in Second Normal Form.





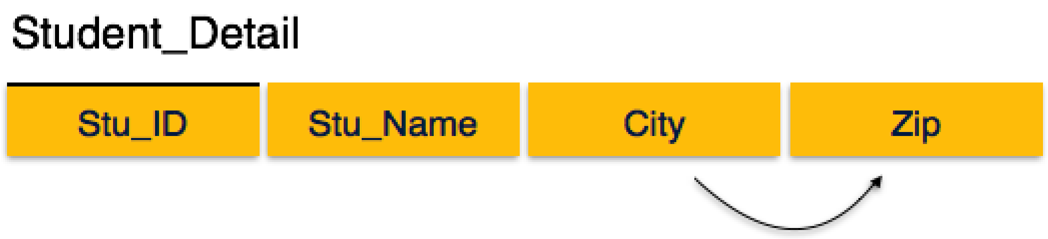
Here there is no partial dependency.

3.Third Normal Form
For a relation to be in Third Normal Form, it must be in Second Normal form and the following must satisfy −

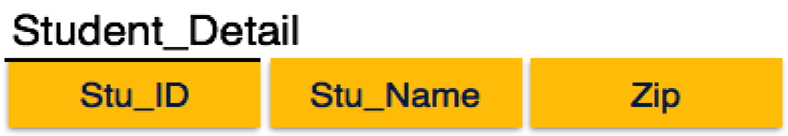
- No non-prime attribute is transitively dependent on prime key attribute.
- For any non-trivial functional dependency, X → A, then either −

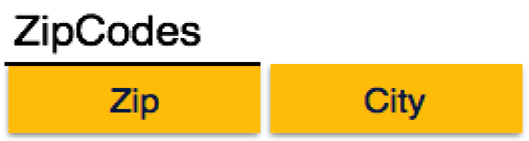
X is a superkey or,

- A is prime attribute.



In the above Student_detail relation, Stu_ID is the key and only prime key attribute.
City can be identified by Stu_ID as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, Stu_ID → Zip → City, so there exists **transitive dependency**.
To bring this relation into third normal form, we break the relation into two relations as follows −





4. Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that −

- For any non-trivial functional dependency, X → A, X must be a super-key.

Stu_ID is the super-key in the relation Student_Detail and Zip is the super-key in the relation ZipCodes.

So,
Stu_ID → Stu_Name, Zip
and
Zip → City
Which confirms that both the relations are in BCNF.

5. 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 → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example
STUDENT

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.
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
74	Biology
59	Physics

STUDENT_HOBBY

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

6. 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).

Example

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

In the above table, 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.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

P1

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

P2

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

P3

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

Functional Dependency

Functional Dependency is a constraint between two sets of attributes in a relation from a database. A functional dependency is denoted by arrow (→). If an attributed 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 time table 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 subset of X.

Examples

ABC → AB
ABC → A
ABC → ABC

Non Trivial Functional Dependencies

X → Y is a non trivial functional dependencies when Y is not a subset of X.

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

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

Decomposition

Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy

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.

Multivalued Dependency

- Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
- A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

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.