Normalization

Normalization is the process of organizing the data in the database.

Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.

Normalization divides the larger table into the smaller table and links them using relationship.

The normal form is used to reduce redundancy from the database table.

Types of Normal Forms

There are the four types of normal forms:

1. 1NF
2. 2NF
3. 3NF
4. BCNF

Normal form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.

5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

Importance of Normalization

Resolving the database anomalies

The forms of Normalization i.e. 1NF, 2NF, 3NF, BCF, 4NF and 5NF remove all the Insert, Update and Delete anomalies.

Insertion Anomaly occurs when you try to insert data in a record that does not exist.

Deletion Anomaly is when a data is to be deleted and due to the poor deign of database, other record also deletes.

Updation Anaomaly occurs when you try to update the values in the database.

Roll	Stdname	Courseid	Coursename	Facultyid	Facultyname	Salary
1	Pabitra	BCA	DBMS	KG	Kiran	10000
2	Rabina	BCA	DBMS	KG	Kiran	10000
3	Saughat	BCA	DBMS	KG	Kiran	10000
4	Sangam	BBS	Account	BS	Bijay	20000
5	Bibek					

Tables:

Roll	Stdname

Courseid	Coursename

Facultyid	Facultyname	Salary

Eliminate Redundancy of Data

Storing same data item multiple times is known as Data Redundancy. A normalized table do not have the issue of redundancy of data.

Data Dependency

The data gets stored in the correct table and ensures normalization.

Isolation of Data

A good designed database states that the changes in one table or field do not affect other. This is achieved through Normalization.

Data Consistency

While updating if a record is left, it can led to inconsistent data, Normalization resolves it and ensures Data Consistency.

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 → Y

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

Here 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.

Functional dependency can be written as:

Emp_Id → Emp_Name
We can say that Emp_Name is functionally dependent on Emp_Id.

Sid → Sname

Sid	Sname
1	Pabitra
1	Pabitra

Sid	Sname
1	Pabitra
2	Anish

Sid	Sname
1	Pabitra

2	Pabitra

Sid	Sname
1	Pabitra
1	Anish

Types of Functional Dependency

1. 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

   For example

   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 tri vial dependencies too.

2. 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.

   For example:

   ID → Name,

   Name → DOB

First Normal Form (1NF)

A relation will be 1NF if it contains an atomic value.

It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.

First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Check whether it is in 1NF or not if not convert it into 1NF.

Roll	Name	Programming Languages
1	Sonam	C
2	Anish	Python
3	Jamuna	C,c++
4	Krishna	Java
5	Anjit	C++,Python

Roll	Name	Programming Languages
1	Sonam	C
2	Anish	Python
3	Jamuna	C
3	Jamuna	C++
4	Krishna	Java
5	Anjit	C++
5	Anjit	Python

Second Normal Form (2NF)

In the 2NF, relational must be in 1NF.

In the second normal form, all non-prime attributes are fully functional dependent on the candidate key.

Example:

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

To convert the given table into 2NF, we decompose it into two tables:

{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		

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,

A relation that is in First and Second Normal Form and in which no non-primary-key attribute is transitively dependent on the primary key, then it is in Third Normal Form (3NF).

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

The normalization 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,

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

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)

BCNF

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}

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

Functional dependencies:

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}

Now, this is in BCNF because left side part of both the functional dependencies is a key.