Normalization

Database 2 - Lecture 5

1. Introduction

Normalization is the process of organizing data in a database. This includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating two factors: redundancy and inconsistent dependency. Redundant data wastes disk space and creates maintenance problems. If data that exists in more than one place must be changed, the data must be changed in exactly the same way in all locations. Inconsistent dependencies can make data difficult to access; the path to find the data may be missing.

Normalization is the analysis of functional dependencies between attributes. Normalization is a formal process for deciding which attributes should be grouped together in a relation. It is the primary tool to validate and improve a logical design so that it satisfies certain constraints that avoid unnecessary duplication of data.

Normalization theory is based on the concepts of normal forms. A relational table is said to be a particular normal form if it satisfied a certain set of constraints. There are currently five normal forms that have been defined. Normalization should remove redundancy but not at the expense of data integrity. In general, the normalization process generates many simple entity specifications from a few semantically complex entity specifications.

2. Purpose of Normalization

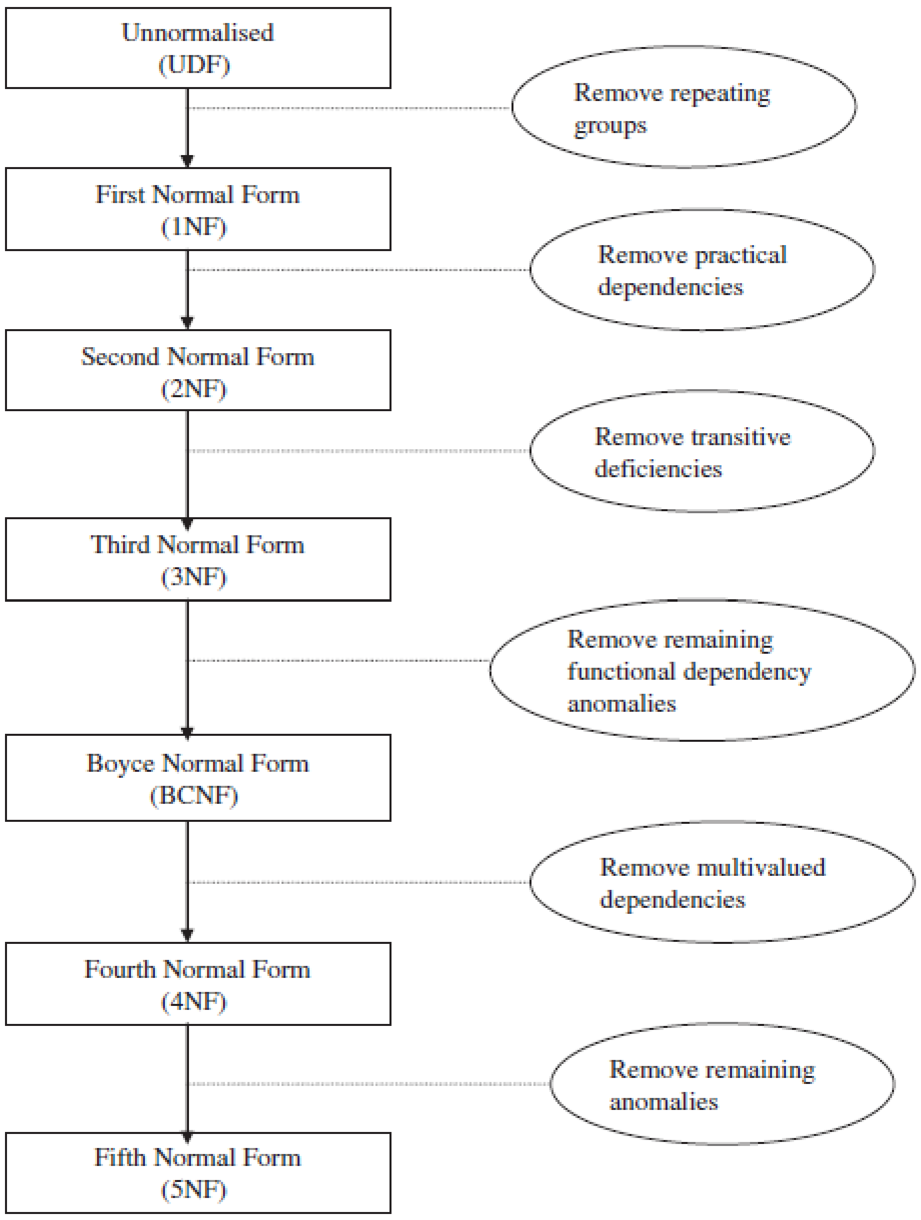
Normalization allows us to minimize insert, update, and delete anomalies and help maintain data consistency in the database.

1. To avoid redundancy by storing each fact within the database only once.
2. To put data into the form that is more able to accurately accommodate change.
3. To avoid certain updating “anomalies”
4. To facilitate the enforcement of data constraint
5. To avoid unnecessary coding. Extra programming in triggers, stored procedures can be required to handle the non-normalized data and this in turn can impair performance significantly.

3. Steps in Normalization

The degree of normalization is defined by normal forms. The normal forms in an increasing level of normalization, are first normal form (1NF), second normal form (2NF), 3NF, Boyce-Codd Normal form, 4NF and 5NF. Each normal form is a set of conditions on a schema that guarantees certain properties relating to redundancy and update anomalies.

In general 3NF is considered good enough. Many modern-day commercial relational database implementations do not go beyond the implementation of 3NF. Application of Normal Forms beyond that of 3NF tends to produce too many tables, resulting in too many tables in SQL joins. Bigger joins result in poor performance. In general, good performance is much more important than granular perfection in relational database design.



Relational theory defines a number of structure conditions called normal forms that assure that certain data anomalies do not occur in a database.

First Normal Form (1NF)

A table is in first normal form (1NF) if and only if all columns contain only atomic values; that is, there are no repeating groups (columns) within a row. It is to be noted that all entries in a field must be of same kind and each field must have a unique name, but the order of the field (column) is irrelevant. Each record must be unique and the order of the rows is irrelevant.

Second Normal Form (2NF)

A table is in second normal form (2NF) if and only if it is in 1NF and every nonkey attribute is fully dependent on the primary key.

Third Normal Form (3NF)

To be in Third Normal Form (3NF) the relation must be in 2NF and no transitive dependencies may exist within the relation. A transitive dependency is when an attribute is indirectly functionally dependent on the key (that is, the dependency is through another nonkey attribute).

Boyce–Codd Normal Form (BCNF)

To be in Boyce–Codd Normal Form (BCNF) the relation must be in 3NF and every determinant must be a candidate key.

Fifth Normal Form (5NF)

The Fifth Normal Form concerns dependencies that are obscure.

3.1 Unnormal Form to First Normal Form

Consider a table DEPARTMENT, the table DEPARTMENT is not in normal form because the table DEPARTMENT has repeating group. The table DEPARTMENT is shown in Table 1.

Table 1. DEPARTMENT (unnormalized form)

Department number	Department name	Location
1	Nilgiris	{Coimbatore, Chennai}
2	Subiksha	{Chennai, Tirunelveli}
3	Krishna	Trichy
4	Kannan	Coimbatore

Table 1 is not in normal form because the values are not atomic. The intersection of row with the column should have only one value. But in Table 1, the department location value is not atomic. That is the department Nilgiris is located in more than one location (Coimbatore, Chennai).

To convert Table 1 from unnormalized form into a normalized form, we have three different ways.

Solution 1

The column location in Table 1 is having more than one value. One way is to divide the column location into location1, location2 as shown in Table 2.

Drawback of Solution 1

The drawback of solution1 is that if a department is started in many places then more locations like location1, location2. . . . . . locationN has to be included in the table DEPARTMENT. Moreover some departments will be in only one place, in such a case more NULL values will be there in the table DEPARTMENT.

Table 2. DEPARTMENT (first normal form)

Department number	Department name	Location1	Location2
1	Nilgiris	Coimbatore	Chennai
2	Subiksha	Chennai	Tirunelveli
3	Krishna	Trichy	
4	Kannan	Coimbatore	

Solution 2

The second solution is to insert tuples for each location as shown in Table 3.

Drawback of Solution 2

The main draw back of solution 2 is that there are more repeating values, hence more number of rows in the Table 3.

Table 3. DEPARTMENT table

Department number	Department name	Location
1	Nilgiris	Coimbatore
1	Nilgiris	Chennai
2	Subiksha	Chennai
2	Subiksha	Tirunelveli
3	Krishna	Trichy
4	Kannan	Coimbatore

Solution 3

The third solution is to decompose the relation DEPARTMENT into two tables as shown in Tables 4 and 5.

Table 4.

Department number	Department name
1	Nilgiris
2	Subiksha
3	Krishna
4	Kannan

Table 5.

Department number	Location
1	Coimbatore
1	Chennai
2	Chennai
2	Tirunelveli
3	Trichy
4	Coimbatore

In the third solution we have divided the DEPARTMENT table into two tables. The process of splitting the table into more than one table is called normalization.

3.2. First Normal Form to Second Normal Form

Second Normal Form

A table is said to be in second normal form if it is in first normal form and all its nonkey attributes depend on all of the key (no partial dependencies). Consider the relation EMPLOYEE PROJECT, the relation EMPLOYEE PROJECT consists of the attributes EmployeeID, Employee name, Project ID, Project name, Total hours. Here total hours imply the time taken to complete the project.

Table 6. EMPLOYEE PROJECT

E_ID	E_NAME	P_ID	P_NAME	Total time

EMPLOYEE PROJECT

– E_ID stands for EmployeeID

– E_NAME stands for Employee name

– P_ID stands for ProjectID

– P_Name stands for Project name

– Total time is the time taken to complete the project

It is to be noted that the “Total Time” attribute depends on the nature of the project and the Employee. If the project is simple, then it can be completed easily and also if the employee is very talented then also the total time required to complete the project is less. Thus total time is determined by the EmployeeID and ProjectID. Clearly the relation EMPLOYEE PROJECT is not in second normal form. The reason is we have to key attributes E_ID which refers to EmployeeID and P_ID which refers to Project ID. Then each other attribute in the relation EMPLOYEE PROJECT should dependent on Employee ID alone, Project ID alone or both EmployeeID and ProjectID.

{E_ID, P_ID}→ “Total Time” is full functional dependency.

{E_ID, P_ID}→ E_NAME and {E_ID, P_ID}→ P_NAME are partial functional dependency.

All dependency must be full functional dependency in order to make this relation in 2NF.

The relation EMPLOYEE PROJECT can be transformed to second normal form by breaking the relation into three relations EMPLOYEE, PROJECT, and HOURS ASSIGNED.

EMPLOYEE (<u>E_ID</u>, E_NAME)

P_ID → P_NAME

PROJECT (<u>P_ID</u>, P_NAME)

E_ID → E_NAME

HOURS ASSIGNED (<u>E ID, P ID</u>, TOTALTIME)

{E_ID, P_ID}→ “Total Time”

In this relation the attribute TOTAL TIME fully depends on the compositekey E_ID and P_ID.

3.3 Second Normal Form to Third Normal Form

Third Normal Form

A table is in third normal form if it is in second normal form and contains no transitive dependencies (if A→B and B→C then A→C).

First let us consider a table HOSTEL which is in second normal form. The attributes of the table HOSTEL are Roll number, Building name, and Fee as shown in Table 7. The table HOSTEL stores information about building in which a student's room is located, and how much that student pays for the room. Here Student Roll number is the key for the table HOSTEL, since the other two columns depend on Student Roll number the table is in second normal form.

Table 7. HOSTEL

Roll number	Building	Fee
100	main	600
101	additional	500
102	new	650

The table HOSTEL is not in third normal form because of transitive dependency. Roll Number→Building, Building→Fee which implies that Roll Number Fees. Because of this transitive dependency, the table is not in third normal form. The table HOSTEL is prone to modification anomalies, since removing the Roll Number 101 from the table HOSTEL also deletes the fact that a room in Additional building costs Rs. 500. The modification anomaly is due to transitive dependency.

Solution to Transitive Dependency

The solution to transitive dependency is to split the HOSTEL table into two as shown in Tables 8 and 9. By splitting the table HOSTEL into two separate relations we can observe that the transitive dependency Roll Number→Fees is avoided hence the table is in third normal form.

Table 8.

Roll number	Building
100	main
101	additional
102	new

Table 9.

Building	Fee
main	600
additional	500
new	650