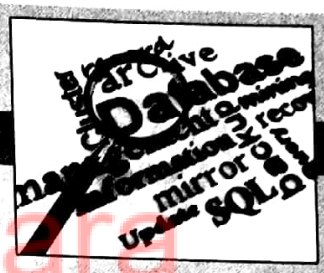
Chapter 8



NORMALIZATION FOR REFINEMENT OF DATA

CHAPTER OBJECTIVES

In this chapter you will learn:

- ❑ **The need of normalization.**
- ❑ **The concept of various dependencies in database like functional, fully functional, transitive, multi-value and projection-join etc.**
- ❑ **Basic normalization techniques proposed by Codd.**
- ❑ **Concept of overlapping of candidate keys.**
- ❑ **Need of BCNF.**
- ❑ **Advanced normalization techniques like fourth and fifth normal form.**

8.1 INTRODUCTION TO NORMALIZATION

Normalization is a design technique that is widely used in designing relational model. In order to design a relational model system, we have to decide logical structure of the database. Logical structure of the database is designed so that basic operations on the database like Insert, Update, Delete or Retrieve can be performed without any problems.

Normalization of data can be defined as a process during which redundant relation schemas are decomposed by breaking up their attributes into smaller relation schemas that possess desirable properties.

Normalization is a process of decomposing a larger table into smaller tables so that it satisfies series of tests. If the database satisfies the test, then database is considered normalized according to that test or rule or degree. There are five series of test that we apply on the database, so there are five degree or rules of normalization which are known as First Normal Form, Second Normal Form and so on. When a test fails, the relation violating that test must be decomposed into relations so that it individually meets the normalization tests.

8.1.1 Objectives of Normalization

The objectives of the normalization process are:

- To create a formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes.
- To obtain powerful relational retrieval algorithms based on a collection of primitive relational operators.
- To free relations from undesirable insertion, update and deletion anomalies.
- To reduce the need for restructuring the relations, as new data types are introduced.
- To carry out series of tests on individual relation schema so that the relational database can be normalized to some degree. When a test fails, the relation violating that test must be decomposed into relations that individually meet the normalization tests.

The entire normalization process is based upon the analysis of relations, their schema, their primary keys and their functional dependencies. Initially E.F. Codd proposed three normal forms known as first, second, and third normal form.

8.2 BASIC CONCEPTS ASSOCIATED WITH NORMAL FORMS

Normalization theory is built around the concept of Normal form.

A relation is said to be in a particular normal form if it satisfies a certain specified set of constraints. Each normal form has a set of constraints, if our data follows these constraints, then our database is in that particular normal form. There are number of normal forms like (1NF, 2NF, 3NF and etc). The idea behind these forms is to make 2NF better than 1NF and 3NF better then 2NF. 3NF has two versions, both given by Boyce and Codd. New version of 3NF is stronger than old form and named as BCNF to distinguish it from the old 3NF.

There are some special cases that are normalized by 4NF and 5NF, discussed in detail at later stages.

First N/F	Second N/F	Third N/F	Fourth N/F	Fifth N/F
Eliminate Repeating Groups	Non-key attribute Fully Functional Dependence on PK	Eliminate Transitive Dependence on PK	Remove Multi Value Dependeency	Projection Join Dependency

In order to understand normalization some basic concepts used during normalization are discussed here:

8.2.1 Functional Dependence (FD)

A functional dependency is an association between two attributes of the same relational database table. One of the attributes is called the determinant and the other attribute is called

the determined. For each value of the determinant, there is associated, one and only one value of the determined.

If A is the determinant and B is the determined then we say that *A functionally determines B* and graphically represent this as A ⟶B. The symbols A ⟶B can also be expressed as *B is functionally determined by* A.

Example

The following table illustrates A ⟶ B:

A	B
1	1
2	4
3	9
4	16
2	4
7	9

Since for each value of A there is associated one and only one value of B.

Example

The following table illustrates that A does not functionally determine B:

A	B
1	1
2	4
3	9
4	16
3	10

Since for A = 3 there is associated more than one value of B.

Functional dependency can also be defined as follows:

An attribute in a relational model is said to be functionally dependent on another attribute in the table if it can take only one value for a given value of the attribute upon which it is functionally dependent.

Example: Consider the database having following tables:

The Supplier table

SNo	Sname	Status	City
S1	Suneet	20	Qadian
S2	Ankit	10	Amritsar
S3	Amit	10	Amritsar

The Part table

PNo	Pname	Color	Weight	City
P1	Nut	Red	12	Qadian
P2	Bolt	Green	17	Amritsar
P3	Screw	Blue	17	Jalandhar
P4	Screw	Red	14	Qadian

The Shipment table

SNo	Pno	Qty
S1	P1	270
S1	P2	300
S1	P3	700
S2	P1	270
S2	P2	700
S3	P2	300

Here in Supplier table

Sno - Supplier number of supplier that is unique

Sname - Supplier name

City - City of the supplier

Status - Status of the city e.g. A grade cities may have status 10, B grade cities may have status 20 and so on.

Here, Sname is FD on Sno. Because, Sname can take only one value for the given value of Sno (e.g. S1) or in other words there must be one Sname for supplier number S1.

FD is represented as:

Sno ⟶Sname

FD is shown by ⟶ which means that Sname is functionally dependent on Sno.

Similarly, city and status are also FD on Sno, because for each value of Sno there will be only one city and status.

FD is represented as:

Sno ⟶ City

Sno ⟶Status

S. Sno⟶ S (Sname, City, Status)

Consider another database of shipment with following attributes:

Sno — Supplier number of the supplier

Pno — Part number supplied by supplier

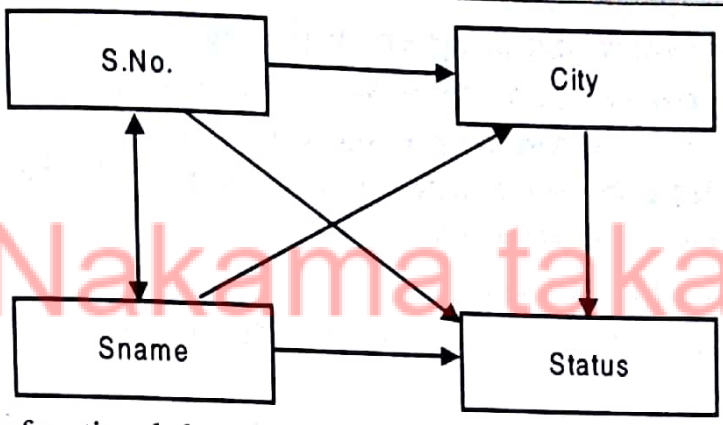
Qty — Quantity supplied by supplier for a particular Part no

In this case Qty is FD on combination of Sno, Pno because each combination of Sno and Pno results only for one Quantity.

SP (Sno, Pno) ⟶ SP.QTY

Dependency Diagrams

A dependency diagram consists of the attribute names and all functional dependencies in a given table. The dependency diagram of Supplier table is



Here, following functional dependencies exist in supplier table

Sno ⟶Sname

Sname ⟶Sno

Sno⟶City

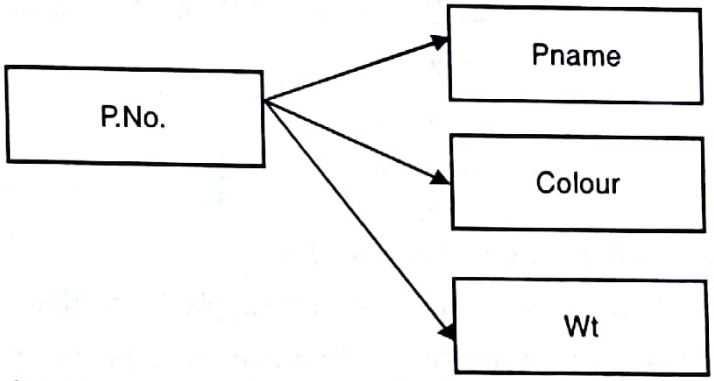
Sno ⟶Status

Sname⟶City

Sname ⟶Status

City ⟶Status

The FD diagram of relation P is

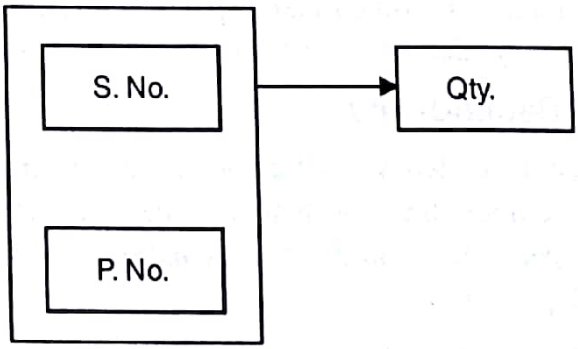


Here following functional dependencies exist in Part table:

Pno ⟶Pname

Pno ⟶Color

Pno ⟶Wt



The FD diagram of relation Shipment is

Here following functional dependencies exist in parts table

SP (Sno, Pno) ⟶ SP.QTY

8.2.2 Fully Functional Dependence (FFD)

Fully Functional Dependence (FFD) is defined, as Attribute Y is FFD on attribute X, if it is FD on X and not FD on any proper subset of X. For example, in relation Supplier, different cities may have the same status. It may be possible that cities like Amritsar, Jalandhar may have the same status 10.

So, the City is not FD on Status.

But, the combination of Sno, Status can give only one corresponding City ,because Sno is unique. Thus,

$$(\text{Sno, Status}) \longrightarrow \text{City}$$

It means city is FD on composite attribute (Sno,Status) however City is not fully functional dependent on this composite attribute, which is explained below:

$$\frac{(\text{Sno, Status})}{X} \longrightarrow \frac{\text{City}}{Y}$$

Here Y is FD on X, but X has two proper subsets Sno and Status; city is FD on one proper subset of X i.e. Sno

$$\text{Sno} \longrightarrow \text{City}$$

According to FFD definition Y must not be FD on any proper subset of X, but here City is FD in one subset of X i.e. Sno, so City is not FFD on (Sno, Status)

Consider another case of SP table:

Here, Qty is FD on combination of Sno, Pno.

$$\frac{(\text{Sno, Pno})}{X} \longrightarrow \frac{\text{Qty}}{Y}$$

Here, X has two proper subsets Sno and Pno

Qty is not FD on Sno, because one Sno can supply more than one quantity.

Qty is also not FD on Pno, because one Pno may be supplied many times by different suppliers with different or same quantities.

So, Qty is FFD on composite attribute of (Sno, Pno)→ Qty.

8.2.3 Other Functional Dependencies

There are some other types of functional dependencies, which play a vital role during the process of normalization of data.

Candidate Functional Dependency

A candidate functional dependency is a functional dependency that includes all attributes of the table. It should also be noted that a well-formed dependency diagram must have at least one candidate functional dependency, and that there can be more than one candidate functional dependency for a given dependency diagram.

Primary Functional Dependency

A primary functional dependency is a candidate functional dependency that is selected to determine the primary key. The determinant of the primary functional dependency is the

primary key of the relational database table. Each dependency diagram must have one and only one primary functional dependency. If a relational database table has only one candidate functional dependency, then it automatically becomes the primary functional dependency. Once the primary key has been determined, there will be three possible types of functional dependencies:

Description

A⟶B A key attribute functionally determines a non-key attribute.

A⟶B A non-key attribute functionally determines a non-key attribute.

A⟶B A non-key attribute functionally determines a key attribute.

A partial functional dependency is a functional dependency where the determinant consists of key attributes, but not the entire primary key, and the determined consists of non-key attributes.

A transitive functional dependency is a functional dependency where the determinant consists of non-key attributes and the determined also consists of non-key attributes.

A Boyce-Codd functional dependency is a functional dependency where the determinant consists of non-key attributes and the determined consists of key attributes.

A Muti-Value Dependency (MVD) occurs when two or more independent multi valued facts about the same attribute occur within the same table. It means that if in a relation R having A, B and C as attributes, B and C are muti-value facts about A, which is represented as A⟶⟶B and A⟶⟶C ,then muti value dependency exist only if B and C are independent of each other.

A Join Dependency exist if a relation R is equal to the join of the projections X, Y,... Z. where X,Y,...Z are projections of R.

8.2.4 Closure of set of dependencies

Let a relation *R* have some functional dependencies *F* specified. The *closure of F* (usually written as *F*+) is the set of all functional dependencies that may be logically derived from *F*. Often *F* is the set of most obvious and important functional dependencies and ***F*+**, the closure, is the set of all the functional dependencies including *F* and those that can be deduced from *F*. The closure is important and may, for example, be needed in finding one or more candidate keys of the relation.

For example, the *student* relation has the following functional dependencies

sno ⟶ *sname*

cno ⟶ *cname*

sno ⟶ *address*

cno ⟶ *instructor*

instructor ⟶ *office*

Let these dependencies be denoted by *F*. The closure of *F*, denoted by *F*+, includes *F* and all functional dependencies that are implied by *F*.

To determine *F*+, we need rules for deriving all functional dependencies that are implied by *F*. A set of rules that may be used to infer additional dependencies was proposed by

Armstrong in 1974. These rules (or axioms) are a complete set of rules in that all possible functional dependencies may be derived from them. The rules are:

1. *Reflexivity Rule* — If X is a set of attributes and Y is a subset of X, then $X \longrightarrow Y$ holds.

   The reflexivity rule is the most simple (almost trivial) rule. It states that each subset of X is functionally dependent on X. In other words Trivial dependence is defined as follows:

   Trivial functional dependency: A trivial functional dependency is a functional dependency of an attribute on a superset of itself.

   For example: {Employee ID, Employee Address} $\longrightarrow$ {Employee Address} is trivial, here {Employee Address} is a subset of {Employee ID, Employee Address}.

2. *Augmentation Rule* — If $X \longrightarrow Y$ holds and W is a set of attributes, then $WX \longrightarrow WY$ holds.

   The augmentation rule is also quite simple. It states that if Y is determined by X then a set of attributes W and Y together will be determined by W and X together. Note that we use the notation WX to mean the collection of all attributes in W and X and write WX rather than the more conventional (W, X) for convenience.

   For example: Rno $\longrightarrow$ Name; Class and Marks is a set of attributes and act as W. Then, {Rno, Class, Marks} $\longrightarrow$ {Name, Class, Marks}

3. *Transitivity Rule* — If $X \longrightarrow Y$ and $Y \longrightarrow Z$ hold, then $X \longrightarrow Z$ holds.

   The transitivity rule is perhaps the most important one. It states that if X functionally determines Y and Y functionally determines Z then X functionally determines Z.

   For example: If Rno $\longrightarrow$ City and City $\longrightarrow$ Status, then Rno $\longrightarrow$ Status should also hold true.

   These rules are called *Armstrong's Axioms*.

Further axioms may be derived from the above although the above three axioms are *sound and complete* in that they do not generate any incorrect functional dependencies (soundness) and they do generate all possible functional dependencies that can be inferred from F (completeness). The most important additional axioms are:

1. Union Rule — If $X \longrightarrow Y$ and $X \longrightarrow Z$ hold, then $X \longrightarrow YZ$ holds.

2. Decomposition Rule — If $X \longrightarrow YZ$ holds, then so do $X \longrightarrow Y$ and $X \longrightarrow Z$.

3. Pseudotransitivity Rule — If $X \longrightarrow Y$ and $WY \longrightarrow Z$ hold then so does $WX \longrightarrow Z$.

Based on the above axioms and the functional dependencies specified for relation *student*, we may write a large number of functional dependencies. Some of these are:

(sno, cno) $\longrightarrow$ *sno* (Rule 1)

(sno, cno) $\longrightarrow$ *cno* (Rule 1)

(sno, cno) $\longrightarrow$ *(sname, cname)* (Rule 2)

cno $\longrightarrow$ *office* (Rule 3)

sno $\longrightarrow$ *(sname, address)* (Union Rule)

etc.

Often a very large list of dependencies can be derived from a given set F since Rule 1 itself will lead to a large number of dependencies. Since we have seven attributes *(sno, sname, address, cno, cname, instructor, office)*, there are 128 (that is, 2^7) subsets of these attributes. These 128 subsets could form 128 values of X in functional dependencies of the type $X \longrightarrow Y$. Of course, each value of X will then be associated with a number of values for Y (Y being a subset of X) leading to several thousand dependencies. These large number of dependencies are not particularly helpful in achieving our aim of normalizing relations.

Although we could follow the present procedure and compute the closure of F to find all the functional dependencies, the computation requires exponential time and the list of dependencies is often very large and therefore not very useful. There are two possible approaches that can be taken to avoid dealing with the large number of dependencies in the closure. One is to deal with one attribute or a set of attributes at a time and find its closure (i.e. all functional dependencies relating to them). The aim of this exercise is to find what attributes depend on a given set of attributes and therefore ought to be together. The other approach is to find the *minimal covers*.

8.2.5 Minimal Functional Dependencies or Irreducibe Set of Dependencies

In discussing the concept of equivalent FDs, it is useful to define the concept of *minimal functional dependencies* or *minimal cover* which is useful in eliminating unnecessary functional dependencies so that only the minimal number of dependencies need to be enforced by the system. The concept of minimal cover of F is sometimes called *Irreducibe Set* of F.

A functional depending set S is irreducible if the set has three following properties:

1. Each right set of a functional dependency of S contains only one attribute.
2. Each left set of a functional dependency of S is irreducible. It means that reducing any one attribute from left set will change the content of S (S will lose some information).
3. Reducing any functional dependency will change the content of S.

Sets of functional dependencies with these properties are also called *canonical* or *minimal*.

8.3 FIRST NORMAL FORM

Definition of First Normal Form

A relation is said to be in First Normal Form (1NF) if and only if every entry of the relation (the intersection of a tuple and a column) has at most a single value. In other words "a relation is in First Normal Form if and only if all underlying domains contain atomic values or single value only."

The objective of normalizing a table is to remove its repeating groups and ensure that all entries of the resulting table have at most a single value. By simply removing the repeating groups of the unnormalized tables, they do not become relations automatically. Some further manipulations of the resulting table(s) may be necessary to ensure that they are indeed relations. Sometimes, during the process of designing a database it may be necessary to transform it into a relation, i.e. the intersection of a row and a column must have only one value.

For example, consider the STUDENT table shown next where one or more students may be assigned a common course. Notice that for each Course_Name every "row" of the table has more than one value under the columns Rollno, Name, System_Used, Hourly_Rate, and Total_Hrs.

Table entries that have more than one value called *multivalue* entries. Tables with *multivalue* entries are called *unnormalized* tables.

STUDENT (Unnormalized table)

Course_ Code	Course_ Name	Teacher_ Name	RollNo	Name	System_ Used	Hourly_ Rate	Total_ Hrs.
C1	Visual Basic	ABC	100	A1	P-I	20	7
			101	A2	P-II	30	3
			102	A3	Celeron	10	6
			103	A4	P-IV	40	1
C2	Oracle&Dev	DEF	100	A1	P-I	20	7
			104	A5	P-III	35	3
			105	A6	P-II	30	1
			101	A2	P-II	30	2
C3	C++	KJP	106	A7	P-IV	40	3
			107	A8	P-IV	40	2
			108	A9	P-I	20	1
C4	Java	Kumar	109	A10	Cyrix	20	2

Within an unnormalized table, we will call a repeating group an attribute or group of attributes that may have multivalue entries for single occurrences of the table identifier. The term refers to the attribute that allows us to distinguish the different rows of the unnormalized table. Using this terminology we can describe the STUDENT table shown above as an unnormalized table where attributes Rollno, Name, System_Used, Hourly_Used, and Total_Hrs are repeating groups. This type of table cannot be considered a relation because there are entries with more than one value. To be able to represent this table as a relation and to implement it in a RDBMS it is necessary to normalize the table. In other words we need to put the table in first normal form.

In general, there are two basic approaches to normalize tables.

First Approach: Flattening the table

The first approach known as "flattening the table" removes repeating groups by filling in the "missing" entries of each "incomplete row" of the table with copies of their corresponding non-repeating attributes. The following example illustrates this.

In the STUDENT table, for each individual Course, under the Rollno Name, System_Used, Hourly_Used, and Total_Hrs attributes, there is more than one value per entry. To normalize this table, we just fill in the remaining entries by copying the corresponding information from the non-repeating attributes. For instance, for the row that contains the course Visual Basic, we fill in the remaining "blank" entries by copying the values of the Course_Code,

Course_Name and Teacher_Name columns. This row has now a single value in each of its entries. We have repeated a similar process for the students of the remaining two courses. The normalized representation of the STUDENT table is:

STUDENT (Normalized Relation)

Course_ Code	Course_ Name	Teacher_ Name	RollNo	Name	System_ Used	Hourly_ Rate	Total_ Hrs.
C1	Visual Basic	ABC	100	A1	P-I	20	7
C1	Visual Basic	ABC	101	A2	P-II	30	3
C1	Visual Basic	ABC	102	A3	Celeron	10	6
C1	Visual Basic	ABC	103	A4	P-IV	40	1
C2	Oracle&Dev	DEF	100	A1	P-I	20	7
C2	Oracle&Dev	DEF	104	A5	P-III	35	3
C2	Oracle&Dev	DEF	105	A6	P-II	30	1
C2	Oracle&Dev	DEF	101	A2	P-II	30	2
C3	C++	KJP	106	A7	P-IV	40	3
C3	C++	KJP	107	A8	P-IV	40	2
C3	C++	KJP	108	A9	P-I	20	1
C4	Java	Kumar	109	A10	Cyrix	20	2

In normalized STUDENT table the attribute Course_Code no longer identifies uniquely any row. Thus, a suitable primary key for this table is the composite key (Course_Code, Rollno).

Second Approach: Decomposition of the table

The second approach for normalizing a table requires that the table be decomposed into two new tables that will replace the original table. Decomposition of a relation involves separating the attributes of the relation to create the schemas of two new relations. However, before decomposing the original table it is necessary to identify an attribute or a set of its attributes that can be used as table identifiers.

Rule of decomposition

- ✦ One of the two tables contains the table identifier of the original table and all the non-repeating attributes.
- ✦ The other table contains a copy of the table identifier and all the repeating attributes.

To transform these tables in to relations, it may be necessary to identify a PK for each table. The tuples of the new relations are the projection of the original relation into their respective schemes. The following example illustrates this second approach for normalizing tables.

To normalize the STUDENT table we need to replace it by two new tables. The first table COURSE contains the table identifier and the non-repeating groups.These attributes are Course_Code (the table identifier), Course_Name, and Teacher_Name.

The second table contains the table identifier and all the repeating groups. Therefore, the attributes of COURSE_STUDENT table are Course_Code, Rollno, Name, System_Used, Hourly_Rate and Total_Hrs.

COURSE

Course_Code	Course_Name	Teacher_Name
C1	Visual Basic	ABC
C2	Oracle&Dev	DEF
C3	C++	KJP
C4	Java	Kumar

COURSE_STUDENT

Course_ Code	Rollno	Name	System_Used	Hourly_Rate	Total_Hrs
C1	100	A1	P-I	20	7
C1	101	A2	P-II	30	3
C1	102	A3	Celeron	10	6
C1	103	A4	P-IV	40	1
C2	100	A1	P-I	20	7
C2	104	A5	P-III	35	3
C2	105	A6	P-II	30	1
C2	101	A2	P-II	30	2
C3	106	A7	P-IV	40	3
C3	107	A8	P-IV	40	2
C3	108	A9	P-I	20	1
C4	109	A10	Cyrix	20	2

To transform the latter table into a relation, it is necessary to assign it a PK. These two new 1NF relations are shown above. Notice that for the COURSE_STUDENT table the composite attribute (Course_Code, Rollno) is an appropriate PK. At this point the reader may ask which of these two approaches is better to use. Actually both approaches are correct because they transform any unnormalized table into a 1NF relation. However the second approach is more efficient because the relations produced are less redundant. In addition as we will see in the next section, the single table obtained using the first approach will eventually broken into the same two tables obtained in the second approach.

Anomalies in 1NF Relations (Considering STUDENT table)

Redundancies in 1NF relations lead to a variety of data anomalies. Data anomalies are divided into three general categories: insertion, deletion and update anomalies.

They are named respectively after the relational operations of Insert, Delete, and Update because it is during the application of these operations that a relation may experience anomalies.

Insert Anomalies

We cannot insert the information about the student until he/she joins any course e.g as shown in the above database we cannot store the information about the rollno 110 until he joins any course, similarly we are unable to store the information about the course until there is a student who enrolls in to that course. For example, we cannot store that C1 course is of Visual Basic until at least one student joins that course.

These anomalies occur because Course_Code, Rollno is the composite primary key and we cannot insert null in any of these two attributes for a record. So, in order to store a record we must know the Course_Code and the Rollno of student who join the course.

Update Anomalies

This relation is also susceptible to update anomalies because the course in which a student studies may appear many times in the table. It is this redundancy of information that causes the anomaly because if a teacher moves to another course, we are now faced with two problems; we either search the entire table looking for that teacher and update his or her Course_Code value or we miss one or more tuples of that STUDENT and end up with an inconsistent database. For small tables, this type of anomaly may not seem to be much of a problem, but it is easy to imagine situations where there may be thousands of tuples that experience similar anomaly.

Let us consider, a situation in which we have to change the teacher for a particular course e.g. we have to update the teacher for Course_Code C1 then, we have to modify multiple records which is equal to the number of the students for that particular Course_Code e.g. C1. This will cause the problem of inconsistency. Suppose we change the name of teacher three times and forget to change the name at one place then data becomes inconsistent.

Delete Anomalies

This relation experiences deletion anomalies whenever we delete the last tuple of a particular student. In this case, we not only delete the course information that connects that student to a particular course, but also lose other information about the system on which this student works.

Let us consider, the case where we have to delete the information of student having Rollno 109, then we also lose the information about course_code C4 i.e now we are unable to identify the name of course as well as the corresponding teacher for that course. Suppose, we have to delete the information of Java course we also lose information about the student Kumar.

Practice Session

1. In the EMPLOYEE table shown below, identify the table identifier, all repeating and non repeating attributes. Flatten the table and state if the resulting table is a relation. If not, how can you make it a relation?

EMPLOYEE

ID	Last_ Name	Department	Dependent_ Name	Dependent_ DOB	Dependent_ Sex	Dependent_ ID
100	Kumar	CS	Mary	01/12/60	F	800980432
			Cindy	04/24/67	F	973637262
			John	07/12/68	M	992776631
101	Singh	VP	Fern	03/28/62	F	790902462
			Victoria	11/12/84	F	800234979
102	Sharma	Sales	Sadie	08/31/67	F	970073473
103	Miller	Sales	Sallie	09/21/47	F	890721289
104	Kroeger	MIS	Jonathan	08/17/87	M	943632772

The table identifier is the attribute ID. The nonrepeating attributes are : ID, Last-name and Department. The repeating attributes are: Dependent_Name, Dependent_DOB, Dependent_Sex and Dependent-ID.

To flatten the table we have to fill in all the entries of the table by copying the information of the corresponding nonrepeating attributes. The normalized table looks like this:

ID	Last_ Name	Department	Dependent_ Name	Dependent_ DOB	Dependent_ Sex	Dependent_ ID
100	Kumar	CS	Mary	01/12/60	F	800980432
100	Kumar	CS	Cindy	04/24/67	F	973637262
100	Kumar	CS	John	07/12/68	M	992776631
101	Singh	VP	Fern	03/28/62	F	790902462
101	Singh	VP	Victoria	11/12/84	F	800234979
102	Sharma	Sales	Sadie	08/31/67	F	970073473
103	Miller	Sales	Sallie	09/21/47	F	890721289
104	Kroeger	MIS	Jonathan	08/17/87	M	943632772

This 'flat' table is not a relation because it does not have a primary key. Notice, for instance, that ID (the table identifier) no longer identifies any of the first three rows of this table. To transform this table into a relation we need to identify a suitable primary key for the relation. The composite key (ID, Dependent-ID) seems to be a suitable primary key.

2. Normalize the table of the previous example by creating two new relations. The table is reproduced below for the convenience of the reader.

ID	Last_ Name	Department	Dependent_ Name	Dependent_ DOB	Dependent_ Sex	Dependent_ ID
200	Kumar	CS	Mary	01/12/60	F	800980432
			Cindy	04/24/67	F	973637262
			John	07/12/68	M	992776631
201	Singh	VP	Fern	03/28/62	F	790902462
			Victoria	11/12/84	F	800234979
202	Sharma	Sales	Sadie	08/31/67	F	970073473
203	Miller	Sales	Sallie	09/21/47	F	890721289
204	Kroeger	MIS	Jonathan	08/17/87	M	943632772

To normalize this table we need to create two new relations. The attributes of the first relation are the table identifier and all the non-repeating attributes. The attributes of the second table are the table identifier and all the repeating attributes. The schemes of these two relations are shown below. Observe that the attribute ID (of Employee) has been renamed in the Dependent relation.

Employee (ID, Last-name, Department)

Dependent (Emp-ID, Dependent-ID, Dependent_Name, Delendent_DOB, Dependent_Sex)

The corresponding instance of these two relations are shown next. Notice that duplicate rows have been deleted to comply with the definition of a relation. PKs are underlined.

Employee

Id	Last_Name	Department
200	Kumar	CS
201	Singh	VP
202	Sharma	Sales
203	Miller	Sales
204	Kroeger	MIS

Dependent

Emp. Id	Department Id	Dependent_ Name	Dependent_ DOB	Dependent_ Sex
200	800980432	Mary	01/12/60	F
200	973637262	Cindy	04/24/67	F
200	992776631	John	07/12/68	M
201	790902462	Fern	03/28/62	F
201	800234979	Victoria	11/12/84	F
202	970073473	Sadie	08/31/67	F
203	890721289	Sallie	09/21/47	F
204	943632773	Jonathan	08/17/87	M

8.4 SECOND NORMAL FORM

Definition of second normal form is:

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

A resultant database of first normal form COURSE_CODE does not satisfy above rule, because non-key attributes Name, System_Used and Hourly_Rate are not fully dependent on the primary key (Course_Code, Rollno) because Name, System_Used and Hourly_Rate are functional dependent on Rollno and Rollno is a subset of the primary key so it does not hold the law of fully functional dependence as shown in figure 8.1. In order to convert COURSE_CODE database into second normal form following rule is used.

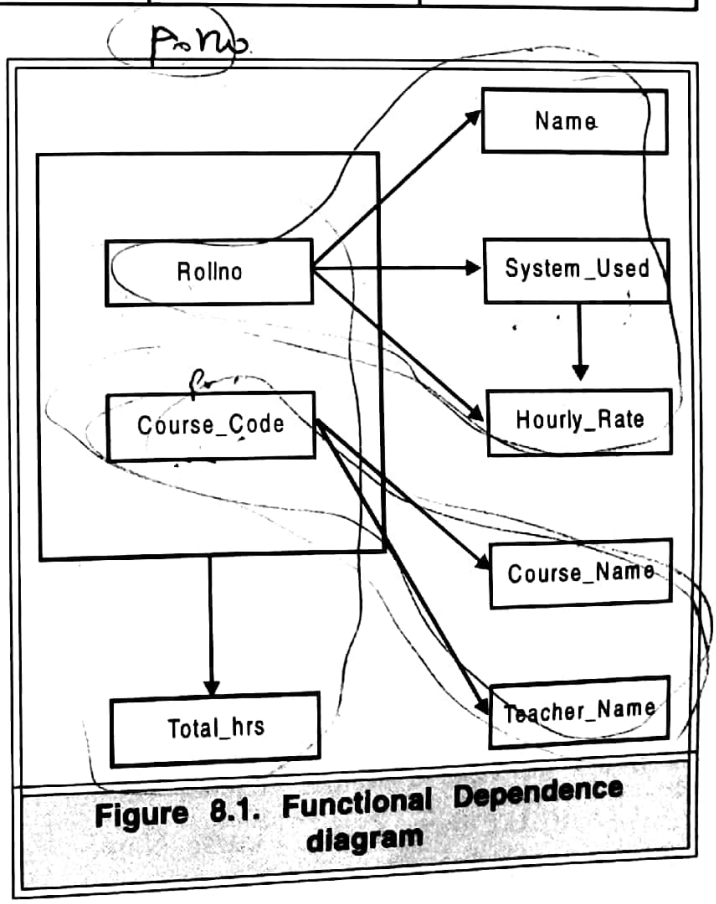


Figure 8.1. Functional Dependence diagram

Rule to convert First Normal Form to Second Normal Form

Consider a relation where a primary key consists of attributes A and B. These two attributes determine all other attributes. Attribute C is fully dependent on the key. Attribute D is partially dependent on the key because we only need attribute A to functionally determine it. Attributes C and D are nonprime or non-key attributes. Here the rule is to replace the original relation by two new relations as shown in figure 8.2. The first new relation has three attributes: A, B and C. The primary key of this relation is (A,B) i.e. the primary key of the original relation. The second relation has A and D as its only two attributes. Observe that attribute A has been designated, as the primary key of the second relation and that attribute D is now fully dependent on the key.

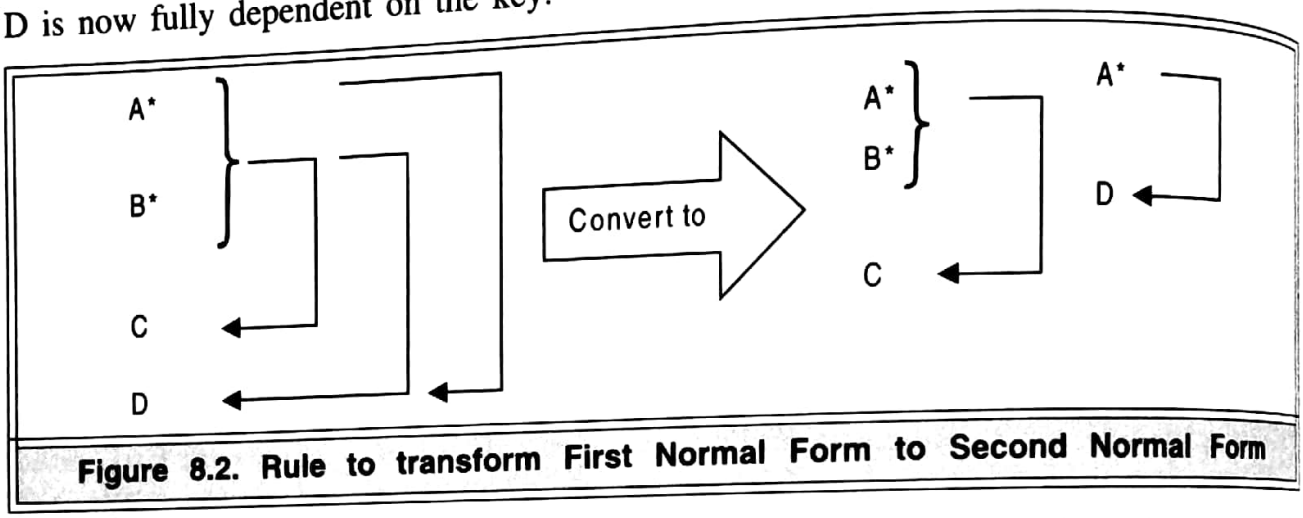


Figure 8.2. Rule to transform First Normal Form to Second Normal Form

Although the figure 8.2 only shows four attributes, we can generalize this procedure for any relation that we need to transform to 2NF if we assume that C stands for the collection of attributes that are fully dependent on the key and D stands for the collection of attributes that are partially dependent on the key. In our case study A stands for Course_Code and B for Rollno. Total_Hrs acts as C and (Name, System_Used, Hourly_Rate) acts as D which depends on Rollno; (Course_Name, Teacher_Name) also acts as D which depends on only Course_Code.

Example : Transformation of STUDENT(Course_Code, Course_Name,Teacher_Name, Rollno, Name, System_Used Hourly_Used,Total-Hours) into a 2NF

The above rule calls for breaking this relation into three new relations. The primary key of STUDENT (Course_Code, Rollno) and the remaining attributes of this relation that fully depends on this composite key is Total_Hours. The scheme of this new relation that we have named HOURS_ASSIGNED is as follows:

HOURS_ASSIGNED (Course_Code, Rollno, Total_Hours)

The second relation contains Rollno as its primary key, because Rollno fully determines the Name, System_Used, and Hourly_Rate. The scheme of this relation is as follows:

STUDENT_SYSTEM_CHARGE(Rollno, Name, System_Used, Hourly_Rate)

The third relation contains Course_Code as its primary key, because Course_Code fully determines the Course_Name, Teacher_Name. The scheme of this relation is as follows:

COURSE(Course_Code, Course_Name, Teacher_Name)

HOURS_ASSIGNED

Course_Code	RollNo	Total__Hrs
C1	100	7
C1	101	3
C1	102	6
C1	103	1
C2	100	7
C2	104	3
C2	105	1
C2	101	2
C3	106	3
C3	107	2
C3	108	1
C4	109	2

STUDENT_SYSTEM_CHARGE

Rollno	Name	System_Used	Hourly_ Rate
100	A1	P-I	20
101	A2	P-II	30
102	A3	Celeron	10
103	A4	P-IV	40
100	A1	P-I	20
104	A5	P-III	35
105	A6	P-II	30
101	A2	P-II	30
106	A7	P-IV	40
107	A8	P-IV	40

COURSE

Course_Code	Course_Name	Teacher_Name
C1	Visual Basic	ABC
C2	Oracle&Dev	DEF
C3	C++	KJP
C4	Java	Kumar

Use of 2NF to remove anomalies of First Normal form

Insert Anomalies

It is now possible to insert the information about the student who does not join any course e.g. we can store the information about the Rollno 110 who does not join any course in STUDENT_SYSTEM_CHARGE database as shown above. Similarly now we are able to store the information about the course which has no enrolled student e.g we can store that C1 course is of Visual Basic in COURSE database. It does not matter, whether it has an enrolled student or not.

Update Anomalies

In the revised structure, it is possible to change the teacher for a particular course in the COURSE database through a single modification.

Delete Anomalies

In the revised structure, we can delete the information of student having Rollno 109 without losing the information about his course i.e. C4.

Data Anomalies in 2NF Relations

Relations in 2NF are still subject to data anomalies. For the sake of explanation, let us assume that the system on which a student works functionally determines the hourly rate charged from the student. That is, System_Used → Hourly_Rate. This fact was not considered in the explanation of the previous normal form but it is not an unrealistic situation. If this functional dependence exists then the following anomalies will occur:

Insertion anomalies

Insertion anomalies occur in the STUDENT_SYSTEM_CHARGE relation. For example, consider a situation where we would like to set in advance the rate to be charged from the students for a particular system. We cannot insert this information until there is a student assigned to that type of system. Suppose we want to store the hourly_rate of laptop we cannot insert it until some student uses that type of system because rollno is primary key and we cannot insert null into it. Notice that the rate that is charged from student for a particular system is independent of whether or not any student uses that system or not.

Update anomalies

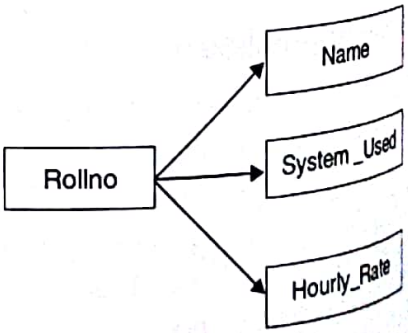
Update anomalies will also occur in the STUDENT_SYSTEM_CHARGE relation because there may be several students who are working on the same type of the system. If the Hourly_Rate for that particular system changes, we need to make sure that the corresponding rate is changed for all students who work on that type of system. Otherwise the database may end up in an inconsistent state. In case of any updation on hourly_rate of any particular type of system we need to make multiple updations which are equal to the number of students using that type of system.

Delete anomalies

The STUDENT_SYSTEM_CHARGE relation is also susceptible to deletion anomalies. This type of anomaly occurs whenever we delete the tuple of a student who happens to be the only student left who is working on a particular system. In this case we will also lose the information about the rate that we charge for that particular system.

Solution of above problems

The anomalies discussed above occur due to transitive dependence of Hourly_Rate on the primary key (Rollno) of STUDENT_SYSTEM_CHARGE database as shown in Functional dependence diagram below:



The solution of all above anomalies is provided by the third normal form, which deals with the problem of transitive dependence.

Practice Session:

1. Consider the relation scheme and FD shown below. What is the highest normal form of this relation ? Transform this relation to its next higher form. Can the information of the given relation be recovered ? What operation is necessary to recover it ?

Programmer-Task (Programme-ID, Programming-Package-ID,

Programming-Package-Name, Total-Hours-Worked-on-Package).

Programming-Package-ID ⟶ Programming-Package-Name

The highest form of this relation is 1NF because there are partial dependences on the composite key. Consider for example, Programming-Package-ID ⟶ Programming-Package-Name.

The next highest form of this relation is to 2NF. To transform it we can use Figure 8.2 as a guide. According to this figure, we need to create two new relations. The first relation has as its key the primary key of the given relation: Programmer-ID, Programming-Package-ID. The scheme of this first relation is:

Programmer-Activity (Programmer-ID, Programmer-Package-ID,

Total-Hours-Worked-on-Package)

The second relation has as its primary key the attribute: Programming-Package-ID. The scheme of this relation is:

Package-Info (Programming-Package-ID, Programming-Package-Name)

The information of the original relation can be recovered by means of a join operation on the common attribute: Programming-Package-ID.

8.5 THIRD NORMAL FORM

A relation R is in Third Normal Form (3NF) if and only if the following conditions are satisfied simultaneously:

1. R is already in 2NF
2. No nonprime attribute is transitively dependent on the key.

Another way of expressing the conditions for Third Normal Form is as follows:

1. R is already in 2NF
2. No nonprime attribute functionally determines any other nonprime attribute.

These two sets of conditions are equivalent.

As these two definitions of 3NF imply, the objective of transforming relations into 3NF is to remove all transitive dependencies. So, first we are going to explain the concept of transitive dependency.

Transitive Dependencies

Assume that A, B and C are the set of attributes of a relation R and following functional dependencies are satisfied simultaneously: $A \longrightarrow B$, $B \not\longrightarrow A$ (B not functionally depends A), $B \longrightarrow C$, $A \longrightarrow C$ and $C \not\longrightarrow A$ (C not functionally depends A). Observe that $C \longrightarrow B$ is neither prohibited nor required. If all these conditions are true, we will say that attribute C is transitively dependent on attribute A. It should be clear that these functional dependencies determine the conditions for having a transitive dependency of attribute C on A. If any of these functional dependencies are not satisfied then attribute C is not transitively dependent on attribute A.

The figure 8.3 shown below summarizes these conditions. In this diagram the arrows are equivalent to the symbol "$\longrightarrow$" that we use for denoting functional dependencies. Notice that the functional dependency $A \longrightarrow C$ may not be explicitly indicated but it holds true due to the Transitivity axiom. The requirements that $B \not\longrightarrow A$ (B not functionally depends A) and $C \not\longrightarrow A$ (C not functionally depends A) are necessary to ensure that attributes A and B are nonprime attributes.

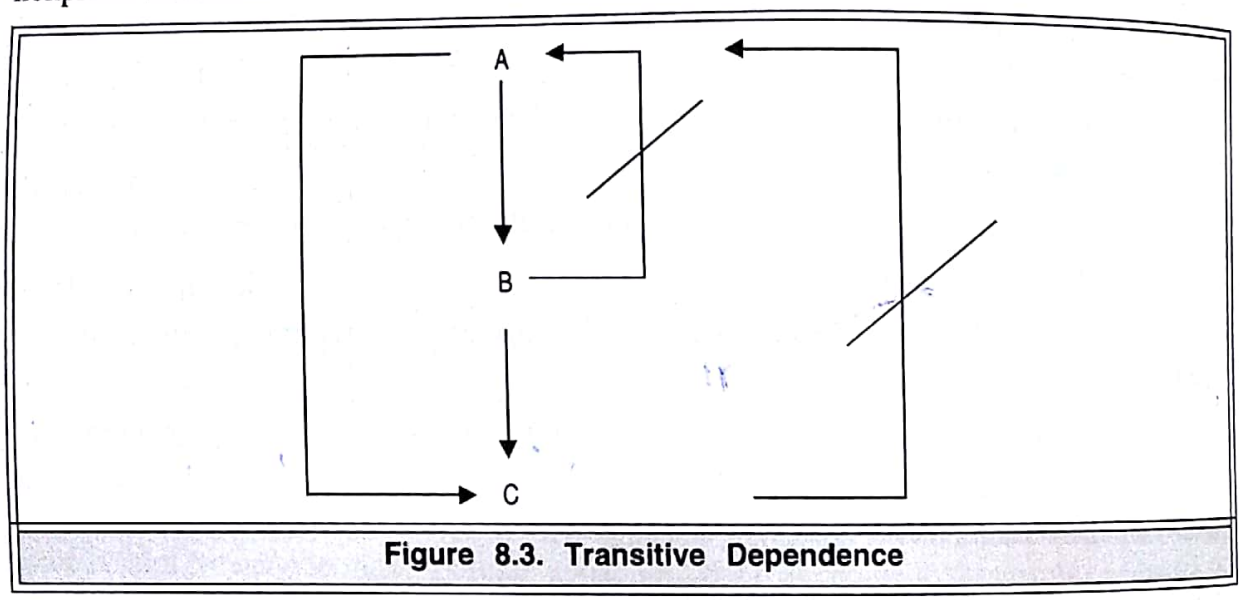


Figure 8.3. Transitive Dependence

Rule to transform a relation into Third Normal Form

To transform a 2NF relation into a 3NF we will follow the approach indicated by figure 8.4. In this figure assume that any FD not implicitly indicated does not hold. An asterisk indicates the key attribute and the arrows denote functional dependencies. The dashed line

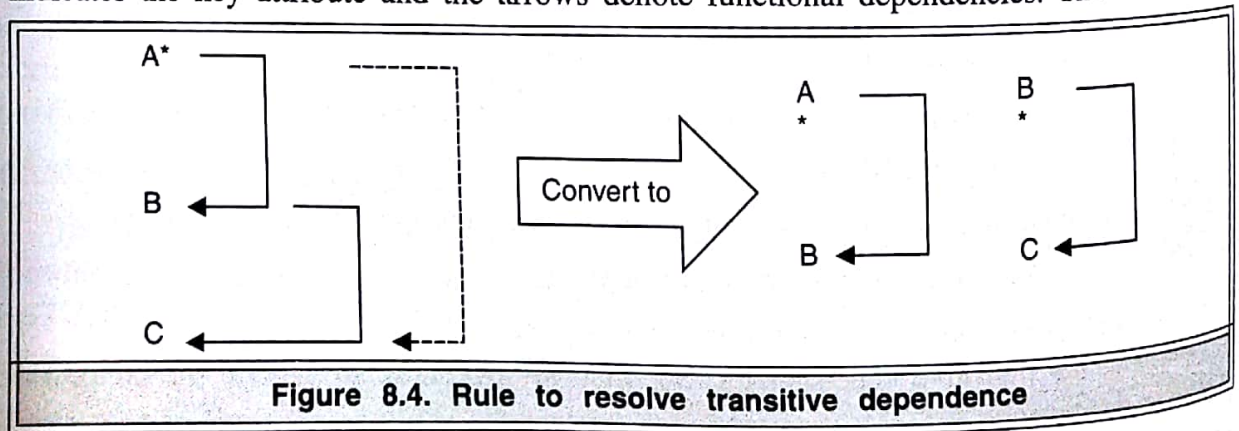


Figure 8.4. Rule to resolve transitive dependence

indicates that the FD A ⟶ C may not be explicitly given but it is always present because it can be derived using the inference axioms.

The above rule, simply states that if B depends on A and C depends on B, then it can be decomposed to (A, B) and (B, C).

Conversion of STUDENT_SYSTEM_CHARGE (Rollno, Name, System_Used, Hourly_Rate) to Third Normal Form:

The relation STUDENT_SYSTEM_CHARGE is not in 3NF because there is a transitive dependency of a nonprime attribute on the primary key of the relation. In this case, the nonprime attribute Hourly_Rate is transitively dependent on the primary key Rollno through the functional dependency System_Used ⟶Hourly_Rate. Notice that all other conditions required by the definition are met by this set of FDs. In particular, we have that Rollno ⟶System_Used and System_Used ⟶Hourly_Rate.

To transform this relation into a 3NF relation, it is necessary to remove any transitive dependency of a nonprime attribute on the key. According to the figure 8.3 it is necessary to create two new relations.

The scheme of the first relation is:

STUDENT_SYSTEM (Rollno,Name,System_Used).

The scheme of the second relation is:

CHARGES (System_Used,Hourly_Rate).

Observe that in the second relation, the System_Used attribute has been made the primary key of the relation as required by the diagram.

Using the second definition of 3NF, we can determine that the STUDENT_SYSTEM_CHARGE relation is not in 3NF by noticing that System_Used ⟶ Hourly_Rate, here both attributes are nonprime. To transform this relation to 3NF we use the same general procedure of above figure.

STUDENT_STSTEM

Rollno	Name	System_Used
100	A1	P-I
101	A2	P-II
102	A3	Celeron
103	A4	P-IV
100	A1	P-I
104	A5	P-III
105	A6	P-II
101	A2	P-II
106	A7	P-IV
107	A8	P-IV
108	A9	P-I
109	A10	Cyrix

CHARGES

System_Used	Hourly_ Rate
P-I	20
P-II	30
Celeron	10
P-IV	40
P-III	35
Cyrix	20

Removal of anomalies of Second Normal form

Insertion anomalies

In the revised structure of STUDENT_SYSTEM and CHARGES, it is possible to insert in advance the rate to be charged from the students for a particular system (in CHARGES database). It means that we can insert that hourly rate of laptop in CHARGES table independent from whether it is used by any student or not.

Update anomalies

If the Hourly_Rate for a particular system changes, we need only to change a single record in CHARGES database for that particular system.

Delete anomalies

We delete the tuple of a student who happens to be the only student left who is working on a particular system without losing the information about the rate that we charge for that particular system (We can get this information from CHARGES database).

Data Anomalies in 3NF Relations

The 3NF helped us to get rid of the data anomalies caused either by transitive dependencies on the PK or by dependencies of a nonprime attribute on another nonprime attribute. However, relations in 3NF are still susceptible to data anomalies particularly when the relations have two overlapping candidate keys or when a nonprime attribute functionally determines a prime attribute. The following example will illustrate this.

Example

Consider the Manufacturer relation shown below where each manufacturer has a unique ID and name. Manufacturers produce items (identified by their unique item numbers) in the amounts indicated. Manufacturers may produce more than one item and different manufacturers may produce the same items.

Manufacturer (Id_No, Name, Item_No, Quantity)

Manufacturer

Id_No	Name	Item_No	Quantity
M101	Electronics USA	H3772	1000
M101	Electronics USA	J08732	700
M101	Electronics USA	Y23490	200
M322	Electronics-R-Us	H3772	900

This Manufacturer relation has two candidate keys: (ID, Item_No) and (Name, Item_No) that overlap on the attribute Item_No. The relation is in 3NF because there is only one nonprime attribute and therefore it is impossible that this attribute can determine another nonprime attribute.

The relation Manufacturer is susceptible to update anomalies. Consider for example the case in which one of the manufacturers changes its name. If the value of this attribute is not changed in all of the corresponding tuples there is the possibility of having an inconsistent database.

We can take another case of Supplier-Part table having following attributes:

(Sno, Sname, Pno, Qty)

Here, let us suppose that Sname (supplier name) is unique for each Sno (supplier number) as shown below:

Sno	Sname	Pno	Qty
S1	Rahat	P1	300
S2	Raju	P2	200
S1	Rahat	P3	100
S2	Raju	P1	200

This Supplier-Part relation has two candidate keys: (Sno, Pno) and (Sname, Pno) that overlap on the attribute Pno. The relation is in 3NF because there is only one nonprime attribute and therefore it is impossible that this attribute can determine another nonprime attribute.

The relation is susceptible to update anomalies. Consider for example, the case in which one of the suppliers changes its name, then we have to make multiple changes which is equal to the number part supplied by that particular supplier. This relation is in second and third normal form because it has only one non-key attribute Qty which is FFD on Primary key. Sname cannot be called as non key attribute because it can participate in the primary key in case of (Sname, Pno) primary key. This relation is normalized with the help of Boyce-Codd normal form.

8.6 BOYCE-CODD NORMAL FORM

To eliminate these anomalies in 3NF relations, it is necessary to carry out the normalization process to the next higher step, the Boyce-Codd Normal Form.

BCNF is simply a stronger definition of 3NF. BCNF makes no explicit reference to first and second normal form as such, nor the concept of full and transitive dependence.

BCNF states that:

- A relation R is in Boyce/Codd N/F (BCNF) if and only if every determinant is a candidate key. Here, determinant is a simple attribute or composite attribute on which some other attribute is fully functionally dependent.

  For example: Qty is FFD on (Sno, Pno)

  (Sno, Pno) ⟶ Qty, here

  (Sno, Pno) is a composite determinant.

  Sno ⟶ Sname

  Here, Sno is simple attribute determinat.

Similarities between 3NF and BCNF

The relations which are achieved after application of 3NF can also be achieved by BCNF. For example, relation COURSE_STUDENT and STUDENT_SYSTEM_CHARGE which are not in 3NF are also not in BCNF.

COURSE_STUDENT

(Course_Code,Rollno,Name,System_Used,Hourly_Rate,Total_Hours)

Here, (Course_Code, Rollno) ⟶ Total_Hours

Rollno⟶ Name | System_Used | Hourly_Rate

Here, Rollno is a determinant but not candidate key (candidate key is course_coode, rollno) so relation COURSE_STUDENT is not in BCNF.

In relation STUDENT_SYSTEM_CHARGE

(Rollno,Name,System_Used,Hourly_Rate)

Rollno ⟶ Name | System_Used | Hourly_Rate

System_Used ⟶ Hourly_Rate

Here, System_Used is also a determinant but it is not unique, so relation STUDENT_SYSTEM_CHARGE is not in BCNF.

Now, Consider COURSE, HOUR_ASSIGNED, STUDENT_SYSTEM, CHARGE relations discussed earlier which are in 3NF.

COURSE (Course_Code,Course_Name,Teacher_Name)

HOUR_ASSIGNED (Course_Code,Rollno,Total_Hours)

STUDENT_SYSTEM (Rollno,Name,System_Used)

CHARGE(System_Used,Hourly_Rate)

These relations are also in BCNF,because

Course_Code ⟶ Course_Name | Teacher_Name (In relation COURSE)

(Course_Code,Rollno) ⟶ Total_Hours (In relation HOUR_ASSIGNED)

Rollno ⟶ Name | System_Used (In relation STUDENT_SYSTEM)

System_Used ⟶ Hourly_Rate (In relation CHARGES)

Here, each determinant is unique in its corresponding relation.

In conclusion, we can say that in these relations which have only single candidate key can be normalized both with 3NF and BCNF without any problem.

Differences in 3NF and BCNF

In order to show the difference between 3NF and BCNF, relations having overlapping of candidate keys are considered in detail.

Overlapping of Candidate keys

Two candidate keys overlap if they involve two or more attributes each and have an attribute in common.

For example, in Manufacturer relation:

Manufacturer (Id_no,Name, Item_No, Quantity)

Manufacturer

Id_No	Name	Item_No	Quantity
M101	Electronics USA	H3772	1000
M101	Electronics USA	J08732	700
M101	Electronics USA	Y23490	200
M322	Electronics-R-Us	H3772	900

Here, Name is considered unique for each Id_no.

FD of above relation is

(Id_no, Item_No) ⟶ Quantity

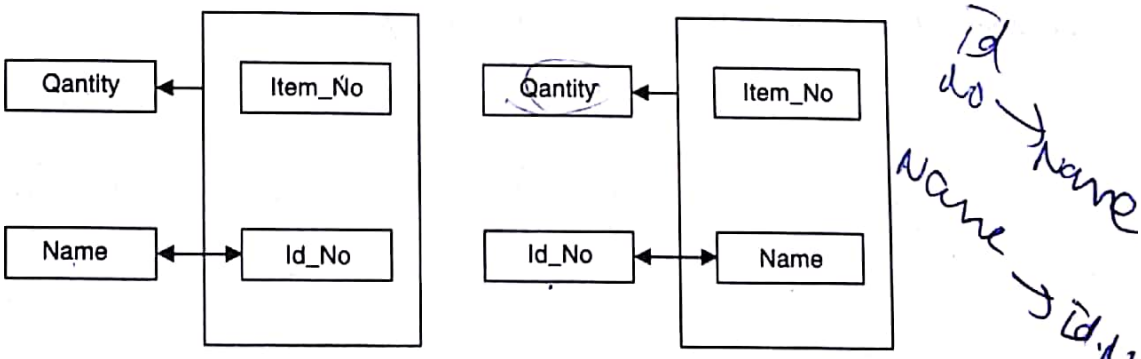
(Name,Item_No) ⟶ Quantity

Id_No ⟶ Name

Name ⟶ Id_No

This relation has two overlapping candidate keys,because there are two composite candidate keys (Id_no, Item_No) and (Name,Item_No) out of which Item_No is common attribute in both the candidate keys, so this is a case of overlapping of candidate keys.

Possible FD diagram of this case is:



Here, both the relations are in 3NF, because every non-key attribute is non-transitively fully functional dependent on the primary key.

In above relation, there is only one non-key attribute i.e. Quantity and it is FFD and non transitively dependent on the primary key.

Name, Id_No are not non-key attributes because they can participate into the primary key as shown in FD diagram.

But, Manufacturer relation is not in BCNF because this relation has four determinants

(Id_no, Item_No) [Qty depends on this combination]

(Name,Item_No) [Qty depends on this combination]

Id_No [Name depends on Id_No]

Name [Id_No depends on Name]

Out of these four determinants two determinants (Id_no, Item_No) and (Name,Item_No) are unique but Id_No and Name determinants are not candidate keys.

In order to make this relation in BCNF we non-loss decompose this relation in two projections ID_NAME (Id_no, Name) and ID_QTY (Id_no, Item_No, Quantity).

ID_NAME relation has two determinants Id_no, Name and both are unique.

ID_QTY has one determinant (Id_no,Item_No) and is also unique.

These two relations ID_NAME and ID_QTY removes all anomalies of Manufacturer relation.

Another Example

For example, consider a relation

SSP (Sno, Sname, Pno, Qty)

Here, Sname is considered unique for each Sno.

FD of above relation is

(Sno,Pno) ⟶ Qty

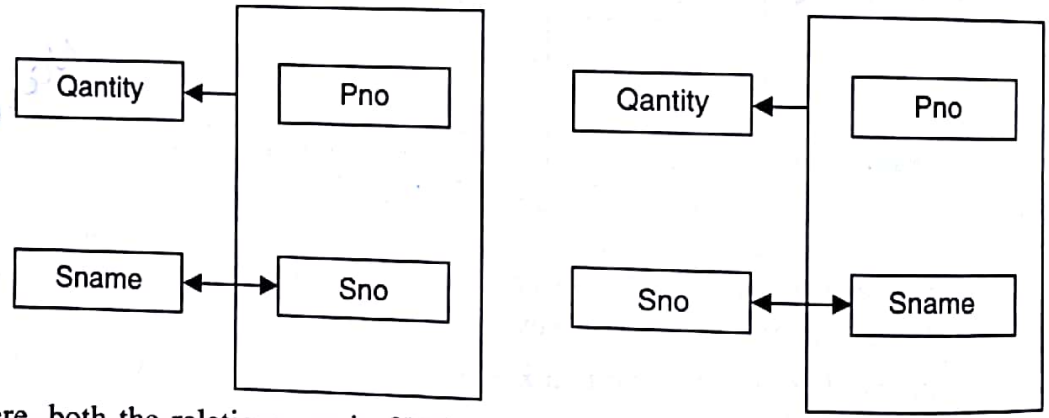
(Sname,Pno) ⟶ Qty

Sno ⟶ Sname

Sname ⟶ Sno

This relation has two overlapping candidate keys,because there are two composite candidate keys (Sno, Pno) and (Sname,Pno) out of which Pno is common attribute in both the candidate keys, so this is due case of overlapping of candidate keys.

Possible FD diagram of this case is:



Here, both the relations are in 3NF, because every non-key attribute is non-transitively fully functional dependent on the primary key.

In above relation, there is only one non-key attribute i.e. Qty and it is FFD and non transitively dependent on the primary key.

Sname, Sno are not non-key attributes because they can participate into the primary key as shown in FD diagram.

But, SSP relation is not in BCNF because this relation has four determinants:

(Sno, Pno)

(Sname, Pno)

(Sno)

(Sname)

Out of these four determinants two determinants (Sno, Pno) and (Sname, Pno) are unique but Sno and Sname determinants are not candidate keys.

In order to make this relation in BCNF we non-loss decompose this relation in two projections SN (Sno, Sname) and SP (Sno, Pno, Qty).

SN relation has two determinants Sno, Sname and both are unique.

SP has one determinant (Sno, Pno) and is also unique.

These two relations (SN, SP) remove all anomalies of SSP relation.

Another Case:

Consider a relation SST with attributes (Student, Subject, Teacher).

There are following rules applied on above relation:

- For each subject, each student of that subject is taught by only one teacher
- Each teacher teaches only one subject
- Each subject is taught by several teachers

Table shows a sample data:

SST

Student	Subject	Teacher
Ajay	Math	Prof. White
Ajay	Physics	Prof. Green
Kumar	Math	Prof. White
Kumar	Physics	Prof. Neha

FD's of above relation are

According to condition 1:

Subject, Student combination gives only one teacher

(Subject, Student) ⟶ Teacher

According to condition 2

Each teacher teaches only one subject

Teacher ⟶ Subject

Since, each teacher teaches only one subject, so a student can taught one subject by only one teacher. In other words, the combination of teacher and student can also determine subject.

(Teacher, Student)⟶ Subject

Once again the relation is in 3NF,but not in BCNF.

This relation has following FDs:

(Subject, Student) ⟶ Teacher

Teacher ⟶ Subject

(Teacher, Student) ⟶ Subject

This is a case of overlapping of candidate keys, because there are two composite candidate keys (Subject, Student) and (Teacher, Student) and Student is a common attribute in both the candidate keys. So, this database must be normalized according the BCNF.

This relation suffers again from the anomalies as discussed below:

For example, if we wish to delete the information that Kumar is studing Physics, we cannot do so without losing the information that Prof. Neha teaches Physics.

These difficulties are caused by the fact that teacher is determinant but not a candidate key.

In order to make it in BCNF, teacher must be candidate key, so original relation is replaced by two projections ST (Student, Teacher) and TJ (Teacher, Subject).

All the anomalies which were present in SST, now removed in these two relations.

Practice Session:

1. Show that every two-attribute relation is in BCNF. That is, if r(X, Y) then r(X,Y) is in BCNF.

   Solution: Let us consider the following cases:

   a) X is the sole key of the relation. In this case, the nontrivial dependency X $\longrightarrow$Y has X as a super key since X $\subset$ Y.

   b) Y is the sole key of the relation. In this case, the nontrivial dependency Y $\longrightarrow$X has Y as a super key since Y $\subset$ Y.

   c) Both X $\longrightarrow$ Y and Y $\longrightarrow$ X hold simultaneously. Then whatever PK we consider for the relation we will have either X or Y as its determinant. Either one of the two possible cases has already been considered under (a) or (b).

2. Consider the relation Supplier (Supplier-No, Part-No, Supplier-Name, Supplier-Control, Price) and assume that only the following FDs hold for this relation: Supplier-No $\rightarrow$ Supplier-Name, Supplier-No $\rightarrow$ Supplier-Control. What type of data anomalies does this relation have in its present form? Transform it to 3NF if not already in that form.

   Solution: This relation presents insertion anomalies, deletion anomalies and update anomalies. In this relation we cannot enter a Supplier-Control until that supplier supplies a part (insertion anomaly). Notice that this is necessary to preserve the integrity constraint of the key. If a supplier stops temporarily supplying a particular part, then the deletion of the last tuple containing that Supplier-No also deletes the Supplier-Control of the supplier (deletion anomaly). Finally, if the Supplier-Control of a particular supplier needs to be updated, we must look for every tuple that contains that supplier as part of the key. If a supplier supplies many parts and we fail to update all the corresponding tuples, the database may end up in an inconsistent state. We can transform this relation into a 2NF as follows:

   Supplier (Supplier-No, Supplier-Name, Supplier-Control) and

   Part (Part-No, Supplier-No, Price)

3. Consider the relation for release RECORD (Title, Performer, Style, Price, Label, Producer) and the dependencies shown below. Indicate what is the highest normal form of this relation. Indicate a possible 3NF decomposition of this relation.

   Title $\longrightarrow$ Label, Style Style $\longrightarrow$ Price Performer $\longrightarrow$ Producer

Solution: The key of the relation is the composite attribute Title, Performer. As it stands the relation is in 1NF because there is at least one partial dependency on the key. For example, Performer ⟶ Producer. A possible decomposition may be Record (Title, Performer), Genre(Title, Label, Style), Producer(Performer, Producer-Name), Cost(Style, Price).

7.7 FOURTH NORMAL FORM (4NF)

A relation R is in Fourth Normal Form (4NF) if and only if the following conditions are satisfied simultaneously:

1. R is already in 3NF or BCNF.
2. If it contains no multi-valued dependencies.

Multi-Valued Dependency (MVD)

MVD is the dependency where one attribute value is potentially a 'multi-valued fact' about another. Consider the table

Table CUSTOMER_ADDRESS

Customer_Name	Address
Raj	New Delhi
Raj	Amritsar
Suneet	Amritsar
Suneet	Batala
Ankit	Qadian

In this example, 'Address' is a multi-valued fact 'Customer_Name' and the converse is also true.

For example, the attribute 'Address' takes on the two values 'New Delhi' and 'Amritsar' for the single 'Customer_Name' value 'Raj'. The attribute 'Customer_Name' takes on the values 'Raj' and 'Suneet' for the single 'address' value 'Amritsar'.

MVD can be defined informally as follows:

MVDs occur when two or more independent multi valued facts about the same attribute occur within the same table. It means that if in a relation R having A, B and C as attributes, B and C are muti-value facts about A, which is represented as A⟶ ⟶B and A⟶⟶C, then muti value dependency exist only if B and C are independent of each other.

There are two things to note about this definition.

Firstly, in order for a table to contain MVD, it must have three or more attributes.

Secondly, it is possible to have a table containing two or more attributes which are inter-dependent multi valued facts about another attribute.

This does not give rise to an MVD. The attributes giving rise to the multi-valued facts must be independent of each other consider the following table:

Table STUDENT_BOOK

Student_name	Librarian	Text_book	Date
Ankit	Jill	Mechanics	Apr
Amit	Mary	Mechanics	Apr
Raj	Mary	first_year_English	Jan
Ankit	Jill	Mechanics	Jun
Raj	Mary	first_year_English	Feb
Raj	Jill	first_year_English	July
Raj	Fred	first_year_german	Jan

The table lists students, the textbooks; they have borrowed, the librarians issuing them and the date of borrowing. It contains three multi-valued facts about students, the books they have borrowed, the librarians who have issued these books to them and the dates upon which the books were borrowed. However, these multi-valued facts are not independent of each other. There is clearly an association between librarians, the textbooks they have issued and the dates upon which they issued the books. Therefore, there are no MVDs in the table. Note that there is no redundant information in this table. The fact that student 'Ankit', for example, has borrowed the book 'Mechanics' is recorded twice, but these are different borrowing, one in April and the other in June and therefore constitute different items of information.

Now consider another table example involving Course, Student_name and text_book.

Table COURSE _ STUDENT_BOOK

Course	Student_name	Text_book
Physics	Ankit	Mechanics
Physics	Ankit	Optics
Physics	Rahat	Mechanics
Physics	Rahat	Optics
Chemistry	Ankit	Organic_chemistry
Chemistry	Ankit	Inorganic_ chemistry
English	Raj	English_literature
English	Raj	English_grammar

This table lists students, the courses they attend and the textbooks they use for these courses. The text books are prescribed by the authorities for each course, that is, the students have no say in the matter. Clearly the attributes 'Student_name' and 'Text_book' are multi-valued facts about the attribute 'Course'. However, since a student has no influence over the text books to be used for a course, these multi-valued facts about courses are independent of each other. Thus the table contains an MVD. Muti-value facts are represented by →→

Here, in above database following MVDs exists:

Course →→ Student_name

Course →→ Text_book

Here, Student_name and Text_book are independent of each other.

Anomalies of database with MVDs

This form of the table is obviously full of anomalies. If a new student joins the physics course then we have to make two insertions for that student in the database, which is equal to the number of physics textbooks. Consider the problem if there are hundred textbooks for a subject. Similarly, if a new textbook is introduced for a course, then again we have to make multiple insertions in the database, which is equal to number of students for that course. So, there is a high degree of redundancy in the database, which will lead to update problems.

The above database is in First, Second and Third normal form because for each row-column intersection we have at-most single entry and primary key is the combination of three columns (Course, Student_name, Text_book). So, it does not have any non-key attribute. It satisfies second and third normal form because it only refers to non-key attributes. The relation is also in BCNF, since all three attributes concatenated together constitute its key, yet it is clearly contained anomalies and requires decomposition with the help of fourth normal form.

Solution of above anomalies with Fourth Normal Form

This problem of MVD is handled in Fourth Normal Form. Here, is the rule for transforming a relation to 4NF given by Fagin.

Rule to transform a relation into Fourth Normal Form

A relation R having A,B, and C, as attributes can be non loss-decomposed into two projections R1(A,B) and R2(A,C) if and only if the MVD A⟶ ⟶B|C hold in R.

Looking again at the un-decomposed COURSE_STUDENT_BOOK table, it contains a multi-valued dependency as shown below:

Course ⟶ ⟶ Student_name

Course ⟶ ⟶ Text_book

To put it into 4NF, two separate tables are formed as shown below:

COURSE_STUDENT (Course,Student_name)

COURSE_BOOK (Course,text_book)

COURSE_STUDENT

Course	Student_name
Physics	Ankit
Physics	Rahat
Chemistry	Ankit
English	Raj

COURSE_BOOK

Course	Text_book
Physics	Mechanics
Physics	Optics
Chemistry	Organic_chemistry
Chemistry	Inorganic_ chemistry
English	English_literature
English	English_grammar

Now, we can easily check that all the above anomalies of STUDENT_COURSE_BOOK database are removed. For example, if now a new student joins a course then we have to make only one insertion in COURSE_STUDENT table and if a new book introduced for a course then again we have to make a single entry in COURSE_BOOK table, so this modified database eliminates the problem of redundancy which also solves the update problems.

Practice Session 3:

Example: Consider the following database of STUDENT and normalize it.

STUDENT (Student_Name, Equipment, Language)

Table STUDENT

Student_Name	Equipment	Language
Suneet	PC	English
	Workstation	German
Raj	Workstation	English
		German
		Spanish

Solution : In order to normalize it first it flattens with first normal form as shown below:

STUDENT1

Student_Name	Equipment	Language
Suneet	PC	English
Suneet	PC	German
Suneet	Workstation	German
Suneet	Workstation	English
Raj	Workstation	English
Raj	Workstation	German
Raj	Workstation	Spanish

This table lists students, the equipment they has been allocated to them and the foreign languages in which they are fluent. This database shows that 'equipment' and 'language' are independent multi-valued facts about 'Student_Name' that is it contains a multi-valued dependency.

This form of the table is obviously full of anomalies. There is a high degree of redundancy that will lead to update problems. If Raj has his workstation taken away, then the information about his language skills is lost and if he acquires a PC, then all the information about his language skills has to be repeated, that is, three new rows have to be inserted (because of the entity integrity rule).

The table is in BCNF, since all three attributes concatenated together constitute its key, yet it contain anomalies and requires decomposition. Since the database contain MVDs, so it should be decomposed with the help of rule of fourth normal form decomposition. The database contain the following MVDs:

Student_Name $\longrightarrow\longrightarrow$ Equipment

Student_Name $\longrightarrow\longrightarrow$ Language

So, it should be decomposed in following database according to forth normal form.

STUDENT_EQUIPMENT (Student_Name, Equipment)

STUDENT_LANGUAGE (Student_Name, Lanuage)

STUDENT_EQUIPMENT

Student_Name	Equipment
Suneet	PC
Suneet	Workstation
Raj	Workstation

STUDENT_LANGUAGE

Student_Name	Equipment
Suneet	English
Suneet	German
Raj	English
Raj	German
Raj	Spanish

Example: Suppose that employees can be assigned to multiple projects. Also suppose that employees can have multiple job skills as shown in database. Try to normalize the database.

EMP_PROJECT_SKILL

Emp_No	Project_No	Skill
1211	1	Analysis
	7	Design
		Program

Solution : In order to normalize it we flatten the database with first normal form as shown below:

EMP_PROJECT_SKILL1

Emp_No	Project_No	Skill
1211	1	Analysis
1211	1	Design
1211	1	Program
1211	7	Analysis
1211	7	Design
1211	7	Program

This database shows that Project_No and Skill are independent multi-valued facts about Emp_No that is it contains a multi-valued dependency.

This form of the table is obviously full of anomalies. There is a high degree of redundancy that will lead to update problems. Since the database contains MVDs, so it should be decomposed with the help of rule of fourth normal form. Here, the database contain the following MVDs:

Emp_No ⟶ ⟶ Project_No

Emp_No⟶ ⟶ Skill

Here, Project_No and Skill are independent to each other, so it should be decomposed in to following database according to forth normal form.

EMP_PROJECT (Emp_No, Project_No)

EMP_SKILL (Emp_No, Skill)

EMP_PROJECT

Emp_No	Project_No
1211	1
1211	7

EMP_SKILL

Emp_No	Skill
1211	Analysis
1211	Design
1211	Program

8.8 FIFTH NORMAL FORM (5NF)

A relation R is in Fifth Normal Form (5NF) if and only if the following conditions are satisfied simultaneously:

1. R is already in 4NF.
2. It cannot be further non-loss decomposed.

5NF is of little practical use to the database designer, but it is of interest from a theoretical point of view and a discussion of it is included here to complete the picture of the further normal forms.

In all of the further normal forms discussed so far, no loss decomposition was achieved by the decomposing of a single table into two separate tables. No loss decomposition is possible because of the availability of the join operator as part of the relational model. In considering 5NF, consideration must be given to tables where this non-loss decomposition can only be achieved by decomposition into three or more separate tables. Such decomposition is not always possible as is shown by the following example.

Consider the table

AGENT_COMPANY_PRODUCT (Agent, Company, Product _Name)

This table lists agents, the companies they work for and the products they sell for those companies. The agents do not necessarily sell all the products supplied by the companies they do business with. An example of this table might be:

Agent	Company	Product_Name
Suneet	ABC	Nut
Suneet	ABC	Screw
Suneet	CDE	Bolt
Raj	ABC	Bolt

The table is necessary in order to show all the information required. Suneet, for example, sells ABC's Nuts and Screws, but not ABC's Bolts. Raj is not an agent for CDE and does not sell ABC's Nuts or Screws. The table is in 4NF because it contains no multi-valued dependency. It does, however, contain an element of redundancy in that it records the fact that Suneet is an agent for ABC twice. But there is no way of eliminating this redundancy without losing information. Suppose that the table is decomposed into its two projections, P1 and P2.

P1

Agent	Company
Suneet	ABC
Suneet	CDE
Raj	ABC

P2

Agent	Product_Name
Sunnet	Nut
Suneet	Screw
Suneet	Bolt
Raj	Bolt

The redundancy has been eliminated, but the information about which companies make which products and which of these products they supply to which agents has been lost. The natural join of these projections over the 'agent' columns is:

Agent	Company	Product_Name
Suneet	ABC	Nut
Suneet	ABC	Screw
Suneet	ABC	Bolt*
Suneet	CDE	Nut*
Suneet	CDE	Screw*
Suneet	CDE	Bolt
Raj	ABC	Bolt

The table resulting from this join is spurious, since the asterisked row of the table contains incorrect information. Now suppose that the original table were to be decomposed into three tables, the two projections, P1 and P2 which have already shown, and the final, possible projection, P3.

P3

Company	Product_Name
ABC	Nut
ABC	Screw
ABC	Bolt
CDE	Bolt

If a join is taken of all three projections, first of P1 and P2 with the (spurious) result shown above, and then of this result with P3 over the 'Company' and 'Product name' column, the following table is obtained:

Agent	Company	Product_Name
Suneet	ABC	Nut
Suneet	ABC	Screw
Suneet	ABC	Bolt*
Suneet	CDE	Bolt
Raj	ABC	Bolt

This still contains a spurious row. The order in which the joins are performed makes no difference to the final result. It is not simply possible to decompose the 'AGENT_COMPANY_PRODUCT' table, populated as shown, without losing information. Thus, it has to be accepted that it is not possible to eliminate all redundancies using normalization techniques, because it cannot be assumed that all decompositions will be non-loss.

But now consider the different case where, if an agent is an agent for a company and that company makes a product, then he always sells that product for the company. Under these circumstances, the 'agent_company_product' table as shown below:

Agent	Company	Product_Name
Suneet	ABC	Nut
Raj	ABC	Bolt
Raj	ABC	Nut
Suneet	CDE	Bolt
Suneet	ABC	Bolt

The assumption being that ABC makes both Nuts and Bolts and that CDE makes Bolts only. This table can be decomposed into its three projections without loss of information as demonstrated below:

P1

Agent	Company
Suneet	ABC
Suneet	CDE
Raj	ABC

P2

Agent	Product_Name
Suneet	Nut
Suneet	Bolt
Raj	Bolt
Raj	Nut

P3

Company	Product_Name
ABC	Nut
ABC	Bolt
CDE	Bolt

All redundancy has been removed, if the natural join of P1 and P2 is taken, the result is:

Agent	Company	Product_Name
Suneet	ABC	Nut
Suneet	ABC	Bolt
Suneet	CDE	Nut*
Suneet	CDE	Bolt
Raj	ABC	Bolt
Raj	ABC	Nut

The spurious row is marked as asterisked "*". Now, if this result is joined with P3 over the column 'company' and 'product_name' the following table is obtained:

Agent	Company	Product_Name
Suneet	ABC	Nut
Suneet	ABC	Bolt
Suneet	CDE	Bolt
Raj	ABC	Bolt
Raj	ABC	Nut

This is a correct recomposition of the original table and a no loss decomposition into the three projections was achieved. Again, the order in which the joins are performed does not affect the final result. The original table, therefore, violated 5NF simply because it was non-loss decomposable into its three projections.

In the first case exemplified above, non-loss decomposition of the 'agent_company_product' table was not possible. In the second it was. If a table is non- loss decomposable as in the second case, it is said to be in violation of 5NF. The difference, of course, lay in certain semantic properties of the information being represented. These properties were not understandable simply by looking at the table, but had to be supplemented by further information about the relationship between products, agents and companies.

Detecting that a table violates 5 NF is very difficult in practice and for this reason this normal form has little if any practical application. The theoretical concept of fifth normal form is discussed in the following paragraphs.

Suppose that the statement, 'The agent_company_product' table is equal to the join of its three projections is to hold true, this is another way of saying that it can be non-loss decomposed into its three projections and is equivalent to saying.

IF the tuple 'agent X, company Y' appears in P1

AND the tuple 'agent X, product Z' appears in P2

AND the tuple 'company Y, product Z' appears in P3

Then the row 'agent X, company Y, product Z' must have appeared in 'agent_company_product'.

If the reader cares to re-examine the projections P1, P2, and P3 from the two versions of the table which were illustrated earlier, then, it will be seen that the earlier version which was in 5NF does not confirm to the above rule, whereas the later version, which violated 5NF does.

The rule is referred to as a Join Dependency, because it holds good only if a table can be reconstituted without loss of information from the join of certain specified projections of it.

The notation used for a join dependency on table T is:

*(X,Y,.....Z)

Where X,Y,...Z are projections of T.

Table T is said to satisfy the above join dependency, if it is equal to the join of the projections X, Y,... Z.

Thus, the second example given of the table 'agent_company_product' can be said to satisfy the join dependency:

*(P1, P2, P3)

In the discussion of the other further normal forms use was made of the concepts of functional and multi-valued dependencies. In dealing with 5NF the concept of join dependency has been introduced (in a very informal way).

5NF is defined by the statement

A table T is in fifth normal form if every join dependency in T is a consequence only of the candidate keys of T.

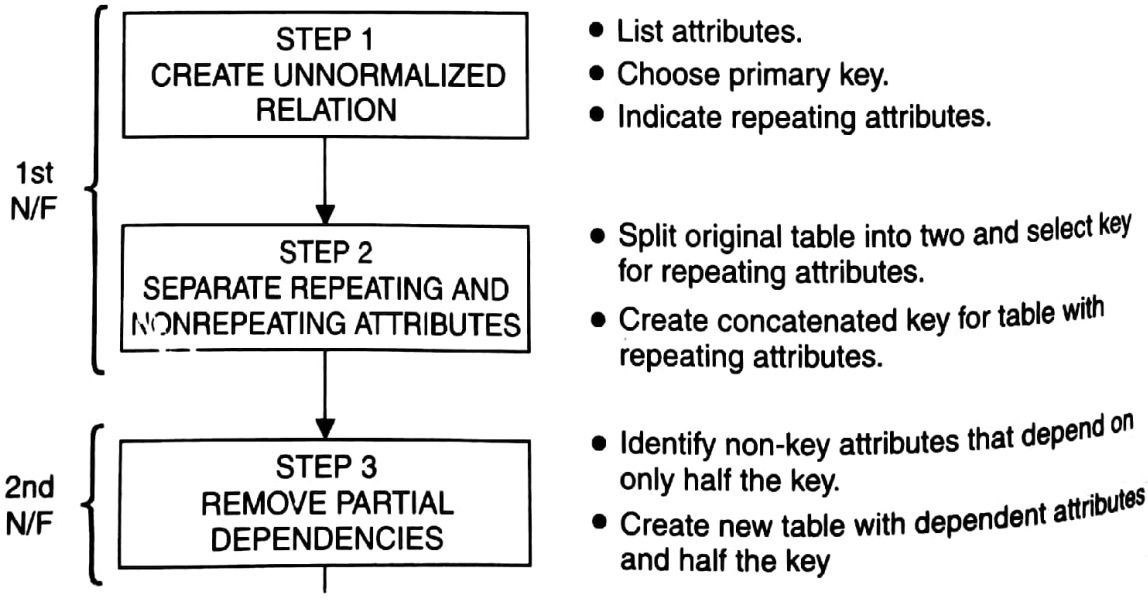
The second version of the table 'agent_company_product' illustrated earlier violated 5NF, because the join dependency *(agent, company, product_name) was not a consequence only of the primary key for the table, but also a consequence of the tuple formation rule which was given earlier.

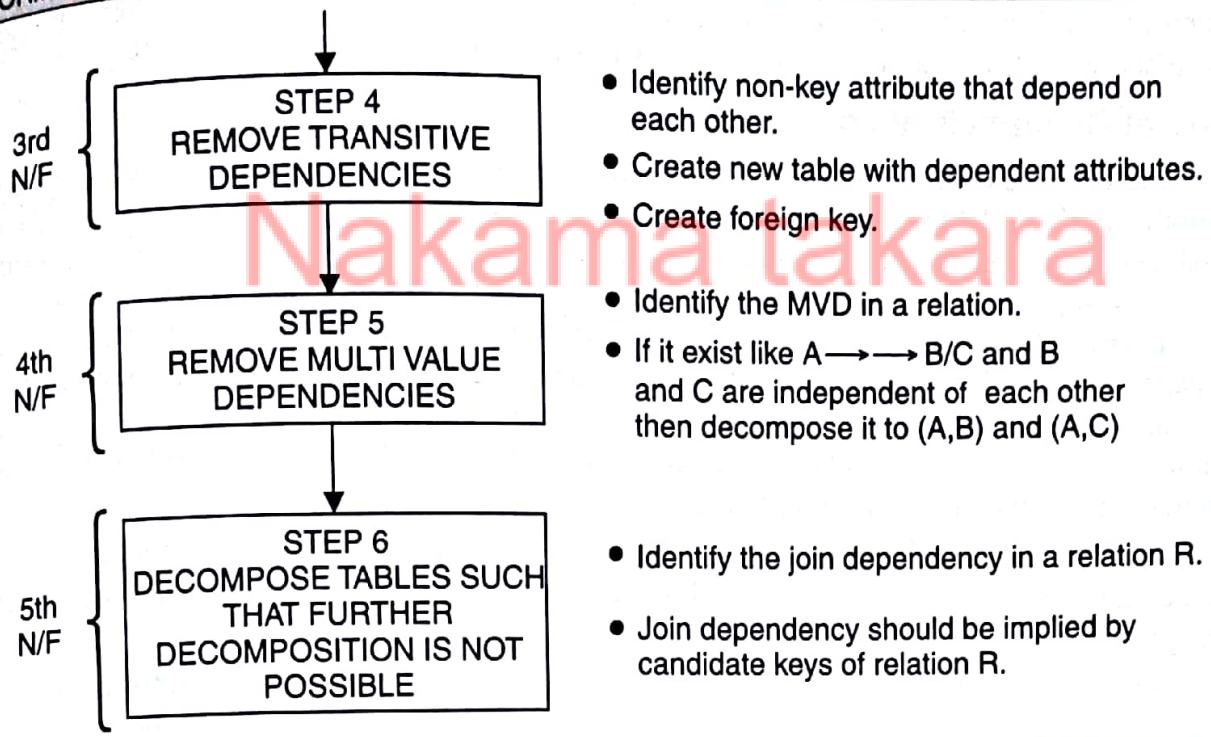
In the first, example of 'agent_company_product' there was no application of this rule, hence no join dependency other than that on the primary key. Thus, the table was in 5NF. It can be shown that if a table is in 5NF, then, because join dependencies are the 'ultimate' form of dependency, it must also be in 4NF and thus confirm to all the further normal forms. The problem with this is that detecting join dependencies is, in practice, very difficult. For this reason, 5NF is largely of academic interest.

8.9 SIXTH NORMAL FORM FOR TEMPORAL DATABASES

A table is in sixth normal form (6NF) if and only if it satisfies no non-trivial join dependencies at all, meaning that the fifth normal form is also satisfied. The sixth normal form was only defined when extending the relational model to take into account the temporal dimension. Most SQL technologies do not take into account this work, and most temporal extensions to SQL are not relational. The detailed discussion on temporal databases is given in chapter 18.

8.10 STEPS OF NORMALIZATION





8.11 DENORMALIZATION

Denormalization is the process of attempting to optimize the performance of a database by adding redundant data or by grouping data. In some cases, denormalization helps cover up the inefficiencies inherent in relational database software. A relational normalized database imposes a heavy access load over physical storage of data even if it is well tuned for high performance.

A normalized design will often store different but related pieces of information in separate logical tables (called relations). If these relations are stored physically as separate disk files, completing a database query that draws information from several relations (a *join operation*) can be slow. If many relations are joined, it may be prohibitively slow. There are two strategies for dealing with this. The preferred method is to keep the logical design normalized, but allow the DBMS to store additional redundant information on disk to optimize query response. In this case it is the DBMS software's responsibility to ensure that any redundant copies are kept consistent. This method is often implemented in SQL as indexed views (MS SQL) or materialized views (Oracle). A view represents information in a format convenient for querying, and the index ensures that queries against the view are optimized.

The more usual approach is to denormalize the logical data design. With care this can achieve a similar improvement in query response, but at a cost—it is now the database designer's responsibility to ensure that the denormalized database does not become inconsistent. This is done by creating rules in the database called *constraints*, that specify how the redundant copies of information must be kept synchronized. It is the increase in logical complexity of the database design and the added complexity of the additional constraints that make this approach hazardous. Some times a denormalized database under heavy write load may actually offer worse performance than its functionally equivalent normalized counterpart.

A denormalized data model is not the same as a data model that has not been normalized, and denormalization should only take place after a satisfactory level of normalization has taken place and that any required constraints and/or rules have been created to deal with the

inherent anomalies in the design. For example, all the relations are in third normal form and any relations with join and multi-valued dependencies are handled appropriately.

Uses of Denormalization

Databases intended for Online Transaction Processing (OLTP) are typically more normalized than databases intended for Online Analytical Processing (OLAP). OLTP Applications are characterized by a high volume of small transactions such as updating a sales record at a super market checkout counter. The expectation is that each transaction will leave the database in a consistent state. By contrast, databases intended for OLAP operations are primarily "read mostly" databases. OLAP applications tend to extract historical data that has accumulated over a long period of time. For such databases, redundant or "denormalized" data may facilitate Business Intelligence applications. Specifically, dimensional tables in a star schema often contain denormalized data.

Denormalization is also used to improve performance on smaller computers as in computerized cash-registers and mobile devices, since these may use the data for look-up only (e.g. price lookups).

Denormalization may also be used when no RDBMS exists for a platform (such as Palm), or no changes are to be made to the data and a swift response is crucial.

8.12 CASE STUDIES BASED ON NORMALIZATION OF DATA

To equip the learners with database designing practices based on normalization, here we provide the number of case studies. Students are requested to normalize the database of each case study on their own and after designing their own database, they can match their designed database with the solution of the each study. We hope that these solved case studies will help to raise your confidence in the database designing.

8.12.1 Supplier-Part Case Study

Consider the following case of Supplier Part database, which is represented, in following database.

SUPPLIER (Sno, City, Status, Pno, Qty)

Sno	City	Status	Pno	Qty
S1	AMRITSAR	20	P1	300
			P2	200
			P3	400
			P4	200
			P5	100
			P6	100
S2	MUMBAI	10	P1	300
			P2	400
S3	MUMBAI	10	P2	200
S4	DELHI	20	P2	200
			P4	300
			P5	400

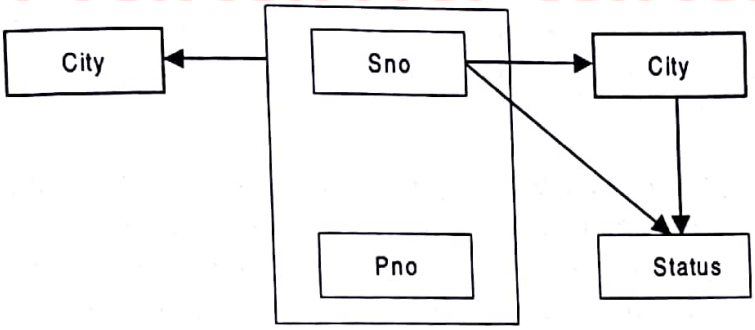
Here, Sno is unique and status is FD on City.

City ⟶Status

(The meaning of this constraint is that a suppliers status is determined by the corresponding location e.g. all Amritsar suppliers must have a status of 20)

The primary key of SUPPLIER is the combination of (Sno, Pno).

Functional dependency diagram for this relation is show below:



Here, Qty is FFD on primary key.

(Sno, Pno.) ⟶ Qty, Because it does not FD on Sno and Pno individually.

But, City is not FFD on primary key, because City is FD on Sno which is proper subset of composite attribute (Sno,Pno)

Similarly, (Sno, Pno)→Status are not FFD because status is FD on proper subset of composite attribute i.e. Sno.

Flatten view of SUPPLIER database:

Sno	City	Status	Pno	Qty
S1	AMRITSAR	20	P1	300
S1	AMRITSAR	20	P2	200
S1	AMRITSAR	20	P3	400
S1	AMRITSAR	20	P4	200
S1	AMRITSAR	20	P5	100
S1	AMRITSAR	20	P6	100
S2	MUMBAI	10	P1	300
S2	MUMBAI	10	P2	400
S3	MUMBAI	10	P2	200
S4	DELHI	20	P2	200
S4	DELHI	20	P4	300
S4	DELHI	20	P5	400

Anomalies in First Normal Form

Insertion

We cannot enter the fact that a particular supplier is located in a particular city until that supplier supplies at least one part. Indeed, the tabulation as shown above does not show supplier S5 is located in Qadian. The reason is that, until S5 supplies some part, we have no

appropriate primary key value. (Remember that, by Integrity Rule 1, no component of a primary key value may be null). In relation SUPPLIER, primary key values consist of a supplier number and a part number.

Deletion

If we delete the only tuple for a particular supplier, we destroy not only the shipment connecting that supplier to some part but also the information that the supplier is located in a particular city. For example, if we delete the SUPPLIER tuple with Sno value S3 and Pno value P2, we lose the information that S3 is located in Mumbai.

Updation

The city value for a given supplier appears in SUPPLIER many times, in general. This redundancy causes update problems. For example, if supplier S1 moves from Amritsar to Jalandhar, we are faced with either the problem of searching the SUPPLIER relation to find every tuple connecting S1 and Delhi (and changing it) or the possibility of producing an inconsistent result (the city for S1 may be given as Amritsar in one place and Jalandhar in another).

Removal of anomalies of First Normal Form

The solution to removal of above anomalies is to convert the first normal form into second normal form.

Definition of Second Normal Form

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

To transform SUPPLIER database into 2NF we move the columns Sno, Status, and City to a new table called SUPPLIER_SECOND and second table SP remains as (Sno, Pno, QTY). The column Sno becomes the primary key of this new table SUPPLIER_SECOND. Below database shows that now it is possible to store the information of supplier S5 who does not supply any part in table SUPPLIER_SECOND.

SUPPLIER_SECOND

SNO	STATUS	CITY
S1	20	Amritsar
S2	10	Mumbai
S3	10	Mumbai
S4	20	Delhi
S5	30	Qadian

SP

SNO	PNO	QTY.
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P7	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

It should be clear that this revised structure overcomes all the problems that we had with previous database.

Insertion

We can enter the information that S7 is located in Qadian, even though S7 does not currently supply any parts, by simply inserting the appropriate tuple into SUPPLIER_SECOND.

Deletion

We can delete the shipment connecting S3 and P2 by deleting the appropriate tuple from SP; we do not lose the information that S3 is located in Mumbai.

Updation

In the revised structure, the city for a given supplier appears once, not many times (the redundancy has been eliminated). Thus, we can change the city for S1 from Amritsar to Jalandhar by changing it once.

In modified databases of SUPPLIER_SECOND (Sno,Status,City) and SP (Sno,Pno,Qty) all the non-key (an attribute is non-key if it does not participate in the primary key) attributes like Status, City (in SUPPLIER_SECOND) and Qty (in SP) are fully functional dependent on the primary key.

Relations SUPPLIER_SECOND and SP are both 2NF (the primary keys are Sno and the combination (Sno, Pno) respectively). Relation SUPPLIER is not 2NF. A relation that is in first normal form and not in second can always be reduced to an equivalent collection of 2NF relations. (Note, incidentally, that a 1NF relation that is not also 2NF must have a composite primary key). The reduction consists of replacing the relations by suitable projections. The collection of these projections is equivalent to the original relation, in the sense that the original relation can always be recovered by taking the natural join of these projections, so no information is lost in the process (which is highly important, of course). In other words, the process is reversible.

In our example, SUPPLIER_SECOND and SP are projections of SUPPLIER, and SUPPLIER is the natural join of SUPPLIER_SECOND and SP over Sno.

Anomalies in SECOND Relation (2nd Normal Form)

The SUPPLIER_SECOND, SP structure still causes problems. However, relation SP is satisfactory; as a matter of fact, relation SP is now in third normal form, and we shall ignore it for the remainder of this section.

Relation SUPPLIER_SECOND, on the other hand, still suffers from following anomalies:

Insertion

We cannot enter the fact that a particular city has a particular status value.

For example, we cannot state that any supplier in Ludhiana must have a status of 70 until we have some supplier located in that city. The reason is again, that until such a supplier exists we have no appropriate primary key value.

Deletion

If we delete the only tuple for a particular city, we destroy not only the information for the supplier concerned but also the information that city has that particular status value. For example, if we delete the tuple for S5, we lose the information that the status for Qadian is 30.

Updation

The status value for a given city appears in SUPPLIER_SECOND many times, in general (the relation still contains some redundancy). Thus, if we need to change the status value for Delhi from 20 to 30, we are faced with either the problem of searching the SECOND relation to find every tuple for Delhi or the possibility of producing an inconsistent result (the status for Delhi may be given as 20 in one place and 30 in another).

Solution of above problems

The anomalies discussed above occur due to transitive dependence of status on primary key (Sno) in table SUPPLIER_SECOND. Transitive dependence means that attribute depends on the primary key on more than one way. Here, status is FD on City as well as on Sno and City is FD on Sno as shown below:

Sno ⟶Staus

Sno ⟶City

City ⟶Status

Thus, we can say that non-key attribute Status is FD on Primary key Sno directly and another way through City, this case of dependence is called as Transitive Dependence.

The solution to the above problems is to normalize with third normal form, which states as:

Definition of Third Normal Form

A relation R is in third normal form (3NF) if and only if it is in 2NF and every non-key attribute is non-transitively dependent on the primary key.

In order to normalize it, we replace the original relation (SUPPLIER_SECOND) by two projections, in this case SC (Sno, City) and CS (City , Status). The process is reversible, once again, since SUPPLIER_SECOND is the join of SC and CS over City. It should be clear that this revised structure overcomes all the problems that we had with previous database.

Insertion

We can enter the information that status of city Ludihana is 20 in table CS.

Deletion

We can delete the information of supplier without losing the information about Cities and their status.

Updation

In the revised structure, the status of city appears once, not many times (the redundancy has been eliminated). Thus, we can change the status of city Delhi from 20 to 30 by changing it once).

Sno → City

SC

SNo.	City
S1	Amritsar
S2	Mumbai
S3	Mumbai
S4	Delhi
S5	Qadian

City → Status

CS

City	Starts
Amritsar	20
Mumbai	10
Delhi	20
Qadian	20
Patiala	20

Relations SC and CS are both 3NF; relation SUPPLIER_SECOND is not. (The primary keys for SC and CS are Sno and City, respectively). A relation that is in second normal form and not in third can always be reduced to an equivalent collection of 3 NF relations. We have already indicated that the process is reversible, and hence no information is lost in the reduction; however, the 3NF collection may contain information, such as the fact that the status for Patiala is 20, that could not be represented in the original 2NF relation. Just as the SUPPLIER_SECOND, SP structure was a slightly better representation of the real world than the 1NF relation SUPPLIER, so the SC, CS structure is a slightly better representation than 2NF relation SUPPLIER_SECOND.

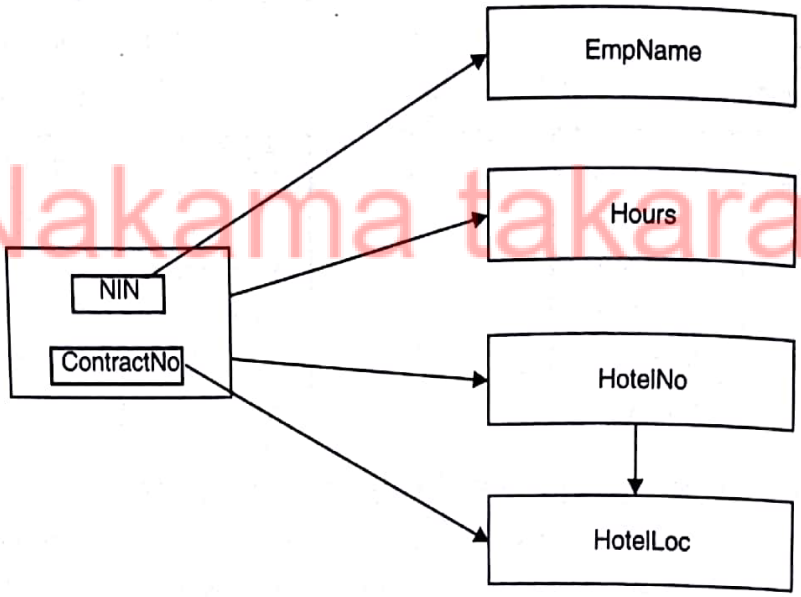
8.12.2 Hotel Services Case Study

The relation below provides some sample data for an agency called Hotel Services that supplies part-time/temporary staff to hotels within the Punjab region. The relation lists the number of hours worked by each staff at various hotels. Every supplier has unique NIN number, the relation is first normal form (1NF). Assuming that a contract is for one hotel only but a staff may work in more than one hotel on different contracts. The sample database of Contracts table is given below. Normalize this database and discuss all the intermediate steps in the process of normalization.

Contracts

NIN	ContractNo	Hours	EmployeeName	HotelNo	HotelLoc
1001	C1010	20	Vikas Garg	H101	Patiala
1015	C1021	20	Urvang Joshi	H123	Amritsar
1020	C1010	31	Urvang Joshi	H101	Patiala
1001	C1021	20	Vikas Garg	H123	Amritsar
1015	C1125	35	Urvang Joshi	H241	Amritsar
1272	C1237	43	Deepak Verma	H371	Patiala

Solution: The given relation is in first normal form, because for every-row column combination there is at most single value. The primary key of the above relation is (NIN, ContractNo) combination and the Functional Dependence diagram is given below.



Functional Dependence Diagram of Contracts relation

From the above FD diagram, it is clear that non-key attribute "Hours" is fully-functional dependent on primary key attributes (NIN, ContractNo). But, the attributes "HotelNo" and "HotelLoc" are not fully functional dependent on the primary key, because they are functional dependent on the subset of the primary key i.e. "ContractNo". Similarly, the attribute "EmpName" is not fully functional dependent on the primary key, because it is functional dependent on the subset of the primary key i.e. "NIN".

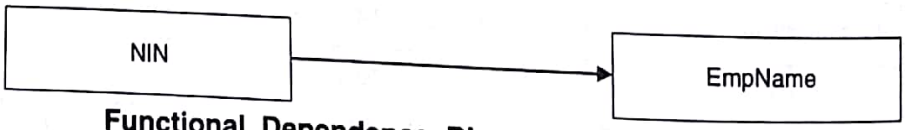
Thus, in order to convert this relation to 2NF it is decomposed into three tables Staff_Contract, Staff and Contract_Hotel according to the transformation rule of Section 7.4.

The functional dependence diagrams of three tables are as follows:

Staff_Contract (NIN, ContractNo, Hours)

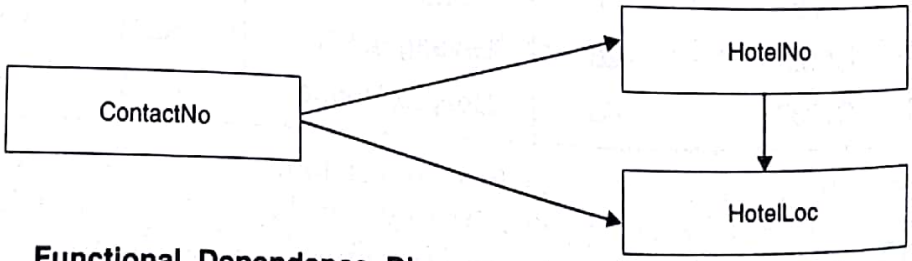
Functional Dependence Diagram of Staff_Contract relation

Staff (NIN, EmpName)



Functional Dependence Diagram of Staff relation

Contract_Hotel (ContractNo, HotelNo, HotelLoc)



Functional Dependence Diagram of Contract_Hotel relation

There is a case of transitive dependence in the Contract_Hotel relation as shown in its FD diagram. Here, one non key attribute "HotelNo" functionally determines other non-key attribute "HotelLoc" which results transitive dependence, because now the attribute "HotelLoc" functionally dependent on primary key two ways one directly and other through or transitively by "HotelNo". The transitive dependence is removed by decomposing the Contract_Hotel according to the 3rd Normal Form, the rule discussed in the Section 8.5. According to 3rd normal form Contract_Hotel relation is decomposed into two relations Contract_Hotel_List and Hotel as shown below.

Contract_Hotel_List (ContractNo, HotelNo)

Functional Dependence Diagram of Contract_Hotel relation

Hotel (HotelNo, HotelLoc)

Functional Dependence Diagram of Contract_Hotel relation

Thus, the given table Contracts is normalized by decomposing it into four tables as shown below with primary keys underlined. The above database is normalized up to 3rd normal form.

Staff_Contract (NIN, ContractNo, Hours)

Staff (NIN, EmpName)

Contract_Hotel_List (ContractNo, HotelNo)

Hotel (HotelNo, HotelLoc)

8.12.3 Book_Order Services Case Study

Normalize the following Order_Book database. Indicate the tables after each normal form:

Order_Number, Book_Name, Quantity, Book_Price, Publisher.

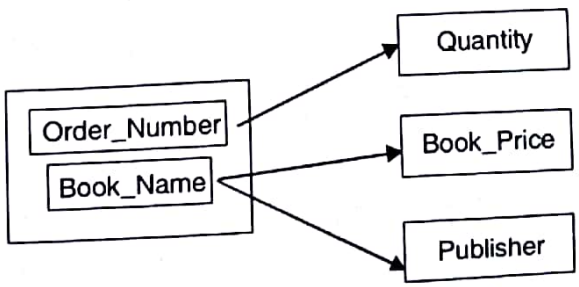
Here, following FD exist:

(Order_number, Book_Name) ⟶ Quantity

Book_Name ⟶ Book_Price

Book_Name ⟶ Publisher

Solution: The given relation is in first normal form. The primary key of the above relation is (Order_number, Book_Name) combination and the Functional Dependence diagram is given below.

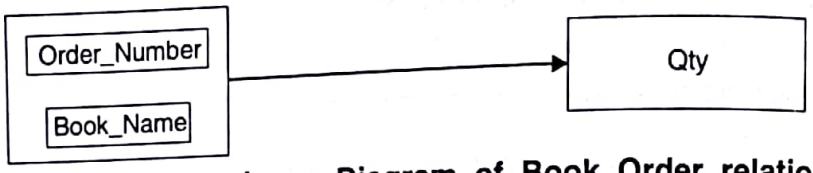


Functional Dependence Diagram of Order_Book relation

From the above FD diagram, it is clear that non-key attribute "Quantity " is fully functional dependent on primary key attributes (Order_number, Book_Name). But, the attributes "Book_Price" and "Publisher" are not fully functional dependent on the primary key, because they are functional dependent on the subset of the primary key i.e. "Book_Name".

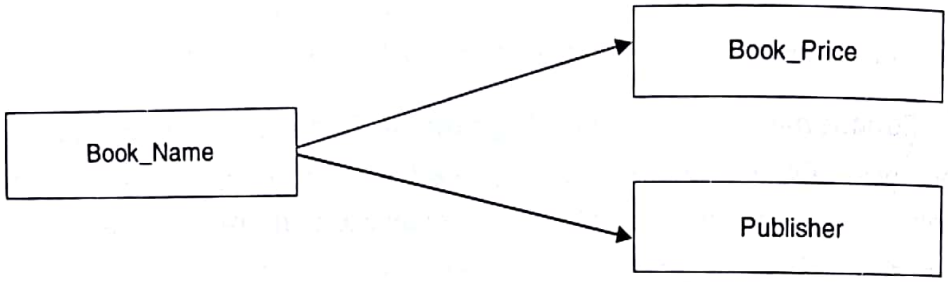
Thus, in order to convert this relation to 2NF it is decomposed into two relations Book_Order and Book according to the transformation rule discussed in the Section 8.4. The functional dependence diagrams of two tables are as follows:

Book_Order (Order_Number, Book_Name, Qty)



Functional Dependence Diagram of Book_Order relation

Book (Book_Name, Book_Price, Publisher)



Functional Dependence Diagram of Book relation

The above relations have no anomalies and there is no case of transitive dependence. Thus, there is no need for application of 3rd normal form rule and database is already normalized in the 2nd form. Thus, the relation is normalized by decomposing it into two relations Book_Order and Book as shown below with primary keys underlined and the above database is normalized up to 2nd normal form.

Book_Order (Order Number, Book Name, Qty)

Book (Book Name, Book_Price, Publisher)

8.12.4 Challan Case Study

Study the following database of Challan relation carefully and normalize it.

Name	License	Offense	Fine	Date	ChallanNo
Ajay	L100	Parking	50	10.10.05	1000
Raj	L101	Red Light	100	12.10.5	1001
Ajay	L100	Red Light	100	13.10.5	1002
Rahat	L102	Backing-in	75	14.10.04	900
Raj	L101	Splitting	50	15.10.5	1003

(i) Identify the functional dependence in the above database.

(ii) Indicate the final tables after 1NF, 2NF ,3NF and so on if applicable. Clearly indicate all the intermediate steps followed during process of normalization.

Solution: Functional Dependencies in the original relation Challan are as follows:

ChallanNo ⟶ Name

ChallanNo ⟶ License

ChallanNo ⟶ Offence

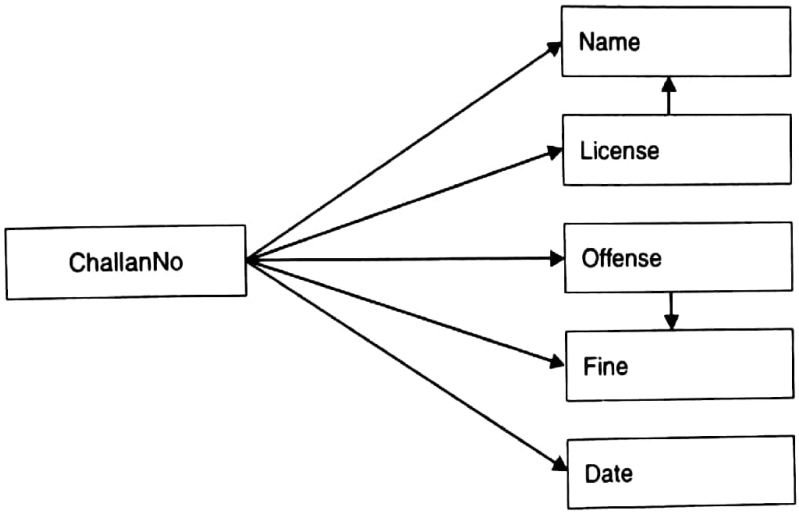
ChallanNo ⟶ Fine

ChallanNo ⟶ Date

License ⟶ Name

Offense ⟶ Fine

The given relation is in first normal form, because for every-row column combination there is at most single value. The primary key of the above relation is "ChallanNo" and the Functional Dependence diagram of relation Challan is as follows.



Functional Dependence Diagram of Challan relation

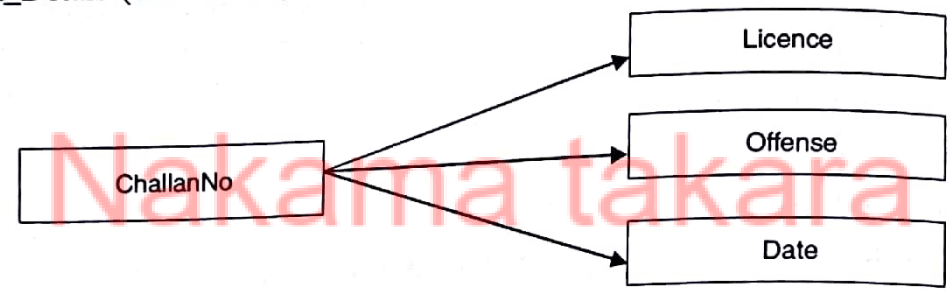
Here, "ChallanNo" is the primary key and it functionally determines all other non-key attributes. The above FD diagram, is in 2nd form, here primary is a simple, single attribute "ChallanNo", so all other non-key attributes are fully-functional dependent on the primary key, because there is no chance of depending on the subset of primary key (because it does not has any).

The functional dependence of attribute "Name" on "License" and functional dependence of attribute "Fine" on "Offence", (because, for a given offence there should be single value of fine and for each license number there will only one name) results in the case of transitive dependence.

Here, one non key attribute "License" functionally determines other non-key attribute "Name" and another non key attribute "Offense" functionally determines other non-key attribute "Fine", which results transitive dependence, because now the attribute "Name" functionally dependent on primary key two ways one directly and other through or transitively by "License". The same case is applicable for attribute "Fine" and "Offense".

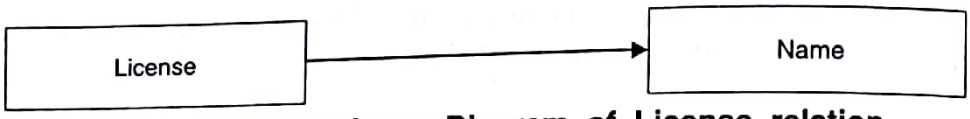
The transitive dependence is removed by decomposing the relation into three relations Challan_Detail, License and Fine as shown below. These, functional dependence diagrams are given below.

Challan_Detail (ChallanNo, License, Offense, Date)



Functional Dependence Diagram of Challan_Detail relation

License (License, Name)



Functional Dependence Diagram of License relation

Offense (Offense, Fine)

Functional Dependence Diagram of Offense relation

Thus, the given table Challan is normalized by decomposing it into three tables as shown below with primary key underlines. The given database is normalized up to 3rd normal form.

Challan_Detail (ChallanNo, License, Offense, Date)

License (License, Name)

Offense (Offense, Fine)

8.12.5 Car Hire Services Case Study

The table Car_Hire shown below, lists customer/car hires data. Each customer may hire cars from various outlets. A car is registered at a particular outlet and can be hired out to a customer on a given date.

CarRegNo	Make	Model	CustNo	CustName	HireDate	OutletNo	OutletLoc
H235 EFG	Honda	Civic	C302	Shirish M.	24/2/09	101	Pune
H235 EFG	Honda	Civic	C501	Nitin D.	12/2/09	101	Pune
C423 RCV	Chevy	Optra	C302	Shirish M.	27/2/09	101	Pune
H427 SHF	Honda	Civic	C423	Anoop B.	24/2/09	104	Mumbai
H427 SHF	Honda	Civic	C302	Shirish M.	18/3/09	104	Mumbai
C581 BJT	Chevy	Optra	C115	Hardik S.	18/3/09	104	Mumbai

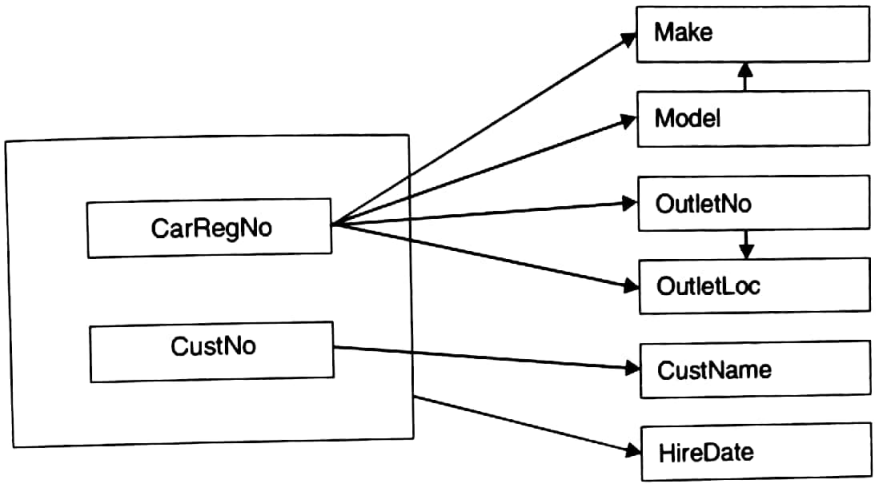
(i) Identify the functional dependencies represented by the data shown in the table.

(ii) Using the functional dependencies identified in part (i), describe and illustrate the process of normalization by converting above table to Third Normal Form (3NF) relations.

Solution: Functional Dependencies in the given relation are as follows:

CarRegNo ⟶ Make
CarRegNo ⟶ Model
CustNo ⟶ CustName
CarRegNo,CustNo ⟶ HireDate
CarRegNo ⟶ OutletNo
OutletNo ⟶ OutletLoc
Model ⟶ Make

The given relation is in first normal form, because for every-row column combination there is at most single value. The primary key of the above relation is (CarRegNo,CustNo) combination and the Functional Dependence diagram is as follows:

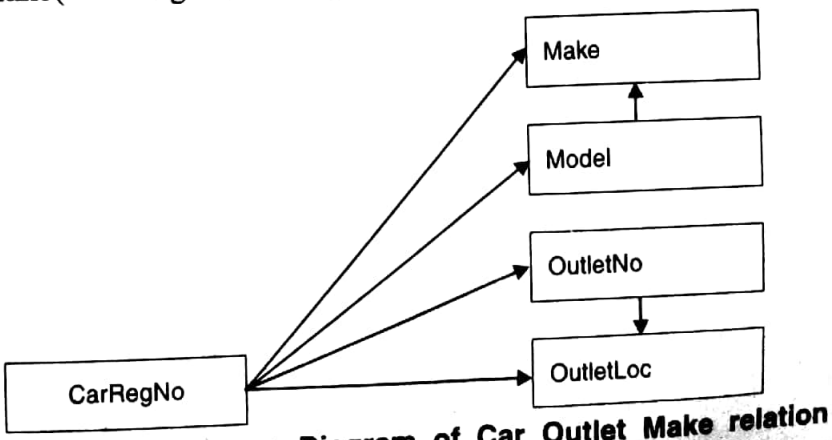


Functional Dependence Diagram of Car_Hire relation

From the above FD diagram, it is clear that only one non key attribute "HireDate" is fully-functional dependent on primary key attributes (CarRegNo, CustNo). But, all other non key attributes are not fully functional dependent on the primary key because they are dependent on the subset of the primary key as shown in the functional dependence diagram.

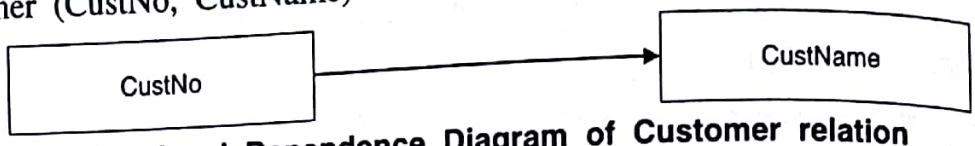
Thus, in order to convert this relation to 2NF it is decomposed into three tables Car_Outlet_Make, Customer and Car_Cust_Hire. The functional dependence diagrams of three tables are as follows:

Car_Outlet_Make(CarRegNo, Make, Model, OutletNo, OutletLoc)



Functional Dependence Diagram of Car_Outlet_Make relation

Customer (CustNo, CustName)



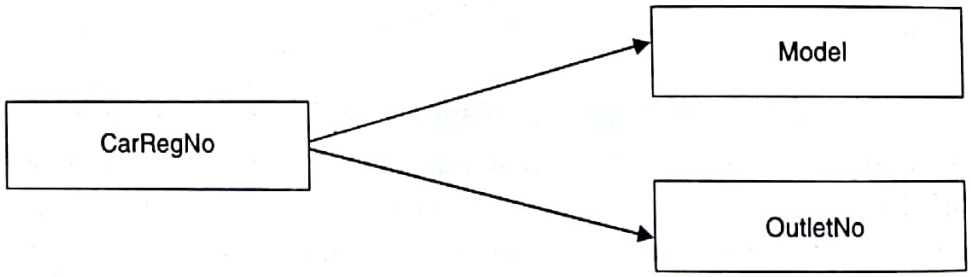
Functional Dependence Diagram of Customer relation

Car_Cust_Hire (CarRegNo, CustNo, HireDate)

Functional Dependence Diagram of Car_Cust_Hire relation

Both, Customer and Car_Cust_Hire relations are normalized. They do not contain any anomalies and do not have any case of transitive dependence. Thus, both the tables are normalized in the 2nd normal form. But the relation Car_Outlet_Make contains the case of transitive dependence. The functional dependence of attribute "Make" on "Model" and functional dependence of attribute "OutletLoc" on "OutletNo" results the case of transitive dependence. The transitive dependence in Car_Outlet_Make relation is removed by decomposing the relation into three relations Car_Outlet_Model, Outlet and Car_Model as shown below. There, functional dependence diagrams are given below.

Car_Outlet_Model (CarRegNo, Model, OutletNo)



Functional Dependence Diagram of Car_Outlet_Model relation

Outlet (OutletNo, OutletLoc)

Functional Dependence Diagram of Outlet relation

Car_Model (Model, Make)

Functional Dependence Diagram of Car_Model relation

Thus, the given original table Car_Hire normalized by decomposing it into five tables as shown below with primary key underlined. The database is normalized in the 3rd normal form.

Customer (<u>CustNo</u>, CustName)

Car_Cust_Hire (<u>CarRegNo, CustNo</u>, HireDate)

Car_Outlet_Model (<u>CarRegNo</u>, Model, OutletNo)

Outlet (OutletNo, OutletLoc)

Car_Model (Model, Make)

8.12.6 Film Actor Role Case Study

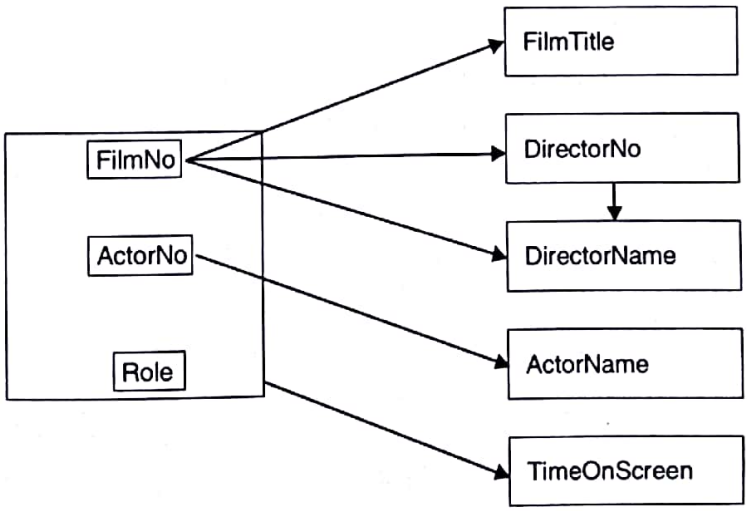
The table Film shown below displays the details of the roles played by actors/actresses in films. Normalize the database by showing all the intermediate steps.

FilmNo	FilmTitle	Director No	Director Name	Actor No	Actor Name	Role	TimeOn Screen
F2132	K3G	D202	Karan Johar	A1172	Shahrukh	Raj	13:45
		D202	Karan Johar	A2531	Hrithik	Rohan	18:34
		D202	Karan Johar	A1172	Shahrukh	Rahul	20:29
F2472	Dil To Pagal Hai	D152	Yash chopra	A1172	Shahrukh	Aman	11:23
		D152	Yash chopra	A2371	Akshay	Karan	16:52

Solution: The given table is not in first normal form, so apply the flattening approach on the original table to convert into first normal form. The resultant first normal form relation is shown below:

FilmNo	FilmTitle	Director No	Director Name	Actor No	Actor Name	Role	TimeOn Screen
F2132	K3G	D202	Karan Johar	A1172	Shahrukh	Raj	13:45
F2132	K3G	D202	Karan Johar	A2531	Hrithik	Rohan	18:34
F2132	K3G	D202	Karan Johar	A1172	Shahrukh	Rahul	20:29
F2472	Dil To Pagal Hai	D152	Yash chopra	A1172	Shahrukh	Aman	11:23
F2132	K3G	D152	Yash chopra	A2371	Akshay	Karan	16:52

The primary key of Film relation is combination of (FilmNo,ActorNo,Role). The functional dependence of the table is as follows.



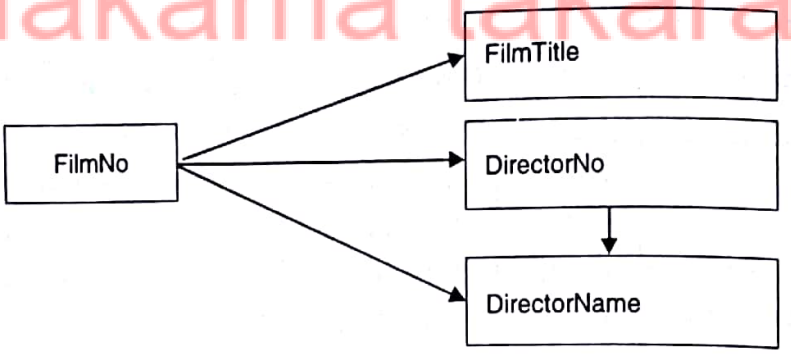
Functional Dependence Diagram of Film relation

From the above FD diagram, it is clear that only one non key attribute "TimeOnScreen" is fully-functional dependent on primary key attributes (FilmNo,ActorNo,Role). But, all other

non key attributes are not fully functional dependent on the primary key because they are dependent on the subset of the primary key as shown in the functional dependence diagram.

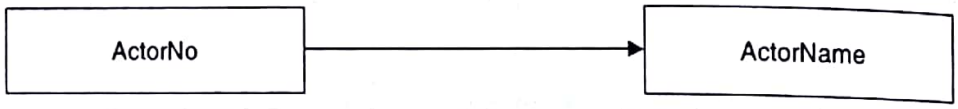
Thus, in order to convert this relation to 2NF it is decomposed into three tables Film_Detail, Actor and Film_Actor_Role. The functional dependence diagrams of three tables are as follows:

Film_Detail (FilmNo, FilmTitle, DirectorNo, DirectorName)



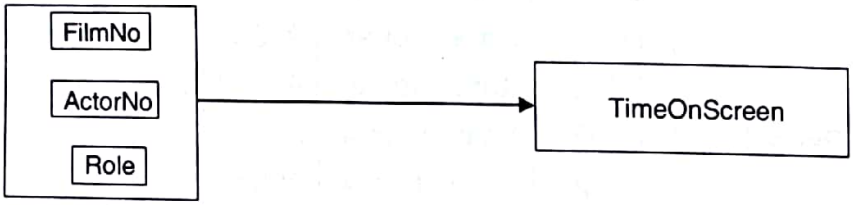
Functional Dependence Diagram of Film_Detail relation

Actor (ActorNo, ActorName)



Functional Dependence Diagram of Actor relation

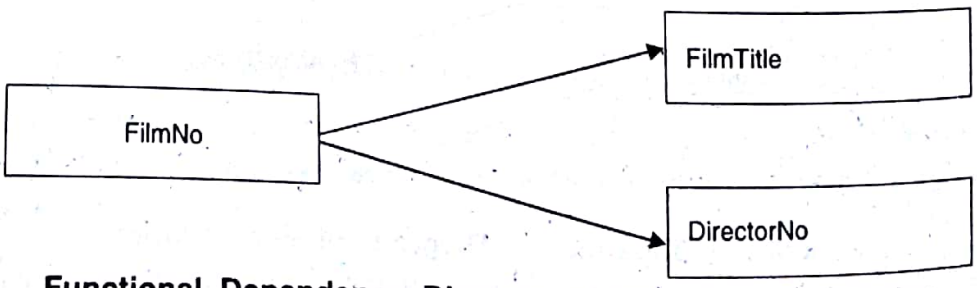
Film_Actor_Role (FilmNo, ActorNo, Role, TimeOnScreen)



Functional Dependence Diagram of Film_Actor_Role relation

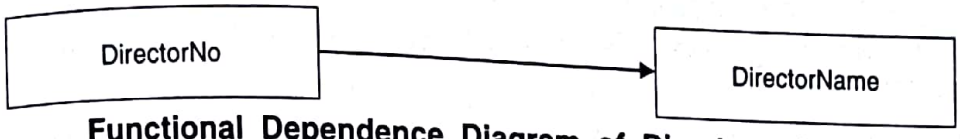
Both, Actor and Film_Actor_Role relations are normalized. They do not contain any anomalies and do not have any case of transitive dependence. Thus, both the tables are normalized in the 2nd normal form. But the relation Film_Detail contains the case of transitive dependence. The functional dependence of attribute "DirectorName" on "DirectorNo" results the case of transitive dependence. The transitive dependence in Film_Detail relation is removed by decomposing the relation into two relations Film_Director and Director as shown below. There, functional dependence diagrams are given below.

Film_Director (FilmNo, FilmTitle, DirectorNo)



Functional Dependence Diagram of Film_Director relation

Director(DirectorNo, DirectorName)



Functional Dependence Diagram of Director relation

Thus, the given original table Film is normalized by decomposing it into five tables as shown below with primary key underlined. The database is normalized in the 3rd normal form.

Actor (ActorNo, ActorName)

Film_Actor_Role (FilmNo, ActorNo, Role, TimeOnScreen)

Film_Director (FilmNo, FilmTitle, DirectorNo)

Director(DirectorNo, DirectorName)

Flash Back

Normalization is a design technique that is widely used in designing relational model. The database is designed so that basic operations on the database like Insert, Update, Delete or Retrieve can be performed without any problems. Normalization of data can be defined as a process during which redundant relation schemas are decomposed by breaking up their attributes into smaller relation schemas that possess desirable properties.

The entire normalization process is based upon the analysis of relations, their schemes, their primary keys and their functional dependencies. Initially E.F. Codd proposed three normal forms known as first, second, and third normal form. A relation is said to be in a particular normal form if it satisfies a certain specified set of constraints. A functional dependency is an association between two attributes of the same relational database table. Fully Functional Dependence (FFD) is defined, as Attribute Y is FFD on attribute X, if it is FD on X and not FD on any proper subset of X.

A relation is said to be in First Normal Form (1NF) if and only if every entry of the relation (the intersection of a tuple and a column) has at most a single value. In other words "a relation is in First Normal Form if and only if all underlying domains contain atomic values or single value only."

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

A relation R is in Third Normal Form (3NF) if and only if the following conditions are satisfied simultaneously: I) R is already in 2NF (II) No nonprime attribute is transitively dependent on the key.

Transitive dependence occurs if A, B and C are the set of attributes of a relation R and following functional dependencies are satisfied simultaneously: A→B, B→A (B not functionally depends A), B→C, A→C and C→A (C not functionally depends A). Observe that C→B is neither prohibited nor required. If all these conditions are true, we will say that attribute C is transitively dependent on attribute A.

BCNF is simply a stronger defination of 3NF. BCNF makes no explictly reference to first and second normal form as such, nor the concpet of full and transitive dependence. A relation R is in Boyce/Codd N/F (BCNF) if and only if every determinant is a candidate key. Here, determinant is a simple attribute or composite attribute on which some other attribute is fully functionally dependent.

A relation R is in Fourth Normal Form (4NF) if and only if the following conditions are satisfied simultaneously: (I) R is already in 3NF or BCNF. (II) If it contains no multi-valued dependencies.

MVD is the dependency where one attribute value is potentially a 'multi-valued fact' about another. MVDs occur when two or more independent multi valued facts about the same attribute occur within the same table. It means that if in a relation R having A, B and C as attributes, B and C are multi-value facts about A, which is represented as A $\longrightarrow$ B and A $\longrightarrow$ C ,then muti value dependency exist only if B and C are independent on each other.

A relation R is in Fifth Normal Form (5NF) if and only if the following conditions are satisfied simultaneously: (I) R is already in 4NF. (II) It cannot be further non-loss decomposed. NF is of little practical use to the database designer, but it is of interest from a theoretical point of view and a discussion of it is included here to complete the picture of the further normal forms. A table T is in fifth normal form if every join dependency in T is a consequence only of the candidate keys of T.

REVIEW QUESTIONS

1. What are the objectives to normalize the database?
2. What is meant by normalization? Why we normalize the database? Explain different techniques.
3. Explain the concept of functional dependence and fully function dependence with examples.
4. What are the possible dependencies, which can exist in a database?
5. What are the basic normalization techniques? Explain each with suitable database.
6. How the anomalies of First Normal Form are rectified? Explain with suitable examples.
7. How the anomalies of Second Normal Form are rectified? Explain with suitable examples.
8. How we convert a database in First Normal Form to Second Normal Form?
9. How we convert a database in Second Normal Form to Third Normal Form?
10. Explain the concept of transitive dependence with appropriate example.
11. What is overlapping of candidate keys? What are problems incurred due to this overlap and how these problems are rectified?
12. Explain the similarities and dissimilarities between BCNF and 3rd Normal Form.
13. Why BCNF is stronger than 3rd Normal Form?
14. Explain the cases where 4th Normal Form is applicable. How it removes the anomalies?
15. What are the anomalies occurred due multi-value dependence and how the anomalies are rectified?
16. Normalize the following database:
    (empno, ename, job, sal, doj, mgr, deptno, dname, loc)
17. Normalize the following database:
    (Sno, Sname, Pno, Pname, Qty)
18. What is meant by Projection Join dependency?
19. Explain the importance of 5th Normal Form?

BRAIN STORMING SESSION

1. Explain what is meant by repetition of information and inability to represent information. Explain why each of these properties may indicate a bad relational database design.
2. Given the three goals of relational database design, is there any reason to design a database schema that is in 2NF, but is in no higher-order normal form?
3. Explain why 4NF is a normal form more desirable than BCNF.
4. The table below represents data about employees of a company and the projects they work on. An employee may work on one or more projects a certain number of hours.

Emp Projects

ProjName	EmpNo.	ProjBudget	hours	projManager	managerDOB
Hardware	E1	20000	20	Smith	Jan 63
Hardware	E3	20000	20	Smith	Jan 63
AI	E2	17000	42	Adams	March 52
Statellite	E2	84000	42	Smith	Jan 63
AI	E3	17000	17	Adams	March 52
Hardware	E2	20000	83	Smith	Jan 63
Software	E4	84000	41	Adams	March 52
Cancer	E5	66000	42	Jones	Jan 63

Assuming that the functional dependencies will hold for any additional data, which of the following functional dependencies are true and which are false? Justify your answer.

(i) projName ⟶ empNo (ii) projName ⟶ projBudget
(iii) projBudget ⟶ projName (iv) projName ⟶ hours

HANDS ON SESSION

1. Normalization is used
   (a) To remove insertion & deletion anomalies
   (b) To replace data
   (c) To add new data (d) None
2. A relation is in Normal form if
   (a) it satisfies set of constraints (b) it reduces dependencies
   (c) it introduces dependencies (d) Both a & b
3. A relation is said to be in 1NF if
   (a) domains contain atomic values (b) domains contain non atomic values
   (c) it contains duplicate values (d) None
4. The normalization can be carried out
   (a) Flattening the table (b) Decomposition & the table
   (c) Both a & b (d) None
5. A relation is said to be in 2NF if
   (a) it is in 1NF
   (b) non key attribute fully dependent on primary key
   (c) Both a & b (d) None

6. A relation is said to be in 3NF if
   (a) it is in 2NF
   (b) non key attribute fully dependent on primary key
   (c) no non prime attribute is transitively dependant on key
   (d) Both a & c

7. BCNF refers to
   (a) Atomicity (b) Primary key
   (c) Overlapping Candidate key (d) None
8. 4NF refers to
   (a) Foreign key (b) Primary key
   (c) multi-valued dependencies (d) None
9. 5NF refers to
   (a) Non-loss decomposition (b) Super key
   (c) Flattening of table (d) None
10. If no multivalued attributes exist in a relation, then the relation is in what normal form?
    (a) First normal form (b) Second normal form
    (c) Third normal form (d) Fourth normal form
11. If no multivalued attributes exist and no partial dependencies exist in a relation, then the relation is in what normal form?
    (a) First normal form (b) Second normal form
    (c) Third normal form (d) Fourth normal form
12. A transitive dependency is which of the following?
    (a) A functional dependency between two or more key attributes.
    (b) A functional dependency between two or more nonkey attributes.
    (c) A relation that is in first normal form.
    (d) A relation that is in second normal form.
13. If a denormalization situation exists with a one-to-one binary relationship, which of the following is true?
    (a) All fields are stored in one relation. (b) All fields are stored in two relations.
    (c) All fields are stored in three relations. (d) All fields are stored in four relations.
14. If a denormalization situation exists with a many-to-many or associative binary relationship, which of the following is true?
    (a) All fields are stored in one relation. (b) All fields are stored in two relations.
    (c) All fields are stored in three relations. (d) All fields are stored in four relations.
15. Consider the table (Relation) CARS. The key is REGNO (car registration number), OID is a foreign key that identifies the car's owner.
    REGNO
    (Key Field)
    MODEL

COLOUR

OID

This table is in

(a) third normal form
(b) second normal form
(c) first normal form
(d) cannot be determined

16. The rule that specifies that there should be no repeating fields and that fields should be atomic is

(a) first normal form
(b) third normal form
(c) second normal form
(d) fourth normal form

Solution Keys

1. (a)	2. (d)	3. (a)	4. (c)	5. (c)	6. (d)	7. (c)	8. (c)	9. (a)
10. (a)	11. (b)	12. (b)	13. (a)	14. (b)	15. (a)	16. (a)		

●●●