UNIT 2 NORMALIZATION

Structure

2.0 INTRODUCTION

The basic objectives of normalization is to reduce redundancy which means that information is to be stored only once. Storing information several times leads to wastage of storage space and increase in the total size of the data stored. Relations are normalized so that when relations in a database are to be altered during the life time of the database, we do not lose information or introduce inconsistencies. The type of alterations normally needed for relations are:

1. **Insertion** of new data values to a relation. This should be possible without being forced to leave blank fields for some attributes.
2. **Deletion** of a tuple, namely, a row of a relation. This should be possible without losing vital information unknowingly.
3. **Updating** or changing a value of an attribute in a tuple. This should be possible without exhaustively searching all the tuples in the relation.

In this unit, in the beginning we discuss the importance of having a consistent database without repetition of data and points out the anomalies that could be introduced in a database with an undesirable scheme. Then we discuss the different forms of normalization.

2.1 OBJECTIVES

After going through this unit, you may be able to:

- discuss the different types of anomalies in a database
- state what is functional dependency
- list the different forms of normalization
- differentiate among different types of normalization.

2.2 FUNCTIONAL DEPENDENCY

As the concept of dependency is very important, it is essential that we first understand it well and then proceed to the idea of normalization. There is no fool-proof algorithmic method of identifying dependency. We have to use our commonsense and judgement of specify dependencies.

Let X and Y be two attributes of a relation. Given the value of X, if there is only one value of Y corresponding to it, then Y is said to be functionally dependent on X. This is indicated by the notation:

$$X \rightarrow Y$$

For example, given the value of item code, there is only one value of item name for it. Thus item name is functionally dependent on item code. This is shows as:

Item code $\rightarrow$ item name

Similarly in table 1, given an order number, the date of the order is known. Thus : order no. Order date

Functional dependency may also be based on a composite attribute. For example, if we write

$$X, Z \rightarrow Y$$

it means that there is only one value of Y corresponding to given values of X, Z. In other words, Y is functionally dependent on the composite X, Z. In other words, Y is functionally dependent on the composite X, Z. In table 1 mentioned below, for example, Order no., and Item code together determine Qty. and Price. Thus :

Order no., Item code $\rightarrow$ Qty., Price

As another example, consider the relation

Student (**Roll no.,** Name, Address, Dept., Year of study)

Order no.	*Order date*	*Item code*	*Quantity*	*Price/unit*
1456	260289	3687	52	50.40
1456	260289	4627	38	60.20
1456	260289	3214	20	17.50
1886	040389	4629	45	20.25
1886	040389	4627	30	60.20
1788	040489	4627	40	60.20

Table 1: Normalized Form of the Relation

In this relation, Name is functionally dependent on Roll no. In fact, given the value of Roll no., the values of all the other attributes can be uniquely determined. Name and Department are not functionally dependent because given the name of a student, one cannot find his department uniquely. This is due to the fact that there may be more than one student with the same name. Name in this case is not a key. Department and Year of study are not functionally dependent as Year of study pertains to a student whereas Department is an independent attribute. The functional dependency in this relation is shown in the following figure as a dependency diagram. Such dependency diagrams shown in figure 1 are very useful in normalization.

Relation Key: Consider the relation of table 1. Given the Vendor code, the Vendor name and Address are uniquely determined. Thus Vendor code is the relation key. Given a relation, if the value of an attribute X uniquely determines the values of all other attributes in

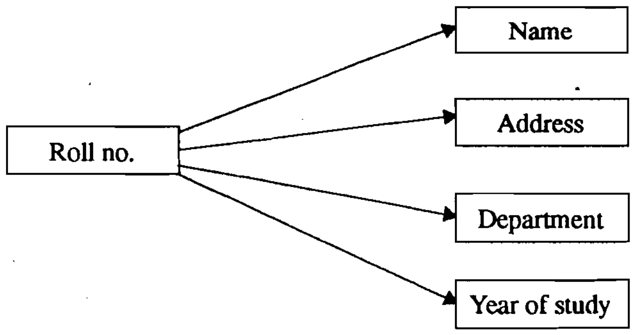

Figure 1: Dependency diagram for the relation "Student"

a row, then X is said to be the key of that relation. Sometimes more than one attribute is needed to uniquely determine other attributes in a relation row. In that case such a set of attributes is the key. In table 1, Order no. and Item code together form the key. In the relation "Supplies" (Vendor code, Item code, Qty. supplied, Date of supply, Price/unit), Vendor code and Item code together form the key. This dependency is shown in the following diagram (figure 2).

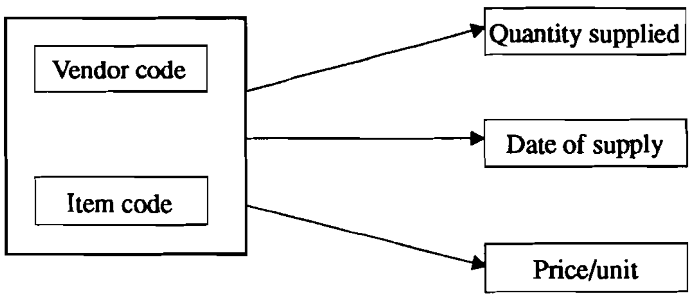

Figure 2: Dependency diagram for the relation "Supplies"

Observe that in the figure the fact that Vendor code and Item code together form a composite key is clearly shown by enclosing them together in a rectangle.

2.3 ANOMALIES IN A DATABASE

Consider the following relation scheme pertaining to the information about a student maintained by a university:

STDINF(Name, Course, Phone_No, Major, Prof, Grade)

Table 2 shows some tuples of a relation on the relation scheme **STDINF(Name, Course,** Phone_No, Major, Prof, Grade). The functional dependencies among its attributes are shown in Figure 3. The key of the relation is Name Course and the relation has, in addition, the following functional dependencies {Name → Phone_No, Name → Major, Name Course → Grade, Course → Prof}.

Name	Course	Phone_No	Major	Prof	Grade
Jones	353	237-4539	Comp Sci	Smith	A
Ng	329	427-7390	Chemistry	Turner	B
Jones	328	237-4539	Comp Sci	Clark	B
Martin	456	388-5183	Physics	James	A
Dulles	293	371-6259	Decision Sci	Cook	C
Duke	491	823-7293	Mathematics	Lamb	B
Duke	356	823-7293	Mathematics	Bond	in prog
Jones	492	237-4539	Comp Sci	Cross	in prog
Baxter	379	839-0827	English	Broes	C

Table 2: Student Data Representation in Relation STDINF

Here the attribute Phone_No, which is not in any key of the relation scheme STDINF, is not functionally dependent on the whole key but only one part of the key, namely, the attribute Name. Similarly, the attributes Major and Prof, which are not in any key of the relation scheme **STDINF** either, are fully functionally dependent on the attributes Name and Course, respectively. Thus the determinants of these functional dependencies are again not the entire key but only part of the key of the relation. Only the attribute Grade is fully functionally dependent on the key **Name Course.**

The relation scheme **STDINF** can lead to several undesirable problems:

- **Redundancy:** The aim of the database system is to reduce redundancy, meaning that information is to be stored only once. Storing information several times leads to the waste of storage space and an increase in the total size of the data stored.

Updates to the database with such redundancies have the potential of becoming inconsistent, as explained below. In the relation of table 2, the Major and Phone_No. of a student are stored several times in the database: once for each course that is or was taken by a student.

- **Update Anomalies:** Multiple copies of the same fact may lead to update anomalies or inconsistencies when an update is made and only some of the muliple copies are updated. Thus, a change in the Phone_No. of Jones must be made, for consistency, in all tuples pertaining to the student Jones. If one of the three tuples of Figure 3 is not changed to reflect the new Phone_No. of Jones, there will be an inconsistency in the data.

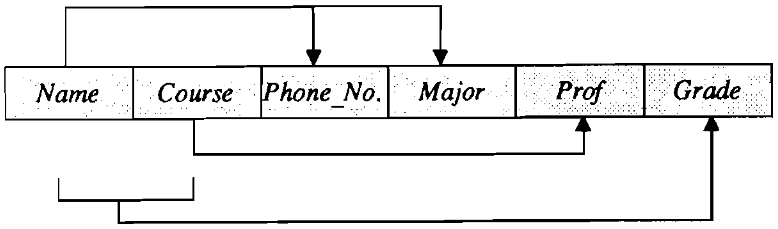

Figure 3: Function dependencies in STDINF

- **Insertion Anomalies:** If this is the only relation in the database showing the association between a faculty member and the course he or she teaches, the fact that a given processor is teaching a given course cannot be entered in the database unless a student is registered in the course. Also, if another relation also establishes a relationship between a course and a professor who teaches that course the information stored in these relations has to be consistent.

- **Deletion Anomalies:** If the only student registered in a given course discontinues the course, the information as to which professor is offering the course will be lost if this is the only relation in the database showing the association between a faculty member and the course she or he teaches. If another relation in the database also establishes the relationship between a course and a professor who teaches that course, the deletion of the last tuple in STDINF for a given course will not cause the information about the course's teacher to be lost.

The problems of database inconsistency and redundancy of data are similar to the problems that exist in the hierarchical and network models. These problems are addressed in the network model by the introduction of virtual fields and in the hierarchical model by the introduction of virtual records. In the relational model, the above problems can be remedied by decomposition. We define decomposition as follows:

Definition: Decomposition

The decomposition of a relation scheme $R = (A_1, A_2, \ldots A_n)$is its replacement by a set of relation schemes $\{R_1, R_2, \ldots R_m\}$, such that $R_1 \leq R$ for $1 \leq i \leq m$ and $R_1 \cup R_2 \cup R_m = R$.

A relation scheme R can be decomposed into a collection of relation schemes $\{R_1, R_2, R_3 \ldots R_m\}$ to eliminate some of the anomalies contained in the original relation R. Here the relation schemes R_1 $(1 \leq i \leq m)$ are subsets of R and the intersection of $R_1 \cap R_j$ for $i \neq j$ need not be empty. Furthermore, the union of R_j $(1 \leq i \leq m)$ is equal to R, i.e. $R = R_1 \; R_2 \ldots R_m$.

The problems in the relation scheme STDINF can be resolved if we replace it with the following relation schemes:

STUDENT_INFO (Name, Phone_No, Major)

TRANSCRIPT (Name, Course, Grade)

TEACHER (Course, Prof)

The first relation scheme gives the phone number and the major of each student and such information will be stored only once for each student. Any change in the phone number will thus require a change in only one tuple of this relation.

The second relation scheme stores the grade of each student in each course that the student is or was enrolled in. (Note: In our database we assume that either the student takes the course only once, or if he or she has to repeat it to improve the grade, the TRANSCRIPT relation stores only the highest grade.)

The third relation scheme records the teacher of each course. One of the disadvantages of replacing the original relation scheme STDINF with the three relation schemes is that the retrieval of certain information requires a natural join operation to be performed. For instance, to find the majors of a student who obtained a grade of A in course 353 requires a join to be performed: (STUDENT_INFO $\bowtie$ TRANSCRIPT). The same information could be derived from the original relation STDINF by selection and projection.

When we replace the original scheme STDINF with the relation schemes STUDENT_INFO, TRANSCRIPT and TEACHER, the consistency and referential integrity constraints have to be enforced. The referential integrity enforcement implies that if a tuple in the relation TRANSCRIPT exists, such as (Jones, 353, in prog), a tuple must exist in STUDENT_INFO with Name = Jones and furthermore, a tuple must exist in STUDENT_INFO with Course = 353. The attribute Name, which forms part of the key of the relation TRANSCRIPT, is a key of the relation STUDENT_INFO. Such an attribute (or a group of attributes), which establishes a relationship between specific tuples (of the same or two distinct relations), is called a foreign key. Notice that the attribute Course in relation TRANSCRIPT is also a foreign key, since it is a key of the relation TEACHER.

Note that the decomposition of STDINF into the relation schemes STUDENT (Name, Phone_No, Major, Grade) and COURSE (Course, Prof.) is a bad decomposition for the following reasons:

1. Redundancy and update anomaly, because the data for the attributes Phone_no and Major are repeated.
2. Loss of information, because we lose the fact that a student has a given grade in a particular course.

2.4 PROPERTIES OF NORMALIZED RELATIONS

Ideal relations after normalization should have the following properties so that the problems mentioned above do not occur for relations in the (ideal) normalized form:

1. No data value should be duplicated in different rows unnecessarily.
2. A value must be specified (and required) for every attribute in a row.
3. Each relation should be self-contained. In other words, if a row from a relation is deleted, important information should not be accidentally lost.
4. When a row is added to a relation, other relations in the database should not be affected.
5. A value of an attribute in a tuple may be changed independent of other tuples in the relation and other relations.

The idea of normalizing relations to higher and higher normal forms is to attain the goals of having a set of ideal relations meeting the above criteria.

2.5 FIRST NORMALIZATION

The relation shown in table 1 is said to be in **First Normal Form**, abbreviated as 1NF. This form is also called a **flat file**. There are no composite attributes, and every attribute is single and describes one property.

Converting a relation to the 1NF form is the first essential step normalization. There are successive higher normal forms known as 2NF, 3NF, BCNF, 4NF and 5NF. Each form is an improvement over the earlier form. In other words, 2NF is an improvement on 1NF, 3NF is an improvement on 2NF, and so on. A higher normal form relation is a subset of lower normal form as shown in the following figure 4. The higher normalization steps are based on three important concepts:

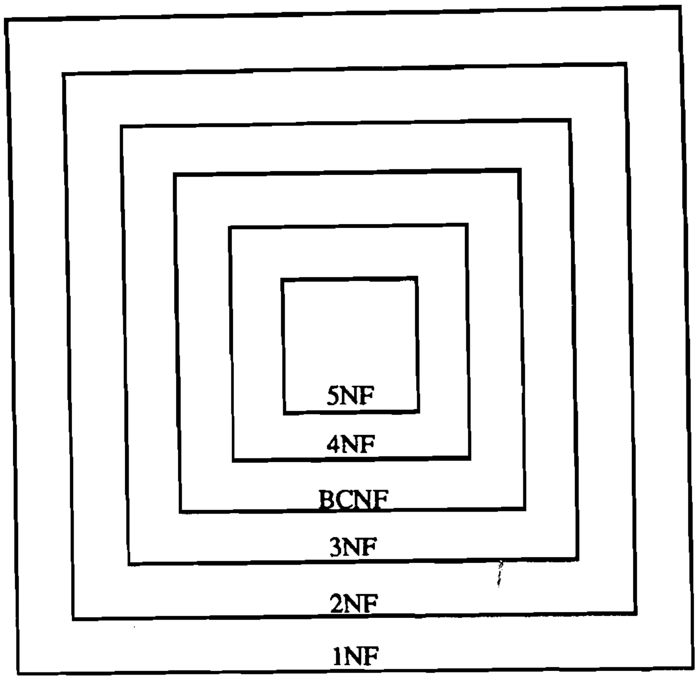

Figure 4: Illustration of successive normal forms of a relation

1. Dependence among attributes in a relation
2. Identification of an attribute or a set of attributes as the key of a relation
3. Multivalued dependency between attributes

2.6 SECOND NORMAL FORM RELATION

We will now define a relation in the Second Normal Form (2NF). A relation is said to be in 2NF if it is in 1NF and non-key attributes are functionally dependent on the key attribute(s). Further, if the key has more than one attribute then no non-key attributes should be functionally dependent upon a part of the key attributes. Consider, for example, the relation given in table 1. This relation is in 1NF. The key is (Order no., Item code). The dependency diagram for attributes of this relation is shown in figure 5. The non-key attribute Price/Unit is functionally dependent on Item code which is part of the relation key. Also, the non-key attribute Order date is functionally dependent on Order no. which is a part of the relation key. Thus the relation is not in 2NF. It can be transformed to 2NF by splitting it into three relations as shown in table 3.

In table 3 the relation **Orders** has **Order no.** as the key. The relation **Order details** has the composite key **Order no.** and **Item code.** In both relations the non-key attributes are functionally dependent on the whole key. Observe that by transforming to 2NF relations the

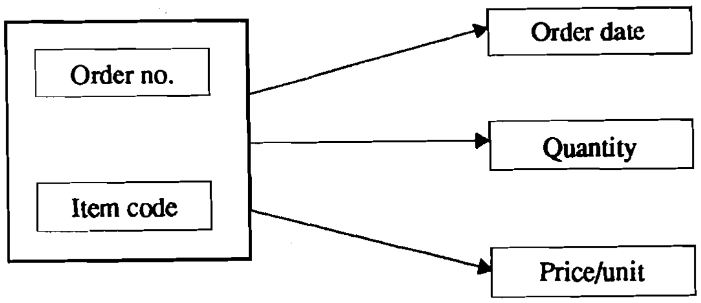

Figure 5: Dependency diagram for the relation given in table

repetition of Order date (table 1) has been removed. Further, if an order for an item is cancelled, the price of an item is not lost. For example, if Order no. **1886** for Item code **4629** is cancelled in table 1, then the fourth row will be removed and the price of the item is lost. In table 3 only the fourth row of the table 3(b) is omitted. The item price is not lost as it is available in table 3(c). The data of the order is also not lost as it is in table 3(a).

(a) Orders

Order no	*Order date*
1456	260289
1886	040389
1788	040489

(b) Order Details

Order no.	Item Code	Qty.
1456	3687	52
1456	4627	38
1456	3214	20
1886	4629	45
1886	4627	30
1788	4627	40

(c) Prices

Item code	Price/ unit
3687	50.40
4627	60.20
3214	17.50
4629	20.25

Table 3: Splitting of Relation given in table 1 into 2NF Relations

These relations in 2NF form meet all the "ideal" conditions specified. Observe that the three relations obtained as self-contained. There is no duplication of data within a relation.

2.7 THIRD NORMAL FORM

A Third Normal Form normalization will be needed where all attributes in a relation tuple are not functionally dependent only on the key attribute. If two non-key attributes are functionally dependent, then there will be unnecessary duplication of data. Consider the relation given in table 4. Here, Roll no. is the key and all other attributes are

Roll no.	Name	Department	Year	Hostel name
1784	Raman	Physics	1	Ganga
1648	Krishnan	Chemistry	1	Ganga
1768	Gopalan	Mathematics	2	Kaveri
1848	Raja	Botany	2	Kaveri
1682	Maya	Geology	3	Krishna
1485	Singh	Zoology	4	Godavari

Table 4: A 2NF Form Relation

functionally dependent on it. Thus it is in 2NF. If it is known that in the college all first year students are accommodated in Ganga hostel, all second year students in Kaveri, all third year students in Krishna, and all fourth year students in Godavari, then the non-key attribute Hostel name is dependent on the non-key attribute Year. This dependency is shown in figure 6.

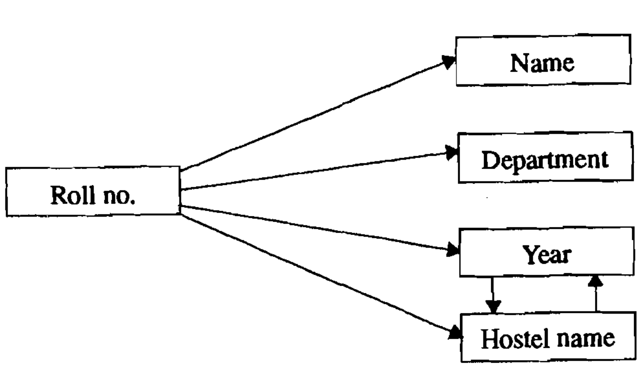

Figure 6: Dependency diagram for the relation

Observe that given the year of student, his hostel is known and vice versa. The dependency of hostel on year leads to duplication of data as is evident from table 4. If it is decided to ask all first year students to move to Kaveri hostel, and all second year students to Ganga hostel, this change should be made in many places in table 4. Also, when a student's year of study changes, his hostel change should also be noted in Table 4. This is undesirable. Table 4 is said to be in 3NF if it is in 2NF and no non-key attribute is functionally dependent on any other non-key attribute. Table 4 is thus not in 3NF. To transform it to 3NF, we should introduce another relation which includes the functionally related non-key attributes. This is shown in table 5.

Roll no.	Name	Department	Year
1784	Raman	Physics	1
1648	Krishnan	Chemistry	1
1768	Gopalan	Mathematics	2
1848	Raja	Botany	2
1682	Maya	Geology	3
1485	Singh	Zoology	4

Year	Hostel name
1	Ganga
2	Kaveri
3	Krishna
4	Godavari

Table 5: Conversion of table 4 into two 3NF relations

It should be stressed again that dependency between attributes is a semantic property and has to be stated in the problem specification. In this example the dependency between Year and Hostel is clearly stated. In case hostel allocated to students do not depend on their year in college, then table 4 is already in 3NF.

Let us consider another example of a relation. The relation **Employee** is given below and its dependency diagram in figure 7.

Employee (**Employee code,** Employee name, Dept., Salary, Project no., Termination date of project).

As can be seen from the figure, the termination date of a project is dependent on the Project no. Thus this relation is not in 3NF. The 3NF relations are:

Employee (**Employee code,** Employee name, Salary, Project no.)

Project (**Project no.** Termination date)

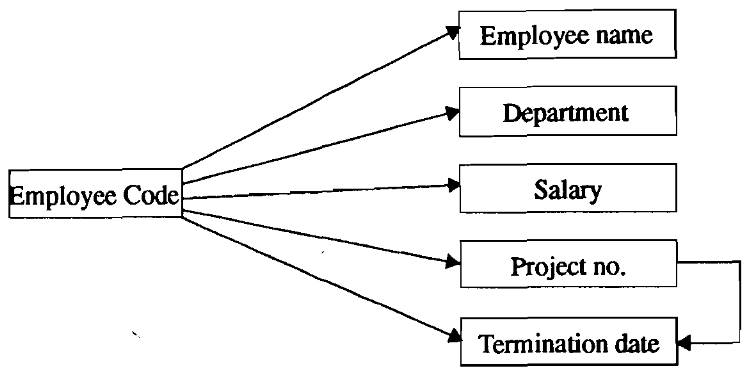

Figure 7: Dependency diagram of employee relation

2.8 BOYCE-CODD NORMAL FORM (BCNF)

Assume that a relation has more than one possible key. Assume further that the composite keys have a common attribute. If an attribute of a composite key is dependent on an attribute of the other composite key, a normalization called BCNF is needed. Consider, as an example, the relation **Professor**:

Professor (**Professor code,** Dept., Head of Dept., Parent time)

It is assumed that

1. A professor can work in more than one department
2. The percentage of the time he spends in each department is given.
3. Each department has only one Head of Department.

The relationship diagram for the above relation is given in figure 8. Table 6 gives the relation attributes. The two possible composite keys are professor code and Dept. or Professor code and Head of Dept. Observe that department as well as Head of Dept. are not non-key attributes. They are a part of a composite key.

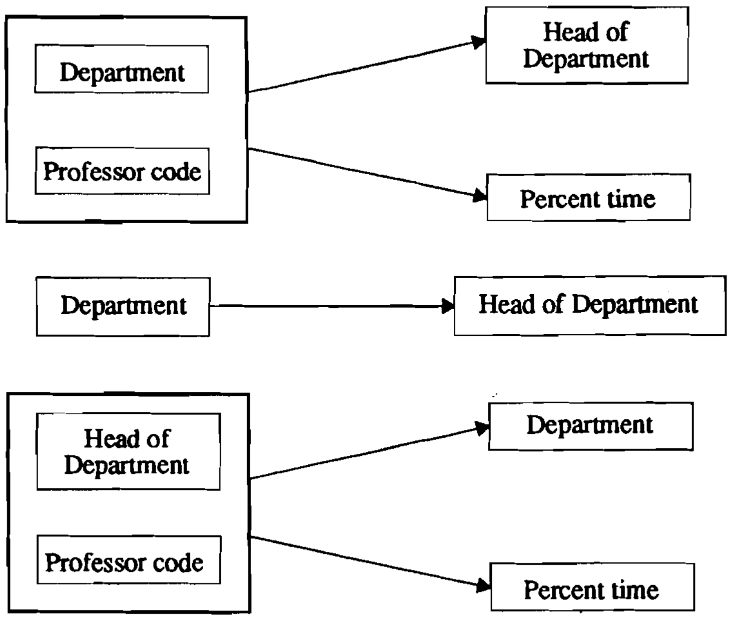

Figure 8: Dependency diagram of Professor relation

Professor Code	Department	Head of Dept.	Parent
P1	Physics	Ghosh	50
P1	Mathematics	Krishnan	50
P2	Chemistry	Rao	25
P2	Physics	Ghosh	75
P3	Mathematics	Krishnan	100

Table 6: Normalization of Relation "Professor"

The relation given in table 6 is in 3NF. Observe, however, that the names of Dept. and Head of Dept. are duplicated. Further, if Professor P2 resigns, rows 3 and 4 are deleted. We lose the information that Rao is the Head of Department of Chemistry.

The normalization of the relation is done by creating a new relation for Dept. and Head of Dept. and deleting Head of Dept. from Professor relation. The normalized relations are shown in the following table 7.

(a)

Professor code	Department	Percent time
P1	Physics	50
P1	Mathematics	50
P2	Chemistry	25
P2	Physics	75
P3	Mathematics	100

(b)

Department	Head of Dept.
Physics	Ghosh
Mathematics	Krishnan
Chemistry	Rao

Table 7: Normalized Professor Relation in BCNF

and the dependency diagrams for these new relations in figure 8. The dependency diagram gives the important clue to this normalization step as is clear from figures 8 and 9.

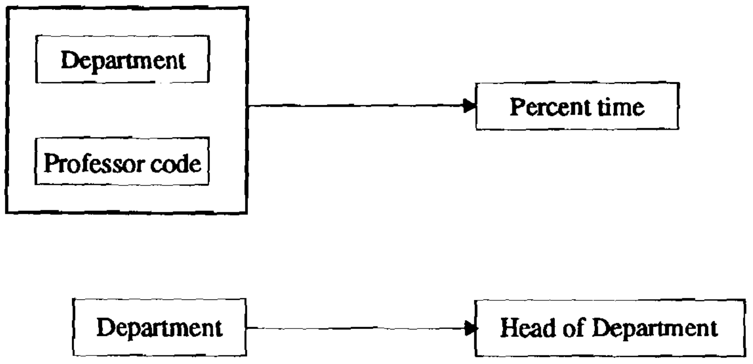

Figure 9: Dependency diagram of Professor relation

2.9 FOURTH AND FIFTH NORMAL FORM

When attributes in a relation have multivalued dependency, further Normalisation to 4NF and 5NF are required. We will illustrate this with an example. Consider a vendor supplying many items to many projects in an organisation. The following are the assumptions:

1. A vendor is capable of supplying many items.
2. A project uses many items.
3. A vendor supplies to many projects.
4. An item may be supplied by many vendors.

Table 8 gives a relation for this problem and figure 10 the dependency diagram(s).

Vendor code	Item code	Project no.l
V1	I1	P1
V1	I2	P1
V1	I1	P3
V1	I2	P3
V2	I2	P1
V2	I3	P1
V3	I1	P2
V3	I1	P2

Table 8: Vendor-supply-projects Relation

The relation given in table 8 has a number of problems. For example:

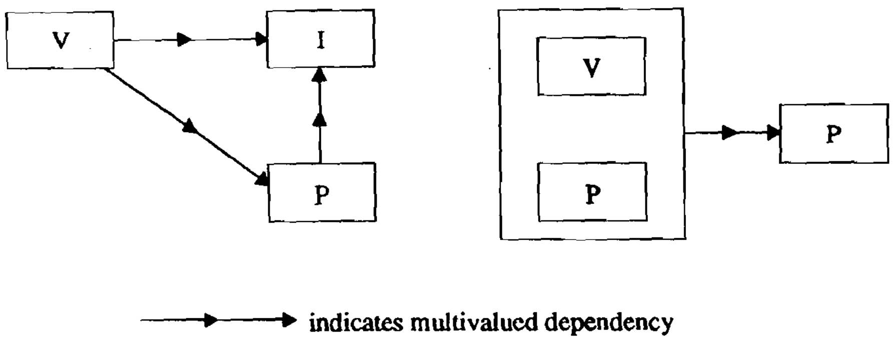

Figure 10 : Dependency diagrams of vendor-supply-project relation

- If vendor V1 has to supply to project P2, but the item is not yet decided, then a row with a blank for item code has to be introduced.
- The information about item 1 is stored twice for vendor V3.

Observe that the relation given in Table 8 is in 3NF and also in BCNF. It still has the problems mentioned above. The problem is reduced by expressing this relation as two relations in the Fourth Normal Form (4NF). A relation is in 4NF if it has no more than one independent multivalued dependency or one independent multivalued dependency with a functional dependency.

Table 8 can be expressed as the two 4NF relations given in Table 9. The fact that vendors are capable of supplying certain items and that they are assigned to supply for some projects in independently specified in the 4NF relation.

(a) Vendor Supply	
Vendor code	Item code
V1	I1
V1	I2
V2	I2
V2	I3
V3	I1

(b) Vendor project	
Vendor code	Project no.
V1	P1
V1	P3
V2	P1
V3	P1
V3	P2

Table 9: Vendor-supply-project Relations in 4NF

These relations still have a problem. Even though vendor V1's capability to supply items and his allotment to supply for specified projects may not need it. We thus need another relation which specifies this. This is called 5NF form. The 5NF relations are the relations in Table 9(a) and 9(b) together with the relation given in table 10.

Project no.	Item code
P1	I1
P1	I2
P2	I1
P3	I1
P3	I3

Table 10: 5NF Additional Relation

In table 11 we summarise the normalisation steps already explained.

Input relation	Transformation	Output relation
All relations	Eliminate variable length records. Remove multiattribute lines in table	1NF
1NF relation	Remove dependency of non-key attribute on part of a multiattribute key	2NF
2NF	Remove dependency of non-key attributes on other non-key attributes	3NF
3NF	Remove dependency of an attribute of a multiattribute key on an attribute of another (overlapping) multi-attribute key	BCNF
BCNF	Remove more than one independent multivalued dependency from relation by splitting relation	4 NF
4NF	Add one relation relating attributes with multivalued dependency to the two relations with multivalued dependency	5 NF

Table 11: Summary of Normalisation Steps

2.10 SOME EXAMPLES OF DATABASE DESIGN

Consider a problem where items are supplied by a vendor and checked by a receiving process against orders for detecting any discrepancy. An order file is used to check whether the deliveries are against orders and whether there is any discrepancy. We will now see how these data can be organised as relations. The E-R diagram of figure 11 applies for this case and the relations are:

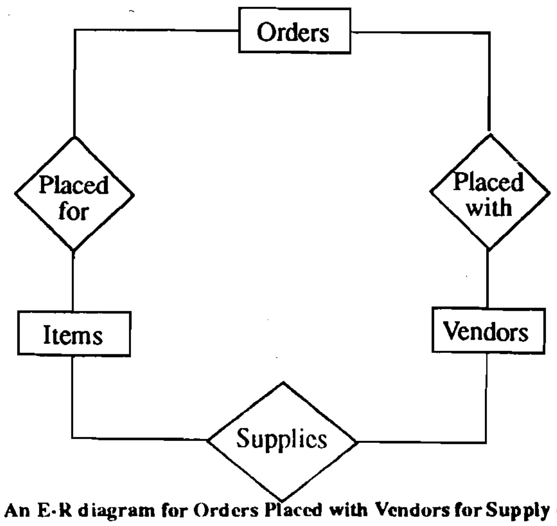

Figure 11: An E-R diagram for Orders Placed with Vendors for Supply of Items

ORDERS (**order no.**, order date)

ORDER PLACED FOR (**order no., item code**)

ORDER PLACED WITH (**order no., vendor code, item code,** qty. ordered, price/unit, delivery time allowed)

VENDORS (**vendor no.**, vendor name, vendor address)

ITEMS (**item code,** item name)

SUPPLIES (**vendor code, item code, order no.,** qty. supplied, date of supply)

The key attribute(s) are in bold letter(s) in each relation.

Let us examine whether the relations are in normal form. ORDERS and ORDER PLACED FOR are simple relations. In the relation ORDER PLACED WITH, the key is the composite attributes order no., vendor code and item code. However, order no. and vendor code are functionally related if we assume that a given order no. has only one vendor specified. Further, an order is with a vendor for an item. The price/unit depends on the item and the vendor. Thus we need to modify the relations ORDER PLACED FOR and ORDER PLACED WITH to :

ORDER PLACED FOR (**order no., Item code,** qty. ordered, price/unit, delivery time)

ORDER PLACED WITH (**vendor code, Item code**)

The two relations have composite keys. The non-key fields are not related to one another. In a key with more than one attribute the individual attributes are not functionally dependent. Thus these two relations are in normalised form and do not need any further change.

There is still one problem. Many orders may be given to the same vendor for the same or different items. In order to organise this data we need one more relation

ORDER WITH VENDOR (**order no.,** vendor code)

so that we can find out which vendor supplied against an order.

The relations VENDOR and ITEM are simple and are in normalised form. The relation SUPPLIES is, however, not normalised. Vendor code and order no. are functionally dependent. There is a multivalued dependency between vendor code and item code as a vendor can supply many items. We thus split the relations into two relations:

ACTUAL VENDOR SUPPLY (**vendor no., item code,** qty. supplied, date of supply)

VENDOR SUPPLY CAPABILITY (**vendor code, item code**)

Observe that the relations VENDOR SUPPLY CAPABILITY and ORDER PLACED WITH have identical attributes. However, VENDOR SUPPLY CAPABILITY relation will have a (**vendor code, item code**) table without a vendor having supplied any item. The relation ORDER PLACED WITH will have a tuple only when a vendor actually supplies an item.

We now consider another problem. Let a database contain the following: Teacher code, Teacher's name, Teacher's address, rank, department, courses taught by the teacher, course name, credits for course, no. of students in the class, course taught in semester no., student no., name, dept., year, and courses taken in semester no. The following information is given on dependencies.

- A teacher may teach more than one course in a semester.
- A teacher is affiliated to only one department.
- A student may take many courses in a semester.
- The same course may have more than one section and different sections may be taught by different teachers.

An entity-relationship diagram for this problem is given in figure 12. The relations corresponding to the E-R diagram are:

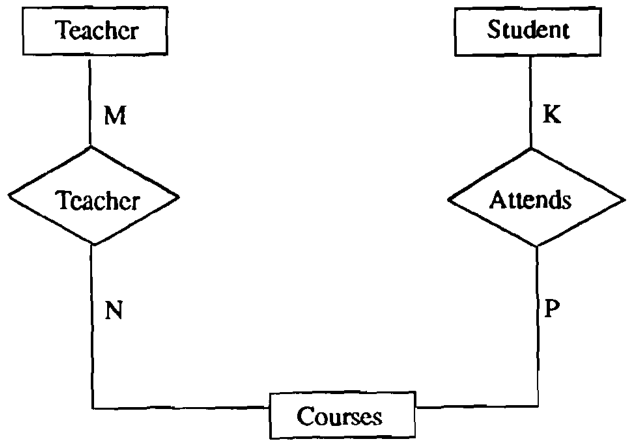

Figure 12 : An E-R diagram for teacher database

TEACHER (**Teacher code, course no.,** no. of rank, dept.)

TEACHES COURSES (**Teacher code, course no.,** no. of students)

COURSES (**course no.,** course name, credits, semester taught)

STUDENT (**student no.,** student's name, dept., year)

STUDENT-COURSES (**student's no., course no.,** semester no.)

TEACHER relation has only one key. All non-key attributes are functionally dependent only on the key. There is no functional dependency among non-key attributes. Thus the relation is normalised in 3NF. (No higher NFs are applicable).

STUDENT relation is also, similarly, in 3NF. In the COURSE relation, course name could also be a key. However, there is no overlapping multiattribute key. The relation is in 3NF and no further normalisation is required. The relations TEACHES COURSES and STUDENT- COURSES have multiattribute keys, but the relations themselves are in normal form. The only point which is not clear, from these relations, is the relation between teacher and student. This has been missed in the E-R diagram. The relationship is between the teacher, courses taught and students. In other words, we should be able to answer the question "Which teacher is teaching course no. X to student no. Y?" Let us add a relation

TEACHES TO (**Teacher code, student no., course no.**)

In this relation **Teacher code** and **course no.**, have a multivalued dependency. Similarly, **Teacher code** and **student no.** as well as **student no.** and **course no.** have multivalued dependency. However, TEACHES COURSES (**Teacher code, course no.**, no. of students) and STUDENT-COURSES (**student no., course no.**, semester no.) relations are already in the database. Thus the relation TEACHES TO as it is specified above is sufficient to give the idea that student Y takes course X from Teacher Z.

Check Your Progress

1. What is the basic purpose of 4NF?

....

....

2. What types of anomalies are found in relational database?

....

....

....

2.11 SUMMARY

In this unit, we pointed out different types of anomalies in the database that could cause an undesirable effect. We also discussed several forms of normalization that could help in removing these anomalies.

2.12 MODEL ANSWERS

1. The 2nd, 3rd and BCNF normal forms deal with functional dependencies only. It is possible for a relation in 3NF to still exhibit update, insertion and deletion anomalies. This can happen when multivalued dependencies are not properly taken care of. In order to eliminate anomalies arise out of these dependencies, the notion of 4NF was developed.

2. There are 3 types of anomalies in database. These are:

   (i) Insertion anomalies

   (ii) Deletion anomalies

   (iii) Update anomalies

2.13 FURTHER READINGS

1. Bipin.C. Desai, *An Introduction to Database System*, Galgotia Publication, New Delhi.

2. V. Rajaraman, *Analysis and Design of Informatic System*, PHI, New Delhi-1995.