This chapter focuses on various stages of accomplishing normalization. Normal forms are discussed in detail for relational databases and database design step to normalise the relational table. This helps in achieving the minimum redundancy without compromising on easy data and information retrieval properties of the database.

10.2 NORMALIZATION

Normalization is a process of decomposing a set of relations with anomalies to produce smaller and well-structured relations that contain minimum or no redundancy. It is a formal process of deciding which attributes should be grouped together in a relation. Normalization provides the designer with a systematic and scientific process of grouping of attributes in a relation. Using normalization, any change to the values stored in the database can be achieved with the fewest possible update operations.

Therefore, the process of normalization can be defined as a procedure of successive reduction of a given collection of relational schemas based on their FDs and primary keys to achieve some desirable form of minimised redundancy, minimised insertion, minimised deletion and minimised update anomalies.

A normalised schema has a minimal redundancy, which requires that the value of no attribute of a database instance is replicated except where tuples are linked by foreign keys (a set of attributes in one relation that is a key in another). Normalization serves primarily as a tool for validating and improving the logical database design, so that the logical design satisfies certain constraints and avoids unnecessary duplication of data. The process of normalization provides the following to the database designers:

- A formal framework for analysing relation schemas based on their keys and on the functional dependencies among their attributes.
- A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be normalised to any desired degree.

However, during normalization, it is ensured that a normalised schema

- does not lose any information present in the un-normalised schema,
- does not include spurious information when the original schema is reconstructed
- preserves dependencies present in the original schema.

The process of normalization was first proposed by E.F. Codd. Normalization is a bottom-up design technique for relational database. Therefore, it is difficult to use in large database designs. However, this technique is still useful in some circumstances mentions below:

- As a different method of checking the properties of design arrived at through EER modelling.
- As a technique for reverse engineering a design from an existing undocumented implementation.

10.3 NORMAL FORMS

A *normal form* is a state of a relation that results from applying simple rules regarding functional dependencies (FDs) to that relation. It refers to the highest normal form of condition that it meets. Hence, it indicates the degree to which it has been normalised. The normal forms are used to ensure that various types of anomalies and inconsistencies are not introduced into the database. For determining whether a particular relation is in normal form or not, the FDs between the attributes in the relation are examined and not the current contents of the relation. First C. Berri and his co-workers proposed a notation to emphasise these relational characteristics. They proposed that the relation is defined as containing two components namely (a) the attributes (b) the FDs between them. It takes the form

$$R_1 = (\{X, Y, Z\}, \{X \rightarrow Y, X \rightarrow Z\})$$

The first component of the relation R_1 is the attributes, and the second component is the FDs. For example, let us look at the relation ASSIGN of Table 10.2.

The first component of the relation ASSIGN is

{EMP-NO, PROJECT, PROJECT-BUDGET,
YRS-SPENT-BY-EMP-ON-PROJECT}

The second component of the relation ASSIGN is

{EMP-NO, PROJECT → YRS-SPENT-BY-EMP-ON-PROJECT,
PROJECT → PROJECT-BUDGET}

Table 10.2 Relation ASSIGN

Relation R_1: ASSIGN			
EMP-NO	PROJECT	PROJECT-BUDGET	YRS-SPENT-BY EMP-ON-PROJECT
106519	P1	INR 100 CR	5
112233	P3	INR 150 CR	2
106519	P2	INR 200 CR	5
123243	P4	INR 100 CR	10
106519	P3	INR 150 CR	3
111222	P1	INR 300 CR	4

The FDs between attributes are important when determining the relation's key. A relation key uniquely identifies a tuple (row). Hence, the key or prime attributes uniquely determines the values of the non-key or non-prime attributes. Therefore, a full FD exists from the prime to the nonprime attributes. It is with full FDs whose determinants are not keys of a relation that problems arise. For example, in the relation ASSIGN of Table 10.2, the key is {EMP-NO, PROJECT}. However, PROJECT-BUDGET depends on only part of the key. Alternatively, the determinant of the FD: PROJECT → PROJECT-BUDGET is not the key of the relation. This undesirable property causes the anomalies. Conversion to normal forms requires a choice of relations that do not contain such undesirable dependencies. Various types of normal forms used in relational database are as follows:

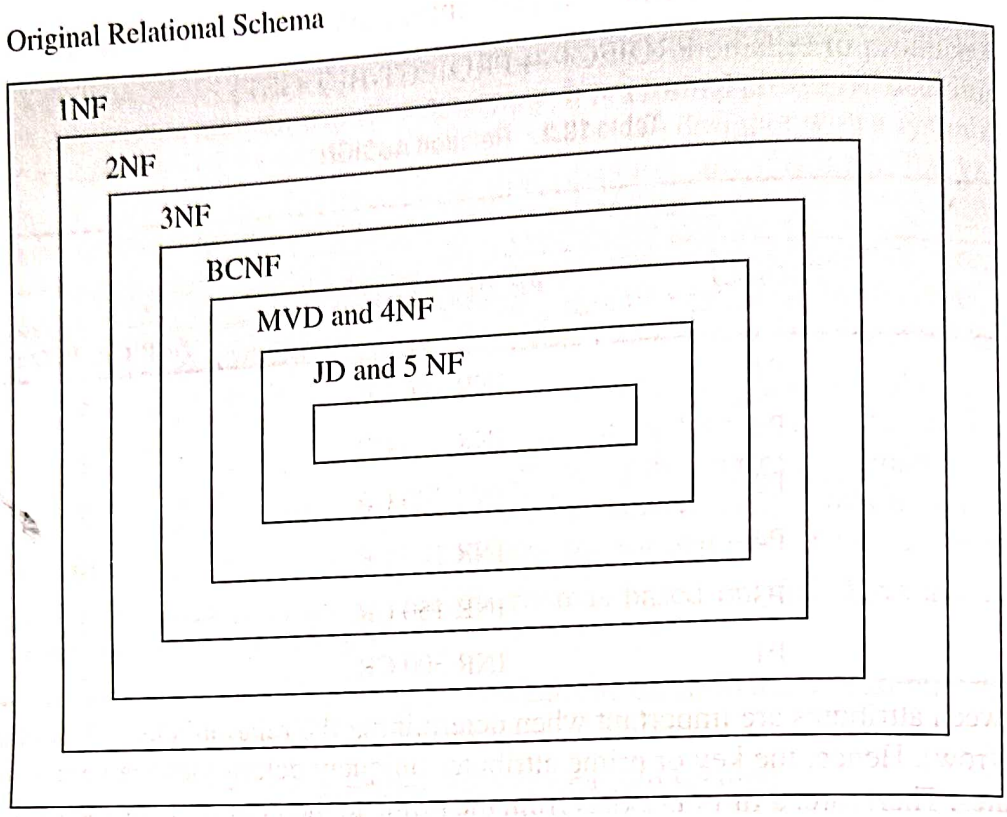
- First normal form (1NF).
- Second normal form (2NF).
- Third normal form (3NF).
- Boyce/Codd normal form (BCNF).
- Fourth normal form (4NF).
- Fifth normal form (5NF).

A relational schema is said to be in a particular normal form if it satisfies a certain prescribed set of conditions, as discussed in subsequent sections. Fig. 10.1 illustrates the levels of normal forms. As shown in the figure, every relation is in 1NF. Also, every relation in 2NF is also in 1NF, every relation in 3NF is also in 2NF and so on.

Initially, E.F. Codd proposed three normal forms namely INF, 2NF and 3NF. Subsequently, BCNF was introduced jointly by R. Boyce and E.F. Codd. Later, the normal forms 4NF and 5NF were introduced, based on the concepts of multi-valued dependencies and join dependencies, respectively.

All of these normal forms are based on functional dependencies among the attributes of a relational table.

Fig. 10.1 Levels of normalization



10.3.1 First Normal Form (1NF)

A relation is said to be in *first normal form* (*1NF*) if the values in the domain of each attribute of the relation are atomic (that is simple and indivisible). In 1NF, all domains are simple and in a simple domain, all elements are atomic. Every tuple (row) in the relational schema contains only one value of each attribute and no repeating groups. 1NF data requires that every data entry, or attribute (field) value, must be non-decomposable. Hence, 1NF disallows having a set of values, a tuple of values or a combination of both as an attribute value for a single tuple. 1NF disallows multi-valued attributes that are themselves composites. This is called "relations within relations", or *nested relations*, or "relations as attributes of tuples".

Example 1

Consider a relation LIVED_IN, as shown in Fig. 10.2 (a), which keeps records of person and his residence in different cities. In this relation, the domain RESIDENCE is not simple. For example, an attribute "Abhishek" can have residence in Jamshedpur, Mumbai or Delhi. Therefore, the relation is un-normalised. Now, the relation LIVED_IN is normalised by combining each row in residence with its corresponding value of PERSON and making this combination a tuple (row) of the relation, as shown in Fig. 10.2 (b). Thus, now non-simple domain RESIDENCE is replaced with simple domains.

Example 2

Let us consider another relation PATIENT_DOCTOR, as shown in Table 10.3 which keeps the records of appointment details between patients and doctors. This relation is in 1NF. The relational table can be depicted as:

PATIENT_DOCTOR (PATIENT-NAME, DATE-OF-BIRTH, DOCTOR-NAME, CONTACT-NO, DATE-TIME, DURATION-MINUTES)

Fig. 10.2 Relation LIVED-IN

Relation: LIVED_IN

PERSON	RESIDENCE	
	CITY	DATE-MOVED
Abhishek	Jamshedpur	121202
	Mumbai	070803
	Delhi	050504
Thomas	CITY	DATE-MOVED
	Singapore	100401
	Kolkata	090202
	Bangalore	010105

(*a*) Un-normalised relation

Relation: LIVED_IN

PERSON	CITY	DATE-MOVED-IN
Abhishek	Jamshedpur	121202
Abhishek	Mumbai	070803
Abhishek	Delhi	050504
Thomas	Singapore	100401
Thomas	Kolkata	090202
Thomas	Bangalore	010105

(*b*) Normalised relation

Table 10.3 Relation PATIENT_DOCTOR in 1NF

Relation: PATIENT_DOCTOR

PATIENT-NAME	DATE-OF-BIRTH	DOCTOR-NAME	CONTACT-NO	DATE-TIME	DURATION-MINUTES
Mathew	10.02.1957	Abhishek	657-2145063	10.01.05 10:00	15
Ravi	27.01.1962	Sanjay	651-2214381	10.01.05 11:00	10
Jose	30.03.1971	Thomas	011-2324567	10.01.05 10:30	10
Jose	30.03.1971	Thomas	011-2324567	08.03.05 09:00	20
Ravi	27.01.1962	Abhishek	657-2145063	10.01.05 10:15	15
Mathew	10.02.1957	Thomas	011-2324567	10.01.05 10:50	20
Ranjan	02.11.1970	Sanjay	651-2214381	10.01.05 11:10	20
Mathew	10.02.1957	Abhishek	657-2145063	05.05.05 16:00	15

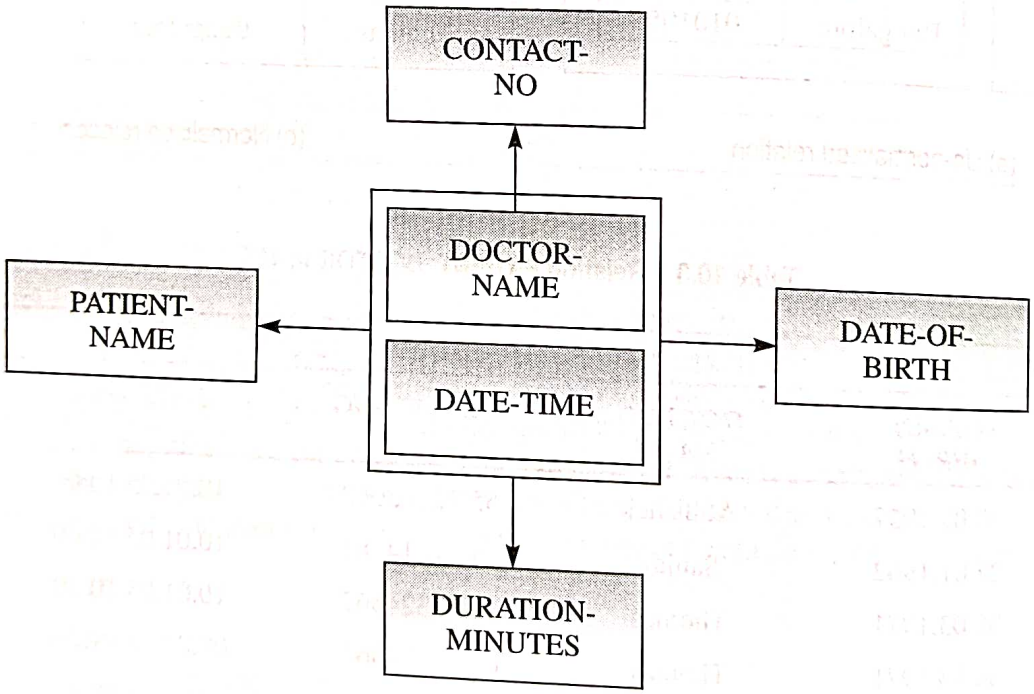
It can be observed from the relational table that a doctor cannot have two simultaneous appointments and thus DOCTOR-NAME and DATE-TIME is a compound key. Similarly, a patient cannot have same time from two different doctors. Therefore, PATIENT-NAME and DATE-TIME attributes are also a candidate key.

Problems with 1NF

- 1NF contains redundant information. For example, the relation PATIENT_DOCTOR in 1NF of Table 10.3 has the following problems with the structure:
  (a) A doctor, who does not currently have an appointment with a patient, cannot be represented.
  (b) Similarly, we cannot represent a patient who does not currently have an appointment with a doctor.
  (c) There is redundant information such as the patient's date-of-birth and the doctor's phone numbers stored in the table. This will require considerable care while inserting new records, updating existing records or deleting records to ensure that all instances retain the correct values.
  (d) While deleting the last remaining record containing details of a patient or a doctor, all records of that patient or doctor will be lost.

Therefore, the relation PATIENT_DOCTOR has to be normalised further by separating the information relating to several distinct entities. Fig. 10.3 shows the functional dependencies diagrams in the PATIENT_DOCTOR relation. Now, it is clear from the functional dependency diagram that although the patient's name, date of birth and the duration of the appointment are dependent on the key (DOCTOR-NAME, DATE-TIME), the doctor's contact number depends on only part of the key (DOCTOR-NAME).

Fig. 10.3 *Functional dependency diagram for relation PATIENT-DOCTOR*



10.3.2 Second Normal Form (2NF)

A relation R is said to be in *second normal form* (*2NF*) if it is in 1NF and every non-prime key attributes of R is fully functionally dependent on each relation (primary) key of R. In other words, no attributes of the relation (or table) should be functionally dependent on only one part of a concatenated primary key. Thus, 2NF can be violated only when a key is a composite key or one that consists of more than one attribute. 2NF is based on the concept of full functional dependency (FFD), as explained in Section 9.2.2, Chapter 9. 2NF is an intermediate step towards higher normal forms. It eliminates the problems of 1NF.

Example 1

As shown in Fig. 10.3, the partial dependency of the doctor's contact number on the key DOCTOR-NAME indicates that the relation is not in 2NF. Therefore, to bring the relation in 2NF, the information about doctors and their contact numbers have to be separated from information about patients and their appointments with doctors. Thus, the relation is decomposed into two tables, namely PATIENT_DOCTOR and DOCTOR, as shown in Table 10.4. The relational table can be depicted as:

PATIENT_DOCTOR (PATIENT-NAME, DATE-OF-BIRTH, DOCTOR-NAME, DATE-TIME, DURATION-MINUTES)

DOCTOR (DOCTOR-NAME, CONTACT-NO)

Table 10.4 Relation PATIENT_DOCTOR decomposed into two tables for refirement into 2NF

Relation: PATIENT_DOCTOR

PATIENT-NAME	DATE-OF-BIRTH	DOCTOR-NAME	DATE-TIME	DURATION-MINUTES
Mathew	10.02.1957	Abhishek	10.01.05 10:00	15
Ravi	27.01.1962	Sanjay	10.01.05 11:00	10
Jose	30.03.1971	Thomas	10.01.05 10:30	10
Jose	30.03.1971	Thomas	08.03.05 09:00	20
Ravi	27.01.1962	Abhishek	10.01.05 10:15	15
Mathew	10.02.1957	Thomas	10.01.05 10:50	20
Ranjan	02.11.1970	Sanjay	10.01.05 11:10	20
Mathew	10.02.1957	Abhishek	05.05.05 16:00	15

(*a*) Relation PATIENT_DOCTOR

Relation: DOCTOR

DOCTOR-NAME	CONTACT-NO
Abhishek	657-2145063
Sanjay	651-2214381
Thomas	011-2324567
Thomas	011-2324567
Abhishek	657-2145063
Thomas	011-2324567
Sanjay	651-2214381
Abhishek	657-2145063

(*b*) Relation DOCTOR

Fig. 10.4 shows the functional dependencies diagrams (FDD) of relations PATIENT_DOCTOR and DOCTOR.

Example 2

Let us consider another relation ASSIGN as shown in Table 10.2. This relation is not in 2NF because the non-prime attribute PROJECT-BUDGET is not fully dependent on the relation (or primary) key

EMP-NO and PROJECT. Here PROJECT-BUDEGT is, in fact, fully functionally dependent on PROJECT, which is a subset of the relation key. Thus, the relation ASSIGN is decomposed into two relations, namely ASSIGN and PROJECTS, as shown in Table 10.5. Now, both the relations ASSIGN and PROJECTS are in 2NF.

Fig. 10.4 FDDs for relations PATIENT-DOCTOR and DOCTOR

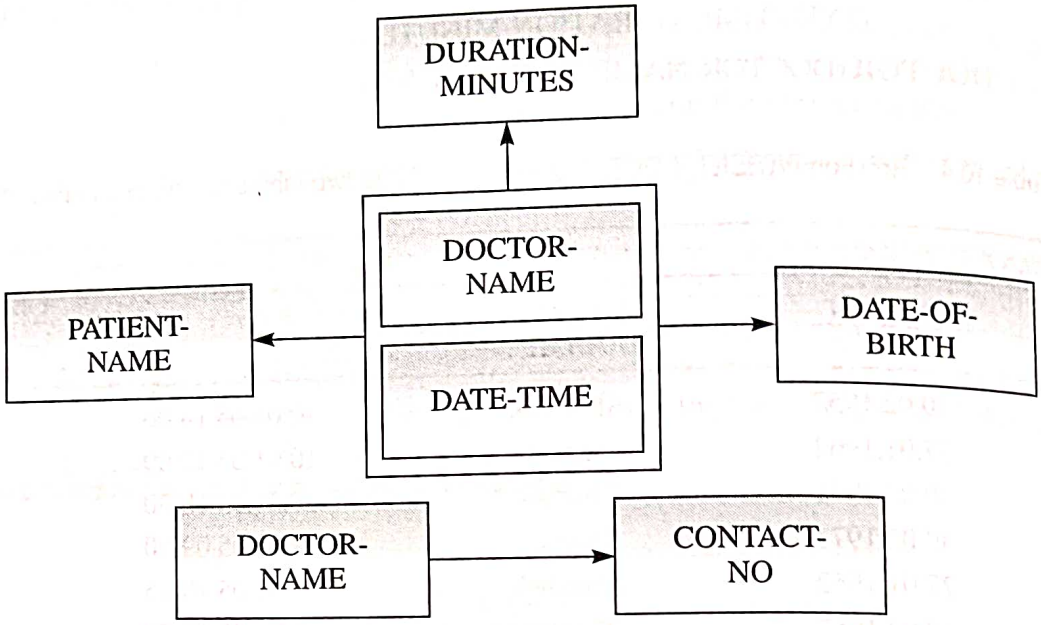


Table 10.5 Decomposition of relations ASSIGN into ASSIGN and PROJECTS as 2NF

Relation: ASSIGN

EMP-NO	*PROJECT*	*YRS-SPENT-BY EMP-ON-PROJECT*
106519	P1	5
112233	P3	2
106519	P2	5
123243	P4	10
106519	P3	3
111222	P1	4

(*a*)

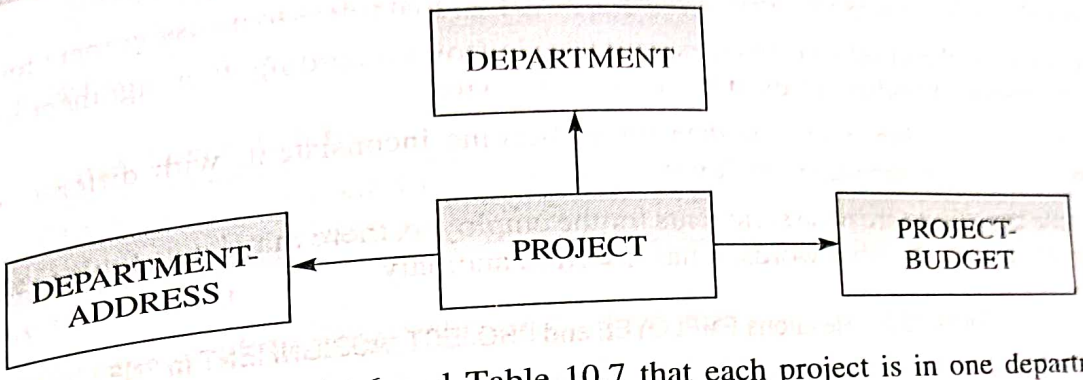
Relation: PROJECT

PROJECT	*PROJECT-BUDGET*
P1	INR 100 CR
P2	INR 150 CR
P3	INR 200 CR
P4	INR 100 CR
P5	INR 150 CR
P6	INR 300 CR

(*b*)

Let us create a new relation PROJECT_DEPARTMENT by adding information DEPARTMENT and DEPARTMENT-ADDRESS in the relation PROJECT of Table 10.5. The new relation PROJECT_DEPARTMENT is shown in Table 10.6. The functional dependencies between the attributes of the relation are shown in Fig. 10.5.

Fig. 10.5 Functional dependency diagram for relation PROJECT_DEPARTMENT



As can been seen from Table 10.6 and Table 10.7 that each project is in one department, and each department has one address. It is however, possible for a department to include more than one project. The relation has only one relation (primary) key, namely, PROJECT. Both DEPARTMENT and DEPARTMENT-ADDRESS are fully functionally dependent on PROJECT. Thus, relation PROJECT_DEPARTMENT is in 2NF.

Table 10.6 Relation PROJECT_DEPARTMENT

Relation: PROJECT_DEPARTMENT			
PROJECT	PROJECT-BUDGET	DEPARTMENT	DEPARTMENT-ADDRESS
P1	INR 100 CR	Manufacturing	Jamshedpur - 1
P2	INR 150 CR	Manufacturing	Jamshedpur - 1
P3	INR 200 CR	Manufacturing	Jamshedpur - 1
P4	INR 100 CR	Training	Mumbai - 2

Table 10.7 Relation EMPLOYEE_PROJECT_ASSIGNMENT

Relation: EMPLOYEE_PROJECT_ASSIGNMENT			
EMP-ID	PROJECT-ID	EMP-NAME	PROJ-START-DATE
106519	P1	Kumar Abhishek	20.05.04
112233	P1	Thomas Mathew	11.11.04
106519	P2	Kumar Abhishek	03.03.05
112233	P3	Thomas Mathew	12.01.05
112233	P-4	Thomas Mathew	30.03.05

Example 3

Let us consider another relation EMPLOYEE_PROJECT_ASSIGNMENT as shown in Table 10.7. This relation has keys EMP-ID and PROJECT-ID together. Employee's name (EMP-NAME) is determined by employee's identification number (EMP-ID) and so is functionally dependent on a part of the key. That means, an attribute EMP-ID of the employee is sufficient to identify the employee's name. Thus, the relation is not in 2NF.

This relation EMPLOYEE_PROJECT_ASSIGNMENT has the following problems:

- The employee's name is repeated in every tuple (row) that refers to an assignment for that employee.
- If the name of the employee changes, every tuple (row) recording an assignment of that employee must be update. In other words, it has update anomaly.
- Because of the redundancy, the data might become inconsistent, with different tuples showing different names for the same employee.
- If at some time there are no assignments for the employee, there may be no tuple in which to keep the employee's name. In other words, it has insertion anomaly.

Table 10.8 Relations EMPLOYEE and PROJECT_ASSIGNMENT in 2NF

Relation: EMPLOYEE

EMP-ID	*EMP-NAME*
106519	Kumar Abhishek
112233	Thomas Mathew

(*a*)

Relation: PROJECT-ASSIGNMENT

EMP-NO	*PROJECT*	*YRS-SPENT-BY EMP-ON-PROJECT*
106519	P1	20.05.04
112233	P1	11.1104
106519	P2	03.03.05
123243	P3	12.01.05
112233	P4	30.03.05

(*b*)

The relation EMPLOYEE_PROJECT_ASSIGNMENT can now be decomposed into the following two relations, as shown in Table 10.8.

EMPLOYEE (EMP-ID, EMP-NAME)

PROJECT_ASSIGNMENT (EMP-ID, PROJECT-ID, PROJ-START-DATE)

To bring the relation EMPLOYEE_PROJECT_ASSIGNMENT into 2NF, it is decomposed into two relations EMPLOYEE and PROJECT_ASSIGNMENT, as shown in Table 10.8. These decomposed relations EMPLOYEE and PROJECT_ASSIGNMENT are now in 2NF and the problems as discussed previously, are eliminated. These decomposed relations are called the projection of the original relation EMPLOYEE_PROJECT_ASSIGNMENT. It can be noticed in Table 10.8 that the relation PROJECT_ASSIGNMENT still has five tuples. This is so because the values for EMP-ID, PROJECT-ID and PROJ-START-DATE, taken together, were unique. However, in the relation EMPLOYEE, there are only two tuples, because there were only two unique sets of values for EMP-ID and EMP-NAME. Thus, data redundancy

and the possibility of anomalies have been eliminated.

Problems with 2NF

- As shown in Table 10.4, deleting a record from relation PATIENT_DOCTOR may lose patient's details.
- Any changes in the details of the patient of Table 10.4, may involve changing multiple occurrences because this information is still stored redundantly.
- As shown in Table 10.6 and Fig. 10.5, a department's address may be stored more than once, because there is a functional dependency between non-prime attributes, as DEPARTMENT-ADDRESS is functionally dependent on DEPARTMENT.

10.3.3 Third Normal Form (3NF)

A relation *R* is said to be in *third normal form* (*3NF*) if the relation *R* is in 2NF and the non-prime attributes (that is, attributes that are not part of the primary key) are

- mutually independent,
- functionally dependent on the primary (or relation) key.

In other words, no attributes of the relation should be transitively functionally dependent on the primary key. Thus, in 3NF, no non-prime attribute is functionally dependent on another non-prime attribute. This means that a relation in 3NF consists of the primary key and a set of independent nonprime attributes. 3NF is based on the concept of transitive dependency, as explained in Section 9.2.3, Chapter 9.The 3NF eliminates the problems of 2NF.

Example 1

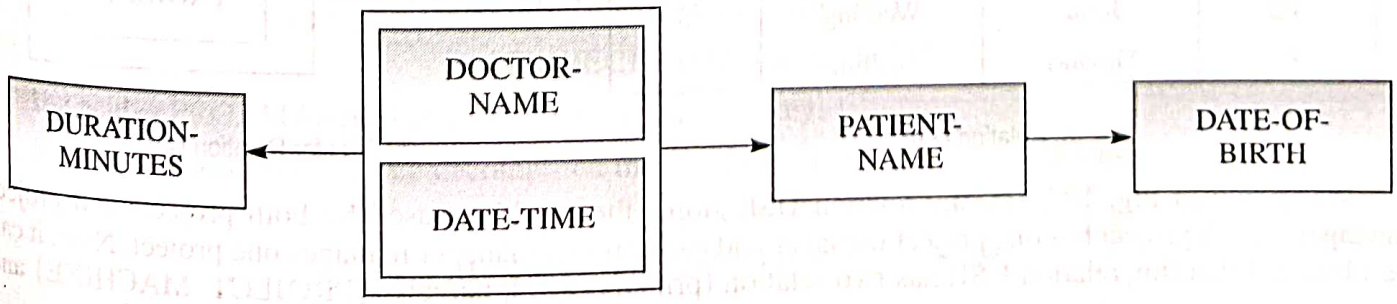
Let us again take example of relation PATIENT_DOCTOR, as shown in Table 10.4 (a). In this relation, there is no dependency between PATIENT-NAME and DURATION-MINUTES. However, PATIENT-NAME and DATE-OF-BIRTH are not mutually independent. Therefore, the relation is not in 3NF. To convert this PATIENT_DOCTOR relation in 3NF, it has to be decomposed to remove the parts that are not directly dependent on relation (or primary) key. Though each value of the primary key has a single associated value of the DATE-OF-BIRTH, there is further dependency called *transitive dependency* linking DATE-OF-BIRTH directly to the primary key, through its dependency on the PATIENT-NAME. A functional dependency diagram is shown in Fig. 10.6. Thus, following three relations are created:

PATIENT (PATIENT-NAME, DATE-OF-BIRTH)

PATIENT_DOCTOR (PATIENT-NAME, DOCTOR-NAME, DATE-TIME, DURATION-MINUTES)

DOCTOR (DOCTOR-NAME, CONTACT-NO)

Fig. 10.6 Functional dependency diagram for relation PATIENT_DOCTOR



Example 2

Similarly, the information in the relation PROJECT_DEPARTMENT of Table 10.5 can be represented in 3NF by decomposing it into two relations, namely, PROJECTS and DEPARTMENT. As shown in Table 10.9, both these relations PROJECTS and DEPARTMENT are in 3NF and department addresses are stored once only.

Table 10.9 Decomposition of relation PROJECT_DEPARTMENT into PROJECTS and DEPARTMENT as 3NF

Relation: PROJECT

PROJECT	PROJECT-BUDGET	DEPARTMENT
P1	INR 100 CR	Manufacturing
P2	INR 150 CR	Manufacturing
P3	INR 200 CR	Manufacturing
P4	INR 100 CR	Training

(*a*)

Relation: DEPARTMENT

DEPARTMENT	DEPARTMENT-ADDRESS
Manufacturing	Jamshedpur-1
Manufacturing	Jamshedpur-1
Manufacturing	Jamshedpur-1
Training	Mumbai-2

(*b*)

Example 3

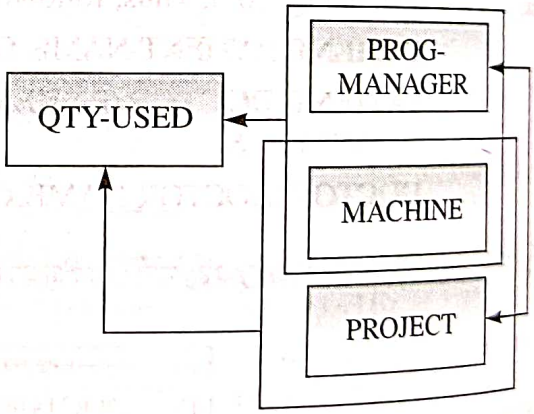
In the previous examples of 3NF, only one relation (primary) key has been used. Conversion into 3NF becomes problematic when the relation has more than one relation keys. Let us consider another relation USE, as shown in Fig. 10.7 (a). Functional dependency diagram (FDD) of relation USE is shown in Fig. 10.7 (b).

Fig. 10.7 Relation USE in 3NF

Relation: USE

PROJECT	PROJ-MANAGER	MACHINE	QTY-USED
P1	Thomas	Excavator	5
P3	John	Shovel	2
P2	Abhishek	Drilling	5
P4	Avinash	Dumper	10
P3	John	Welding	3
P1	Thomas	Drilling	4

(*a*) Relation USE



(*b*) FFD for Relation USE

As shown in Fig. 10.7 (a), the relation USE stores the machines used by both projects and project managers. Each project has one project manager and each project manager manages one project. Now, it can be observed that this relation USE has two relation (primary) keys, namely, {PROJECT, MACHINE} and

{PROJ-MANAGER, MACHINE}. The keys overlap because MACHINE appears in both keys, whereas, PROJECT and PROJ-MANAGER each appear in one relation key only.

The relation USE of Fig. 10.7 has only one non-prime attribute called, QTY-USED, which is fully functionally dependent on each of the two relations. Thus, relation USE is in 2NF. Furthermore, as there is only one non-prime attribute QTY-USED, there can be no dependencies between non-prime attributes. Thus, the relation USE is also in 3NF.

Problems with 3NF

- Since relation USE of Fig. 10.7 has two relation keys that overlap because MACHINE is common to both, the relation has following undesirable properties:
  - The project manager of each project is stored more than once.
  - A project's manager can not be stored until the project has ordered some machines.
  - A project can not be entered unless that project's manager is known.
  - If a project's manager changes, some *n* tuples (rows) also must be changed.
- There is dependency between PROJECT and MANAGER, both of which appear in one relation key only. This dependency leads to redundancy.

10.4 BOYCE-CODD NORMAL FORM (BCNF)

To eliminate the problems and redundancy of 3NF, R.F. Boyce proposed a normal form known as *Boyce-Codd normal form* (*BCNF*). Relation R is said to be in BCNF if for every nontrivial FD: $X \rightarrow Y$ between attributes X and Y holds in R. That means:

- X is super key of R,
- $X \rightarrow Y$ is a trivial FD, that is, $Y \subset X$.

In other words, a relation must only have candidate keys as determinants. Thus, to find whether a relation is in BCNF or not, FDs within each relation is examined. If all non-key attributes depend upon only the complete key, the relation is in BCNF.

Any relation in BCNF is also in 3NF and consequently in 2NF. However, a relation in 3NF is not necessarily in BCNF. The BCNF is a simpler form of 3NF and eliminates the problems of 3NF. The difference between 3NF and BCNF is that for a functional dependency $A \rightarrow B$, 3NF allows this dependency in a relation if B is a primary key attribute and A is not a candidate key. Whereas, BCNF insists that for this dependency to remain in a relation, A must be a candidate key. Therefore, BCNF is a stronger form of 3NF, such that every relation in BCNF is also in 3NF.

Example 1

Relation USE in Fig. 10.7 (a) does not satisfy the above condition, as it contains the following two functional dependencies:

PROJ-MANAGER → PROJECT

PROJECT → PROJ-MANAGER

But neither PROJ-MANAGER nor PROJECT is a super key.

Now, the relation USE can be decomposed into the following two BCNF relations:

USE (PROJECT, MACHINE, QTY-USED)

PROJECTS (PROJECT, PROJ-MANAGER)

Both of the above relations are in BCNF. The only FD between the USE attributes is

PROJECT, MACHINE → QTY-USED

and (PROJECT, MACHINE) is a super key.

The two FDs between the PROJECTS attributes are

PROJECT → PROJ-MANAGER

PROJ-MANAGER → PROJECT

Both PROJECT and PROJ-MANAGER are super keys of relation PROJECTS and PROJECTS is in BCNF.

Example 2

Let us consider another relation PROJECT_PART, as shown in Table 10.10. The relation is given as:

PROJECT_PART (PROJECT-NAME, PART-CODE, VENDOR-NAME, QTY)

This table lists the projects, the parts, the quantities of those parts they use and the vendors who supply these parts. There are two assumptions. Firstly, each project is supplied with a specific part by only one vendor, although a vendor can supply that part to more than one project. Secondly, a vendor makes only one part but the same part can be made by other vendors. The primary keys of the relation PROJECT_PART are PROJECT-NAME and PART-CODE. However, another, overlapping, candidate key is present in the concatenation of the VENDOR-NAME (assumed unique for all vendors) and PROJECT-NAME (assumed unique for all projects) attributes. These would also uniquely identify each tuple of the relation (table).

Table 10.10 Relation PROJECT_PART

Relation: PROJECT_PART			
PROJECT-NAME	***PART-CODE***	***QTY***	***VENDOR-NAME***
P1	abc	10	Thomas
P1	bca	20	John
P2	abc	30	Thomas
P2	bca	40	Abhishek

The relation PROJECT_PART is in 3NF, since there are no transitive FDs on the prime key. However, it is not in BCNF because the attribute VENDOR-NAME is the determinant of PART-CODE (vendor make only one part). As a result of this, the relation can give rise to anomalies. For example, if the bottom tuple (row) is updated because "John" replaces "Abhishek" as the supplier of part "bca" to project "P2", then the information that "Abhishek" makes part "bca" is lost from the database. If a new vendor becomes a part supplier, this fact cannot be recorded in the database until the vendor is contracted to a project. There is also an element of redundancy present in that "Thomas", for example, is shown twice as making part "abc". Decomposing this single relation PROJECT_PART into two relations PROJECT_VENDOR and VENDOR_PART solves the problem. The decomposed relations are given as:

PROJECT_VENDOR (PROJECT-NAME, VENDOR-NAME, QTY)

VENDOR_PART (VENDOR-NAME, PART-CODE)

10.4.1 Problems with BCNF

- Even if a relation is in 3NF or BCNF, undesirable internal dependencies are exhibited with dependencies between elements of compound keys composed of three or more attributes.
- The potential to violate BCNF may occur in a relation that:
  (i) contains two or more composite candidate keys.
  (ii) the candidate keys overlap, that have atleast one attribute in common.
- Let us consider a relation PERSON_SKILL, as shown in Table 10.11. This relation contains the following:
  (a) The SKILL-TYPE possessed by each person. For example, "Abhishek" has "DBA" and "Quality Auditor" skills.
  (b) The PROJECTs to which a person is assigned. For example, "John" is assigned to projects "P1" and "P2".
  (c) The MACHINEs used on each project. For example, "Excavator", "Shovel" and "Drilling" are used on project "P1".

Table 10.11 Relation PERSON_SKILL

Relation: PERSON_SKILL			
PERSON	***PROJECT***	***MACHINE***	***SKILL-TYPE***
Thomas	P1	Excavator	Analyst
John	P1	Shovel	Programmer
Abhishek	P2	Drilling	DBA
Abhishek	P2	Dumper	Quality auditor
John	P2	Welding	Programmer
Thomas	P1	Drilling	DBA

There are no FDs between attributes of relation PERSON_SKILL and yet there is a clear relationship between them. Thus, relation PERSON_SKILL contains many undesirable characteristics, such as:

(i) The fact that "Abhishek has both "DBA" and "Quality Auditor" skills, is stored a number of times.

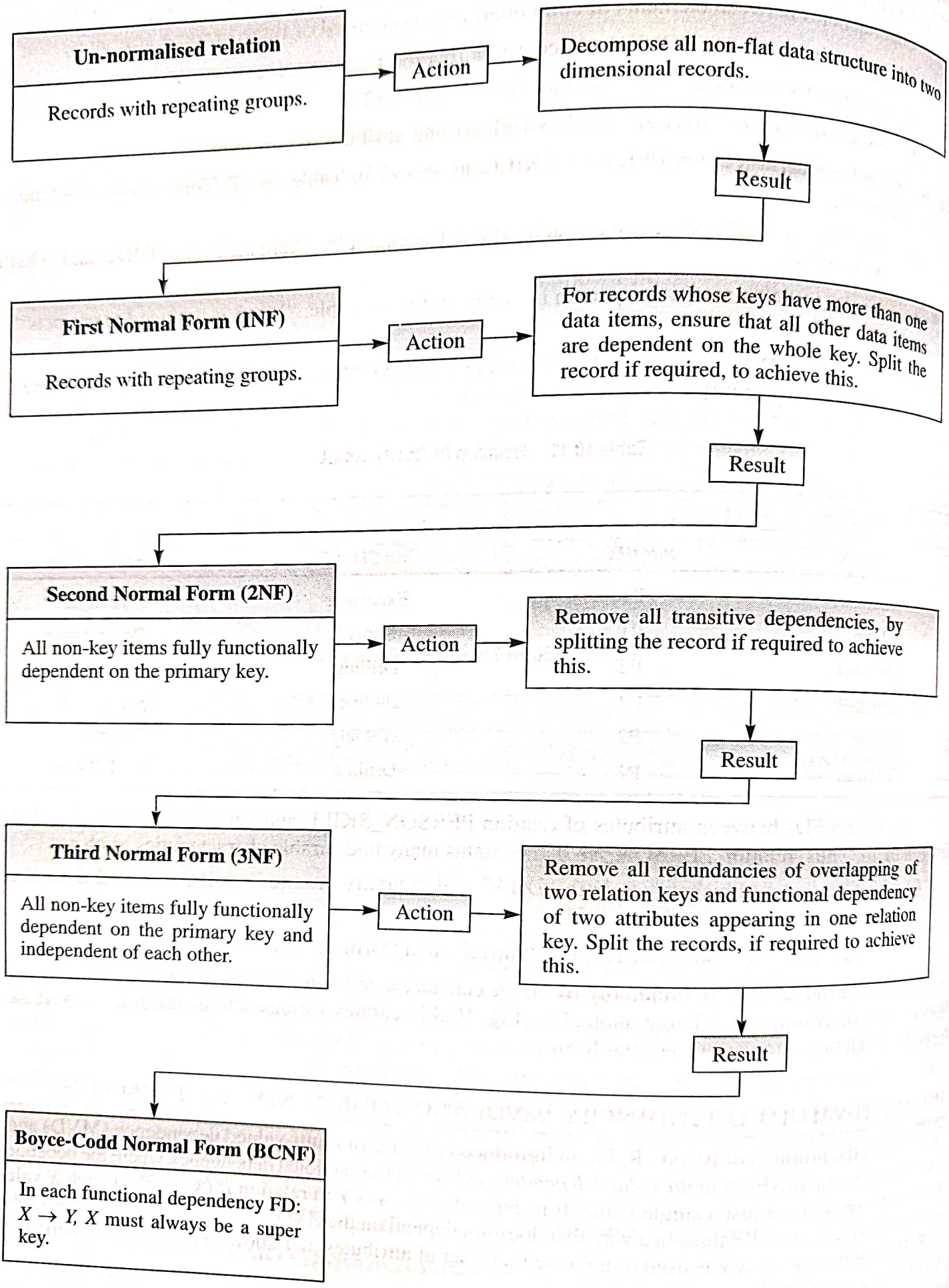
(ii) The fact that "P1" uses "Excavator", "Shovel", and "Drilling" machines.

For most purposes 3NF, or preferably BCNF, is considered to be sufficient to minimise problems arising from update, insertion, and deletion anomalies. Fig. 10.8 illustrates various actions taken to convert an un-normalised relation into various normal forms.

10.5 MULTI-VALUED DEPENDENCIES (MVD) AND FOURTH NORMAL FORM (4NF)

To deal with the problem of BCNF, R. Fagin introduced the idea of multi-valued dependency (MVD) and the fourth normal form (4NF). A *multi-valued dependency (MVD)* is a functional dependency where the dependency may be to a set and not just a single value. It is defined as $X \rightarrow\rightarrow Y$ in relation R (X, Y, Z), if each X value is associated with a set of Y values in a way that does not depend on the Z values. Here X and Y are both subsets of R. The notation $X \rightarrow\rightarrow Y$ is used to indicate that a set of attributes of Y shows a multi-valued dependency (MVD) on a set of attributes of X.

Fig. 10.8 Actions to convert un-normalized relation into normal forms



Thus, informally, MVDs occur when two or more independent multi-valued facts about the same attribute occur within the same relation. There are two important things to be noted in this definition of MVD. Firstly, in order for a relation to contain an 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 make the relation an MVD. For a relation to be MVD, the attributes must be independent of each other.

Functional dependency (FD) concerns itself with the case where one attribute is potentially a 'single-value fact' about another. Multi-valued dependency (MVD), on the other hand, concerns itself with the case where one attribute value is potentially a 'multi-valued fact' about another.

Example 1

Let us consider a relation STUDENT_BOOK, as shown in Table 10.12. The relation STUDENT_BOOK lists students (STUDENT-NAME), the text books (TEXT-BOOK) they have borrowed from library, the librarians (LIBRARIAN) issuing the books and the month and year (MONTH-YEAR) of borrowing. It contains three multi-valued facts about students, namely, the books they have borrowed, the librarians who have issued these books to them and the month and year upon which the books were borrowed.

Table 10.12 Relation STUDENT_BOOK

Relation: STUDENT_BOOK			
STUDENT-NAME	***LIBRARIAN***	***TEXT-BOOK***	***MONTH-YEAR***
Thomas	Jose	Database Management	May, 04
Thomas	Alka	Software Engineering	May ,04
John	Alka	Computer Networking	Sep, 04
Thomas	Jose	Database Management	Oct, 04
Abhishek	Alka	Computer Networking	Nov, 04
Abhishek	Jose	Computer Networking	Dec, 04
Abhishek	Rajesh	Data Communication	Sep, 04

However, these multi-valued facts are not independent of each other. There is clearly an association between librarians, the text books they have issued and the month and year upon which they have issued the books. Therefore, there are no MVDs in the relation. Also, there is no redundant information in this relation. The fact that student "Thomas", for example, has borrowed the book "Database Management" is recorded twice, but these are different borrowings, one in "May, 04" and the other in "Oct, 04" and therefore constitute different items of information.

Table 10.13 Relation COURSE_STUDENT_BOOK

Relation: COURSE_STUDENT_BOOK		
COURS	***STUDENT-NAME***	***TEXT-BOOK***
Computer Engg	Thomas	Database Management
Computer Engg	Thomas	Software Engineering
Computer Engg	John	Database management
Computer Engg	John	Software Engineering
Electronics Engg	Thomas	Digital Electronics
Electronics Engg	Thomas	Pulse Theory
MCA	Abhishek	Computer Networking
MCA	Abhishek	Data Communication

Now, let us consider another relation COURSE_STUDENT_BOOK, as shown in Table 10.13. This relation involves courses (COURSE) being attended by the students, students (STUDENT-NAME) taking the courses and text books (TEXT-BOOKS) applicable for the 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 give 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 relation COURSE_STUDENT_BOOK contains an MVD. Because being MVD, it contains high degree of redundant information, unlike the STUDENT_BOOK relation example of Table 10.12. For example, the fact that the student "Thomas" attends the "Computer Engg" course, is recorded twice, as are the text books prescribed for that course.

The formal definition of MVD specifies that, given a particular value of X, the set of values of Y determined by this value of X is completely determined by X alone and does not depend on the values of the remaining attributes Z of R. Hence, whenever two tuples (rows) exist that have distinct values of Y but the same value of X, these values of Y must be repeated in separate tuples with every distinct value of Z that occurs with that same value of X. Unlike FDs, MVDs are not properties of the information represented by relations. In fact, they depend on the way the attributes are structured into relations. MVDs occur whenever a relation with more than one non-simple domain is normalised.

Example 2

The relation PERSON_SKILL of Table 10.11 is a relation with more than one non-simple domain. Let us suppose that X is PERSON and Y is SKILL-TYPE, then Z becomes a relation key {PROJECT, MACHINE}. Suppose, a particular value of PERSON "John" is selected. Consider all tuples (rows) that have some value of Z, for example, PROJECT = P1 and MACHINE = "Shovel". The value of Y in this tuples (rows) is (‹Programmer›). Consider also all tuples with same value of Y, that is PERSON but with some other value of Z, say, PROJECT = P2 and MACHINE = "Welding". The values of Y in these tuples is again (‹Programmer›). This same set of values of Y is obtained for PERSON = "John", irrespective of the values chosen for PROJECT and MACHINE. Hence $X \rightarrow\rightarrow Y$, or PERSON $\rightarrow\rightarrow$ SKILL-TYPE. It can be verified that in the relation PERSON_SKILL the result is:

$$\text{PROJECT} \rightarrow\rightarrow \text{PERSON, SKILL-TYPE}$$

$$\text{PROJECT} \rightarrow\rightarrow \text{MACHINE}$$

$$\text{PERSON} \rightarrow\rightarrow \text{PROJECT, MACHINE}$$

$Y_{X,Z}$ is defined as the set of Y values, given a set of X and a set of Z values. Thus, in relation PERSON_SKILL, we get:

$$\text{SKILL\_TYPE}_{\text{John, P1, Excavator}} = (\text{‹Programmer›})$$

$$\text{SKILL\_TYPE}_{\text{John, P2, Dumper}} = (\text{‹Programmer›})$$

In a formal definition of MVD the values of attribute Y depend only on attributes X but are independent of the attributes Z. So, when given a value of X, the value of Y will be the same for any two values Z_1 or Z_2, of Z.

The value of Y_1, given a set of values X and Z_1, is Y_{X, Z_1} and the value Y_2, given a set of values X and Z_2, is Y_{x, z_2} / MCD requires that $Y_1 = Y_2$ so $X \rightarrow\rightarrow Y$ in relation R (X, Y, Z) if $Y_{x, z_1} = Y_{x, z_2}$ for any values Z_1, Z_2.

It may be noticed that MVDs always come in pairs. Thus, if $X \rightarrow\rightarrow Y$ in relation R (X, Y, Z), it is also true that $X \rightarrow\rightarrow Z$.

Thus, alternatively it can be stated that if $X \rightarrow\rightarrow Y$ is an MVD in relation $R(X, Y, Z)$, whenever the two tuples (x_1, y_1, z_1) and (x_2, y_2, z_2) are in R, the tuples (x_1, y_1, z_2) and (x_2, y_2, z_1) must also be in R. In this definition, X, Y and Z represent sets of attributes rather than individual attributes.

Example 3

Let us examine relation PERSON_SKILL, as shown in Fig. 10.9 which does not contain the MVD. As we discussed that for MVD

$$\text{PERSON} \rightarrow\rightarrow \text{PROJECT}$$

But, we observe from Fig. 10.9 that

Fig. 10.9 Normal form of relation PERSON_SKILL not containing MVD

Relation: PEARSON_SKILL

PEARSON (X)	PROJECTS (Z)		SKILL-TYPE (Y)
	PROJECT	MACHINE	
Thomas	P1	Excavator Shovel Drilling	Analyst DBA
John	P1	Excavator Shovel Drilling	Programmer
	P2	Drilling Dumper Welding	
Abhishek	P2	Drilling Dumper Welding	DBA Quality Auditor

Whereas,

$$\text{PROJECT}_{\text{John, Drilling, Programmer}} = (<\text{P1}>, <\text{P2}>)$$

$$\text{PROJECT}_{\text{John, Excavator, Programmer}} = (<\text{P1}>)$$

Thus, $\text{PROJECT}_{\text{John,Excavator,Programmer}} = (< \text{P1} >)$ does not equal $\text{PROJECT}_{\text{John,Drilling,Programmer}} = (< \text{P1} >, < \text{P2} >)$ which it would have to hold if PEARSON $\rightarrow\!\!\rightarrow$ PROJECT. But, if MACHINE is projected out of relation PERSON_SKILL, then PEARSON $\rightarrow\!\!\rightarrow$ PROJECT will become an MVD. An MVD that does not hold in the relation R, but holds for a projection on some subset of the attributes of relation R is sometimes called *embedded* MVD of R.

Alike trivial FDs, there are trivial MVDs also. An MVD $X \rightarrow\!\!\rightarrow Y$ in relation R is called a trivial MVD if

(a) Y is a subset of X, or

(b) $X \cup Y = R$.

It is called trivial because it does not specify any significant or meaningful constraint on R. An MVD that satisfies neither (a) nor (b) is called *non-trivial MVD*.

Like trivial FDs, there are two kinds of trivial MVDs as follows:

(a) $X \rightarrow\!\!\rightarrow \phi$, where ϕ is an empty set of attributes.

(b) $X \rightarrow\!\!\rightarrow A - X$, where A comprises all the attributes in a relation.

Both these types of trivial MVDs hold for any set of attributes of R and therefore can serve no purpose as design criteria.

10.5.1 Properties of MVDs

Berri described the relevant rules to derive MVDs. Following four axioms were proposed to derive a closure D^+ of MVDs:

Rule 1	***Reflexivity (inclusion)***	If $Y \subset X$, then $X \rightarrow\!\!\rightarrow Y$.
Rule 2	***Augmentation:***	If $X \rightarrow\!\!\rightarrow Y$ and $W \subset U$ and $V \subset W$, then $WX \rightarrow\!\!\rightarrow VY$.
Rule 3	***Transitivity:***	If $X \rightarrow\!\!\rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow\!\!\rightarrow Z - Y$.
Rule 4	***Complementation:***	If $X \rightarrow\!\!\rightarrow Y$, then $X \rightarrow\!\!\rightarrow U - X - Y$ holds.

In the above rules, X, Y, and Z all are sets of attributes of a relation R and U is the set of all the attributes of R. These four axioms can be used to derive the closure of a set D^+, of D of multi-valued dependencies. It can be noticed that there are similarities between the Armstrong's axioms for FDs and Berri's axioms for MVDs. Both have reflexivity, augmentation, and transitivity rules. But, the MVD set also has a complementation rule.

Following additional rules can be derived from the above Berri's axioms to derive closure of a set of FDs and MVDs:

Rule 5	***Intersection:***	If $X \rightarrow\!\!\rightarrow Y$ and $X \rightarrow\!\!\rightarrow Z$, then $X \rightarrow\!\!\rightarrow Y \cap Z$.
Rule 6	***Pseudo-transitivity:***	If $X \rightarrow\!\!\rightarrow Y$ and $YW \rightarrow\!\!\rightarrow Z$, then $XW \rightarrow (Z - WY)$.
Rule 7	***Union:***	If $X \rightarrow\!\!\rightarrow Y$ and $X \rightarrow\!\!\rightarrow Z$, then $X \rightarrow\!\!\rightarrow YZ$.
Rule 8	***Difference:***	If $X \rightarrow\!\!\rightarrow Y$ and $X \rightarrow\!\!\rightarrow Z$, then $X \rightarrow\!\!\rightarrow Y - Z$ and $X \rightarrow\!\!\rightarrow Z - Y$.

Further additional rules can be derived from above rules, which are as follows:

Rule 9	***Replication:***	If $X \rightarrow Y$, then $X \rightarrow\rightarrow Y$.
Rule 10	***Coalescence:***	If $X \rightarrow\rightarrow Y$ and $Z \subset Y$ and there is a W such that $W \subset U$ and $W \cap Y = null$ and $W \rightarrow Z$, then $X \rightarrow Z$.

10.5.2 Fourth Normal Form (4NF)

A relation R is said to be in *fourth normal form (4NF)* if it is in BCNF and for every non-trivial $MVD(X \rightarrow\rightarrow Y)$ in F^+, X is a super key for R. The fourth normal form (4NF) is concerned with dependencies between the elements of compound keys composed of three or more attributes. The 4NF eliminates the problems of 3NF. 4NF is violated when a relation has undesirable MVDs and hence can be used to identify and decompose such relations.

Example 1

Let us consider a relation EMPLOYEE, as shown in Fig. 10.10. A tuple in this relation represents the fact that an employee (EMP-NAME) works on the project (PROJ-NAME) and has a dependent (DEPENDENT-NAME). This relation is not in 4NF because in the non-trivial MVDs EMP-NAME $\rightarrow\rightarrow$ PROJ-NAME and EMP-NAME $\rightarrow\rightarrow$ DEPENDENT-NAME, EMP-NAME is not a super key of EMPLOYEE.

Now the relation EMPLOYEE is decomposed into EMP_PROJ and EMP_DEPENDENTS. Thus, both EMP_PROJ and EMP_DEPENDENT are in 4NF, because the MVDs EMP-NAME $\rightarrow\rightarrow$ PROJ-NAME in EMP_PROJ and EMP-NAME $\rightarrow\rightarrow$ DEPENDENT-NAME in EMP_DEPENDENTS are trivial MVDs. No other non-trivial MVDs hold in either EMP_PROJ or EMP_DEPENDENTS. No FDs hold in these relation schemas either.

Example 2

Similarly, the relation PERSON_SKILL of Fig. 10.9 is not in 4NF because it has the non-trivial MVDs PROJECT $\rightarrow\rightarrow$ MACHINE, but PROJECT is not a super key. To convert this relation into 4NF, it is necessary to decompose relation PERSON_SKILL into the following relations:

R_1(PROJECT, MACHINE)

R_2(PERSON, SKILL-TYPE)

R_3(PERSON, PROJECT)

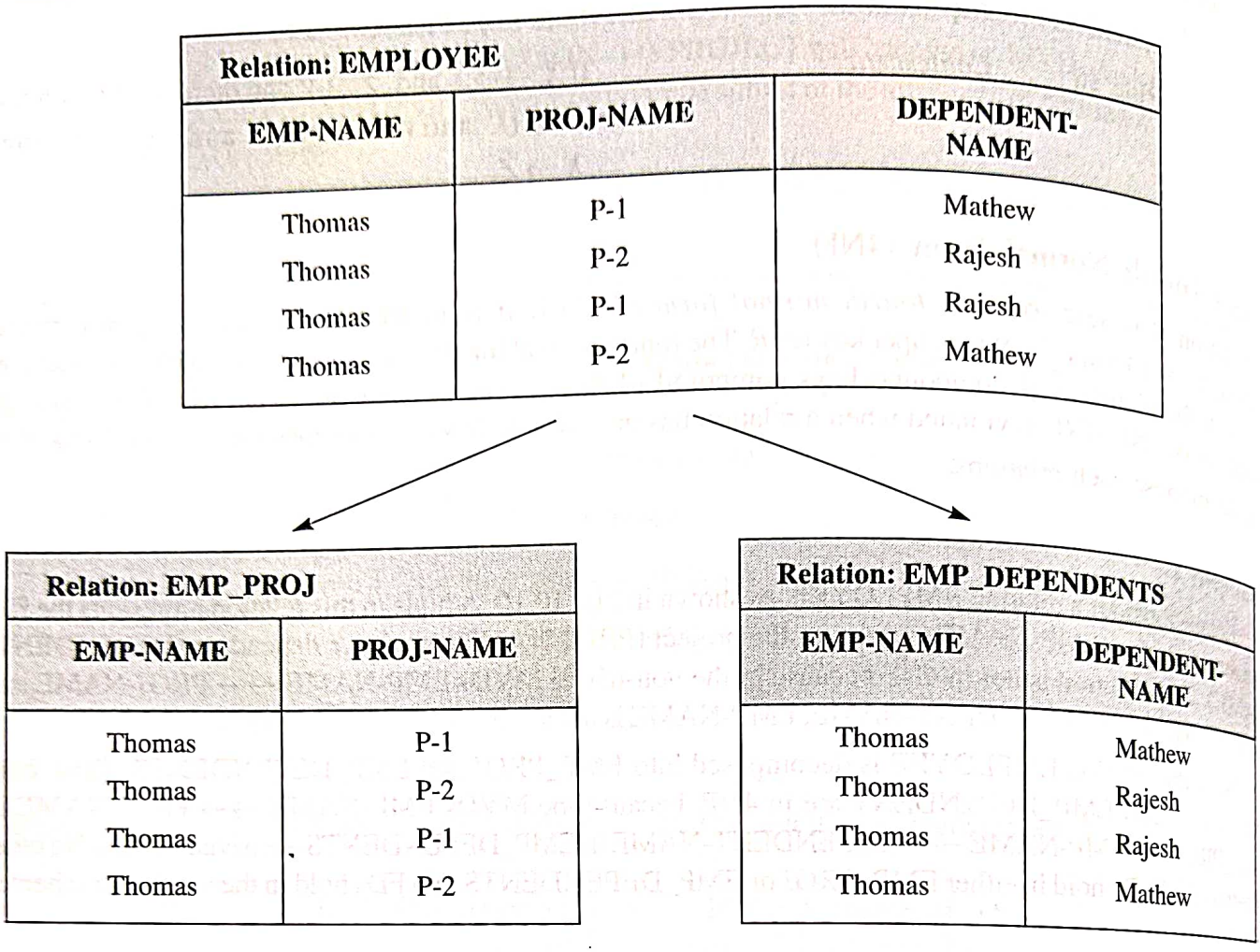
Example 3

Let us consider a relation R (A, B, C, D, E, F) with FDs and with MVDs $A \rightarrow\rightarrow B$ and $CD \rightarrow\rightarrow EF$.

Let us decompose the relation R into two relations as $R_1(A, B)$ and $R_Z(A, C, D, E, F)$ by applying the MVD: $A \rightarrow\rightarrow B$ and its compliment $A \rightarrow CDEF$. The relation R_1 is now in 4NF because $A \rightarrow\rightarrow B$ is trivial and is the only MVD in the relation. The relation R_Z, however is still in BCNF because of the nontrivial MVD: $CD \rightarrow\rightarrow EF$.

Now, R_Z is decomposed into relations R_{Z_1} (C, D, E, F) and R_{Z_2} (C, D, A) by applying the MVD: $CD \rightarrow\rightarrow EF$ and its compliment $CD \rightarrow\rightarrow A$. Both the decomposed relations R_{Z_1} and R_{Z_2} are now in 4NF.

Fig. 10.10 A relation EMPLOYEE decomposed into EMP_PROJ and EMP_DEPENDENTS



Relation: EMPLOYEE

EMP-NAME	PROJ-NAME	DEPENDENT-NAME
Thomas	P-1	Mathew
Thomas	P-2	Rajesh
Thomas	P-1	Rajesh
Thomas	P-2	Mathew

Relation: EMP_PROJ

EMP-NAME	PROJ-NAME
Thomas	P-1
Thomas	P-2
Thomas	P-1
Thomas	P-2

Relation: EMP_DEPENDENTS

EMP-NAME	DEPENDENT-NAME
Thomas	Mathew
Thomas	Rajesh
Thomas	Rajesh
Thomas	Mathew

10.5.3 Problems with MVDs and 4NF

FDs, MVDs and 4NF are not sufficient to identify all data redundancies. Let us consider a relation PERSONS_ON_JOB_SKILLS, as shown in Table 10.14. This relation stores information about people applying all their skills to the jobs to which they are assigned. But, they use particular or all skills only when the job needs that skill.

Table 10.14 Relation PERSON_ON_JOB_SKILL in BCNF and 4NF

Relation: PERSONS_ON_JOB_SKILLS

PERSON	*SKILL-TYPE*	*JOB*
Thomas	Analyst	J-1
Thomas	Analyst	J-2
Thomas	DBA	J-2
Thomas	DBA	J-3
John	DBA	J-1
Abhishek	Analyst	J-1

The relation PERSONS_ON_JOB_SKILLS of Table 10.14 is in BCNF and 4NF. It can lead to anomalies because of the dependencies between the joins. For example, person "Thomas" who possesses skills "Analyst" and "DBA" applies them to job J-2, as J-2 needs both these skills. The same person "Thomas" applies skill "Analyst" only to job J-1, as job J-1 needs only skill "Analyst" and not skill "DBA". Thus, if we delete ‹Thomas, DBA, J-2›, we must also delete ‹Thomas, Analyst, J-2›, because persons must apply all their skills to a job if that requires those skills.

10.6 JOIN DEPENDENCIES AND FIFTH NORMAL FORM (5NF)

The anomalies of MVDs and are eliminated by join dependency (JD) and 5NF.

10.6.1 Join Dependencies (JD)

A *join dependency* (*JD*) can be said to exist if the join of R_1 and R_2 over C is equal to relation R. Where, R_1 and R_2 are the decompositions $R_1(A, B, C)$, and R_2 (C,D) of a given relations R (A, B, C, D). Alternatively, R_1 and R_2 is a lossless decomposition of R. In other words, $*(A, B, C, D), (C, D)$ will be a join dependency of R if the join of the join's attributes is equal to relation R. Here, $*(R_1, R_2, R_3, ...)$ indicates that relations R_1, R_2, R_3 and so on are a join dependency (JD) of R. Therefore, a necessary condition for a relation R to satisfy a JD $*(R_1, R_2, ..., R_n)$ is that

$$R = R_1 \cup R_2 \cup ..... \cup R_n$$

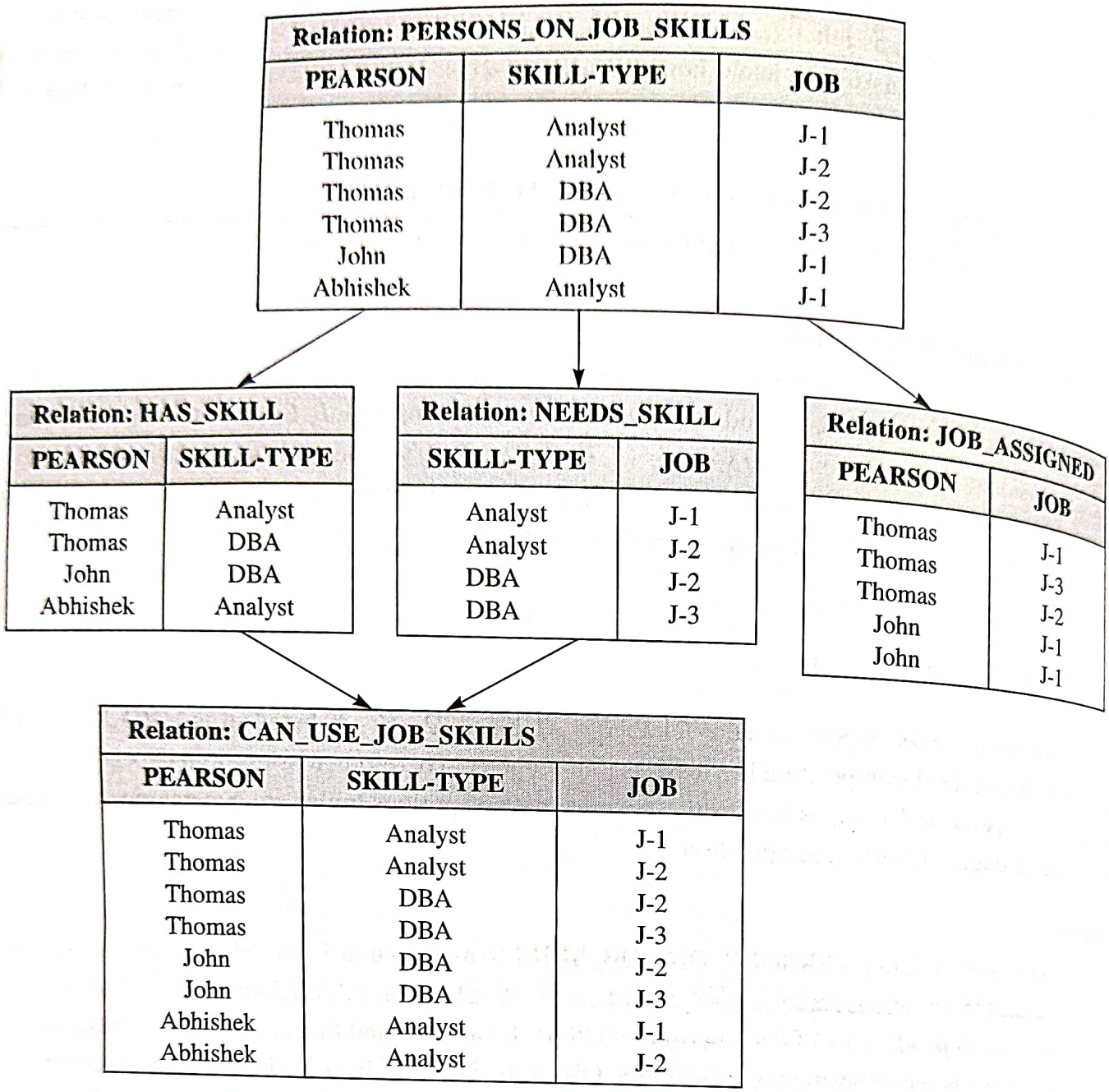
Thus, whenever we decompose a relation R into $R_1 = XUY$ and $R2 = (R - Y)$ based on an MVD $X \rightarrow\rightarrow Y$ that holds in relation R, the decomposition has lossless join property. Therefore, lossless-join dependency can be defined as a property of decomposition, which ensures that no spurious tuples are generated when relations are returned through a natural join operation.

Example 1

Let us consider a relation PERSONS_ON_JOB_SKILLS, as shown in Table 10.14. This relation can be decomposed into three relations namely, HAS_SKILL, NEEDS_SKILL and ASSIGNED_TO_JOBS. Fig. 10.11 illustrates the join dependencies of decomposed relations. It can be noted that none of the two decomposed relations are a lossless decomposition of PERSONS_ON_JOB_SKILLS. In fact, a join of all three decomposed relations yields a relation that has the same data as does the original relation PERSONS_ON_JOB_SKILLS. Thus, each relation acts as a constraint on the join of the other two relations.

Now, if we join decomposed relations HAS_SKILL and NEEDS_SKILL, a relation CAN_USE_JOB_SKILL is obtained, as shown in Fig. 10.11. This relation stores the data about persons who have skills applicable to a particular job. But, each person who has a skill required for a particular job need not be assigned to that job. The actual job assignments are given by the relation JOB_ASSIGNED. When this relation is joined with HAS_SKILL, a relation is obtained that will contain all possible skills that can be applied to each job. This happens because persons assigned to that job, possesses those skills. However, some of the jobs do not require all the skills. Thus, redundant tuples (rows) that show unnecessary SKILL-TYPE and JOB combinations are removed by joining with relation NEEDS_SKILL.

Fig. 10.11 Join dependencies of relation PERSONS_ON_JOB_SKILLS



Relation: PERSONS_ON_JOB_SKILLS

PEARSON	SKILL-TYPE	JOB
Thomas	Analyst	J-1
Thomas	Analyst	J-2
Thomas	DBA	J-2
Thomas	DBA	J-3
John	DBA	J-1
Abhishek	Analyst	J-1

Relation: HAS_SKILL

PEARSON	SKILL-TYPE
Thomas	Analyst
Thomas	DBA
John	DBA
Abhishek	Analyst

Relation: NEEDS_SKILL

SKILL-TYPE	JOB
Analyst	J-1
Analyst	J-2
DBA	J-2
DBA	J-3

Relation: JOB_ASSIGNED

PEARSON	JOB
Thomas	J-1
Thomas	J-3
Thomas	J-2
John	J-1
John	J-1

Relation: CAN_USE_JOB_SKILLS

PEARSON	SKILL-TYPE	JOB
Thomas	Analyst	J-1
Thomas	Analyst	J-2
Thomas	DBA	J-2
Thomas	DBA	J-3
John	DBA	J-2
John	DBA	J-3
Abhishek	Analyst	J-1
Abhishek	Analyst	J-2

10.6.2 Fifth Normal Form (5NF)

A relation is said to be in *fifth normal form* (*5NF*) if every join dependency is a consequence of its relation (candidate) keys. Alternatively, for every non-trivial join dependency $*(R_1, R_2, R_3)$ each decomposed relation R_i is a super key of the main relation R. 5NF is also called ***project-join normal form (PJNM)***.

There are some relations, who cannot be decomposed into two or higher normal form relations by means of projections as discussed in 1NF, 2NF, 3NF and BCNF. Such relations are decomposed into three or more relations, which can be reconstructed by means of a three-way or more join operation. This is called fifth normal form (5NF). The 5NF eliminates the problems of 4NF. 5NF allows for relations with join dependencies. Any relation that is in 5NF, is also in other normal forms namely 2NF, 3NF and 4NF. 5Nf is mainly used from theoretical point of view and not for practical database design.

Example 1

Let us consider the relation PERSONS_ON_JOB_SKILLS of Fig. 10.11. The three relations are

HAS_SKILL (PERSON, SKILL-TYPE)

NEEDS_SKILL (SKILL-TYPE, JOB)

JOB_ASSIGNED (PERSON, JOB))

Now by applying the definition of 5NF, the join dependency is given as:

*((PERSON, SKILL-TYPE), (SKILL-TYPE, JOB), (PERSON, JOB))

The above statement is true because a join relation of these three relations is equal to the original relation PERSONS_ON_JOB_SKILLS. The consequence of these join dependencies is that the SKILL-TYPE, JOB or PERSON, is not relation key, and hence the relation is not in 5NF. Now suppose, the second tuple (row 2) is removed form relation PERSONS_ON_JOB_SKILLS, a new relation is created that no longer has any join dependencies. Thus the new relation will be in 5NF.