UNIT 3 DATABASE INTEGRITY AND NORMALISATION

Structure **Page Nos.**

3.0 Introduction 56
3.1 Objectives 56
3.2 Relational Database Integrity 57
3.2.1 The Keys
3.2.2 Referential Integrity
3.2.3 Entity Integrity
3.3 Redundancy and Associated Problems 62
3.4 Single-Valued Dependencies 64
3.5 Single-Valued Normalisation 66
3.5.1 The First Normal Form
3.5.2 The Second Normal Form
3.5.3 The Third Normal Form
3.5.4 Boyce Codd Normal Form
3.6 Desirable Properties of Decomposition 72
3.6.1 Attribute Preservation
3.6.2 Lossless-join Decomposition
3.6.3 Dependency Preservation
3.6.4 Lack of redundancy
3.7 Rules of Data Normalisation 74
3.7.1 Eliminate Repeating Groups
3.7.2 Eliminate Redundant Data
3.7.3 Eliminate Columns Not Dependent on Key
3.8 Summary 76
3.9 Answers/Solutions 77

3.0 INTRODUCTION

In the previous unit, we have discussed relations. Relations form the database. They must satisfy some properties, such as no duplicate tuples, no ordering of tuples, and atomic attributes, etc. Relations that satisfy these basic requirements may still have some undesirable characteristics such as data redundancy, and anomalies.

What are these undesirable characteristics and how can we eliminate them from the database system? This unit is an attempt to answer this question. However, please note that most of these undesirable properties do not arise if the database modeling has been carried out very carefully using some technique like the Entity-Relationship Model that we have discussed in the previous unit. It is still important to use the techniques in this unit to check the database that has been obtained and ensure that no mistakes have been made in modeling.

The central concept in these discussions is the concept of Database integrity, the notion of functional dependency, which depends on understanding the semantics of the data and which deals with what information in a relation is dependent on what other information in the relation. Our prime objective in this unit is to define the concept of data integrity, functional dependence and then define normal forms using functional dependencies and using other types of data dependencies.

3.1 OBJECTIVES

After going through this unit you should be able to

- define the concept of entity integrity;

- describe relational database referential integrity constraints;
- define the concept of functional dependency, and
- show how to use the dependencies information to decompose relations; whenever necessary to obtain relations that have the desirable properties that we want without losing any of the information in the original relations.

3.2 RELATIONAL DATABASE INTEGRITY

A database is a collection of data. But, is the data stored in a database trustworthy? To answer that question we must first answer the question. What is integrity?

Integrity simply means to maintain the consistency of data. Thus, integrity constraints in a database ensure that changes made to the database by authorised users do not compromise data consistency. Thus, integrity constraints do not allow damage to the database.

There are primarily two integrity constraints: the entity integrity constraint and the referential integrity constraint. In order to define these two, let us first define the term Key with respect to a Database Management System.

3.2.1 The Keys

Candidate Key: In a relation R, a candidate key for R is a subset of the set of attributes of R, which have the following two properties:

(1)	*Uniqueness*	No two distinct tuples in R have the same value for the candidate key
(2)	*Irreducible*	No proper subset of the candidate key has the uniqueness property that is the candidate key.

Every relation must have at least one candidate key which cannot be reduced further. Duplicate tuples are not allowed in relations. Any candidate key can be a composite key also. For Example, (student-id + course-id) together can form the candidate key of a relation called marks (student-id, course-id, marks).

Let us summarise the properties of a candidate key.

Properties of a candidate key

- A candidate key must be unique and irreducible
- A candidate may involve one or more than one attributes. A candidate key that involves more than one attribute is said to be composite.

But why are we interested in candidate keys?

Candidate keys are important because they provide the basic **tuple-level identification** mechanism in a relational system.

For example, if the enrolment number is the candidate key of a STUDENT relation, then the answer of the query: "Find student details from the STUDENT relation having enrolment number A0123" will output at most one tuple.

Primary Key

The primary key is the candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set. The remaining candidate keys, if any, are called **alternate keys.**

Foreign Keys

Let us first give you the basic definition of foreign key.

Let R2 be a relation, then a foreign key in R2 is a subset of the set of attributes of R2, such that:

1. There exists a relation R1 (R1 and R2 not necessarily distinct) with a candidate key, and
2. For all time, each value of a foreign key in the current state or instance of R2 is identical to the value of Candidate Key in some tuple in the current state of R1.

The definition above seems to be very heavy. Therefore, let us define it in more practical terms with the help of the following example.

Example 1

Assume that in an organisation, an employee may perform different roles in different projects. Say, RAM is doing coding in one project and designing in another. Assume that the information is represented by the organisation in three different relations named EMPLOYEE, PROJECT and ROLE. The ROLE relation describes the different roles required in any project.

Assume that the relational schema for the above three relations are:

EMPLOYEE (EMPID, Name, Designation)
PROJECT (PROJID, Proj_Name, Details)
ROLE (ROLEID, Role_description)

In the relations above EMPID, PROJID and ROLEID are unique and not NULL, respectively. As we can clearly see, we can identify the complete instance of the entity set employee through the attribute EMPID. Thus EMPID is the primary key of the relation EMPLOYEE. Similarly PROJID and ROLEID are the primary keys for the relations PROJECT and ROLE respectively.

Let ASSIGNMENT is a relationship between entities EMPLOYEE and PROJECT and ROLE, describing which employee is working on which project and what the role of the employee is in the respective project. Figure 1 shows the E-R diagram for these entities and relationships.

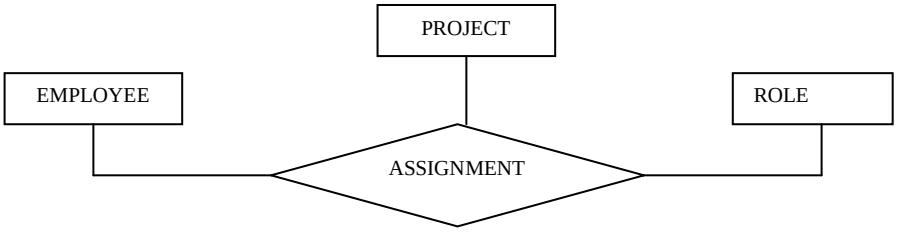


Figure 1: E-R diagram for employee role in development team

Let us consider sample relation instances as:

EMPLOYEE

EMPID	Name	Designation
101	RAM	Analyst
102	SITA	Receptionist
103	ARVIND	Manager

PROJECT

PROJID	Proj_name	Details
TCS	Traffic Control System	For traffic shaping.
LG	Load Generator	To simulate load for input in TCS.
B++1	B++_TREE	ISS/R turbo sys

ROLE

ROLEID	Role_descrption
1000	Design
2000	Coding
3000	Marketing

ASSIGNMENT

PROJID	Proj_name	Details
101	TCS	1000
101	LG	2000
102	B++1	3000

Figure 2: An example relation

We can define the relational scheme for the relation ASSIGNMENT as follows:

ASSIGNMENT (EMPID, PROJID, ROLEID)
Please note now that in the relation ASSIGNMENT (as per the definition to be taken as R2) EMPID is the foreign key in ASSIGNMENT relation; it references the relation EMPLOYEE (as per the definition to be taken as R1) where EMPID is the primary key. Similarly PROJID and ROLEID in the relation ASSIGNMENT are **foreign keys** referencing the relation PROJECT and ROLE respectively.

Now after defining the concept of foreign key, we can proceed to discuss the actual integrity constraints namely Referential Integrity and Entity Integrity.

3.2.2 Referential Integrity

It can be simply defined as:

The database must not contain any unmatched foreign key values.

The term "unmatched foreign key value" means a foreign key value for which there does not exist a **matching value** of the relevant candidate key in the relevant target (referenced) relation. For example, any value existing in the EMPID attribute in ASSIGNMENT relation must exist in the EMPLOYEE relation. That is, the only EMPIDs that can exist in the EMPLOYEE relation are 101, 102 and 103 for the present state/ instance of the database given in *Figure 2*. If we want to add a tuple with EMPID value 104 in the ASSIGNMENT relation, it will cause violation of referential integrity constraint. Logically it is very obvious after all the employee 104 does not exist, so how can s/he be assigned any work.

Database modifications can cause violations of referential integrity. We list here the test we must make for each type of database modification to preserve the referential-integrity constraint:

Delete

During the deletion of a tuple two cases can occur:

Deletion of tuple in relation having the foreign key: In such a case simply delete the desired tuple. For example, in ASSIGNMENT relation we can easily delete the first tuple.

Deletion of the target of a foreign key reference: For example, an attempt to delete an employee tuple in EMPLOYEE relation whose EMPID is 101. This employee appears not only in the EMPLOYEE but also in the ASSIGNMENT relation. Can this tuple be deleted? If we delete the tuple in EMPLOYEE relation then two unmatched tuples are left in the ASSIGNMENT relation, thus causing violation of referential integrity constraint. Thus, the following two choices exist for such deletion:

RESTRICT – The delete operation is "restricted" to only the case where there are no such matching tuples. For example, we can delete the EMPLOYEE record of EMPID 103 as no matching tuple in ASSIGNMENT but not the record of EMPID 101.

CASCADE – The delete operation "cascades" to delete those matching tuples also.

For example, if the delete mode is CASCADE then deleting employee having EMPID as 101 from EMPLOYEE relation will also cause deletion of 2 more tuples from ASSIGNMENT relation.

Insert

The insertion of a tuple in the target of reference does not cause any violation. However, insertion of a tuple in the relation in which, we have the foreign key, for example, in ASSIGNMENT relation it needs to be ensured that all matching target candidate key exist; otherwise the insert operation can be rejected. For example, one of the possible ASSIGNMENT insert operations would be (103, LG, 3000).

Modify

Modify or update operation changes the existing values. If these operations change the value that is the foreign key also, the only check required is the same as that of the Insert operation.

What should happen to an attempt to update a candidate key that is the target of a foreign key reference? For example, an attempt to update the PROJID "LG" for which there exists at least one matching ASSIGNMENT tuple? In general there are the same possibilities as for DELETE operation:

RESTRICT: The update operation is "restricted" to the case where there are no matching ASSIGNMENT tuples. (it is rejected otherwise).

CASCADE – The update operation "cascades" to update the foreign key in those matching ASSIGNMENT tuples also.

3.2.3 Entity Integrity

Before describing the second type of integrity constraint, viz., Entity Integrity, we should be familiar with the concept of **NULL.**

Basically, NULL is intended as a basis for dealing with the problem of missing information. This kind of situation is frequently encountered in the real world. For example, historical records sometimes have entries such as "Date of birth unknown", or police records may include the entry "Present whereabouts unknown." Hence it is necessary to have some way of dealing with such situations in database systems. Thus Codd proposed an approach to this issue that makes use of special markers called NULL to represent such missing information.

A given attribute in the relation might or might not be allowed to contain NULL. But, can the Primary key or any of its components (in case of the primary key is a composite key) contain a NULL? To answer this question an **Entity Integrity Rule** states: No component of the primary key of a relation is allowed to accept NULL. In other words, the definition of every attribute involved in the primary key of any basic relation must explicitly or implicitly include the specifications of NULL NOT ALLOWED.

Foreign Keys and NULL

Let us consider the relation:

DEPT		
DEPT ID	**DNAME**	**BUDGET**
D1	Marketing	10M
D2	Development	12M
D3	Research	5M

EMP			
EMP ID	**ENAME**	**DEPT ID**	**SALARY**
E1	Rahul	D1	40K
E2	Aparna	D1	42K
E3	Ankit	D2	30K
E4	Sangeeta		35K

Suppose that Sangeeta is not assigned any Department. In the EMP tuple corresponding to Sangeeta, therefore, there is no genuine department number that can serve as the appropriate value for the DEPTID foreign key. Thus, one cannot determine DNAME and BUDGET for Sangeeta's department as those values are NULL. This may be a real situation where the person has newly joined and is

undergoing training and will be allocated to a department only on completion of the training. Thus, NULL in foreign key values may not be a logical error.

So, the foreign key definition may be redefined to include NULL as an acceptable value in the foreign key for which there is no need to find a matching tuple.

Are there any other constraints that may be applicable on the attribute values of the entities? Yes, these constraints are basically related to the domain and termed as the domain constraints.

Domain Constraints

Domain constraints are primarily created for defining the logically correct values for an attribute of a relation. The relation allows attributes of a relation to be confined to a range of values, for example, values of an attribute age can be restricted as Zero to 150 or a specific type such as integers, etc. These are discussed in more detail in Block 2 Unit 1.

Check Your Progress 1

Consider supplier-part-project database;

SUPPLIERS

S.NO	SNAME	CITY
S1	Smita	Delhi
S2	Jim	Pune
S3	Ballav	Pune
S4	Sita	Delhi
S5	Anand	Agra

PARTS

P.NO	PNAME	COLOUR	CITY
P1	Nut	Red	Delhi
P2	Bolt	Blue	Pune
P3	Screw	White	Mumbai
P4	Screw	Blue	Delhi
P5	Cam	Brown	Pune
P6	Cog	Grey	Delhi

PROJECTS

PROJNO	JNAME	CITY
PROJ1	Sorter	Pune
PROJ2	Display	Mumbai
PROJ3	OCR	Agra
PROJ4	Console	Agra
PROJ5	RAID	Delhi
PROJ6	EDS	Udaipur
PROJ7	Tape	Delhi

SUP_PAR_PROJ

SNO	PNO	PROJ NO	QUANTITY
S1	P1	PROJ1	200
S1	P1	PROJ4	700
S2	P3	PROJ2	400
S2	P2	PROJ7	200
S2	P3	PROJ3	500
S3	P3	PROJ5	400
S3	P4	PROJ3	500
S3	P5	PROJ3	600
S3	P6	PROJ4	800
S4	P6	PROJ2	900
S4	P6	PROJ1	100
S4	-	PROJ7	200
S5	P5	PROJ5	300
S6	P4	PROJ6	400

1) What are the Candidate keys and PRIMARY key to the relations?

……

……

……

……

2) What are the entity integrity constraints? Are there any domain constraints?

……

……

……

……

3) What are the referential integrity constraints in these relations?

……

……

……

……

4) What are referential actions you would suggest for these referential integrity constraints?

……

……

……

……

3.3 REDUNDANCY AND ASSOCIATED PROBLEMS

Let us consider the following relation STUDENT.

Enrolment no	Sname	Address	Cno	Cname	Instructor	Office
050112345	Rahul	D-27, main Road Ranchi	MCS–011	Problem Solution	Nayan Kumar	102
050112345	Rahul	D-27, Main Road Ranchi	MCS–012	Computer Organisation	Anurag Sharma	105
050112345	Rahul	D-27, Main Road Ranchi	MCS-014	SSAD	Preeti Anand	103
050111341	Aparna	B-III, Gurgaon	MCS-014	SSAD	Preeti Anand	103

Figure 3: A state of STUDENT relation

The above relation satisfies the properties of a relation and is said to be in first normal form (or 1NF). Conceptually it is convenient to have all the information in one relation since it is then likely to be easier to query the database. But the relation above has the following undesirable features:

Data Redundancy-A lot of information is being repeated in the relation. For example, the information that MCS-014 is named SSAD is repeated, address of Rahul is "D-27, Main road, Ranchi" is being repeated in the first three records. Please find the other duplicate information. So we find that the student name, address, course name, instructor name and office number are being repeated often, thus, the table has **Data Redundancy**.

Please also note that every time we wish to insert a student record for a subject taken by him/her, say for example, MCS-013, we must insert the name of the course as well as the name and office number of its instructor. Also, every time we insert a new course we must repeat the name and address of the student who has taken it. This repetition of information results in problems in addition to the wastage of space. Check these problems in the STUDENT relation. What are these problems? Let us define them.

These problems are called database anomalies. There are three anomalies in database systems:

1. *Update Anomaly*: This anomaly is caused due to data redundancy. Redundant information makes updates more difficult since, for example, changing the name of the instructor of MCS-014 would require that all tuples containing MCS-014 enrolment information be updated. If for some reason, all tuples are not updated, we might have a database that gives two names of instructor for the subject MCS-014 which is inconsistent information. This problem is called update anomaly. An update anomaly results in **data inconsistency.**

2. *Insertion Anomaly*: Inability to represent certain information-The primary key of the above relation be (enrolment number, Cno). Any new tuple to be inserted in the relation must have a value for the primary key since entity integrity constraint requires that a key may not be totally or partially NULL. However, in the given relation if one wanted to insert the number and name of a new course in the database, it would not be possible until a student enrols in that course. Similarly information about a new student cannot be inserted in the database until the student enrols in a course. These problems are called insertion anomalies.

3. *Deletion Anomalies*: Loss of Useful Information: In some instances, useful information may be lost when a tuple is deleted. For example, if we delete the tuple corresponding to student 050111341 enrolled for MCS-014, we will lose relevant information about the student viz. enrolment number, name and address of this student. Similarly deletion of tuple having Sname "Rahul" and Cno 'MCS-012" will result in loss of information that MCS-012 is named computer organisation having an instructor "Anurag Sharma", whose office number is 105. This is called deletion anomaly.

The anomalies arise primarily because the relation STUDENT has information about students as well as subjects. One solution to the problems is to decompose the relation into two or more smaller relations. But what should be the basis of this decomposition? To answer the questions let us try to formulate how data is related in the relation with the help of the following *Figure 4*:

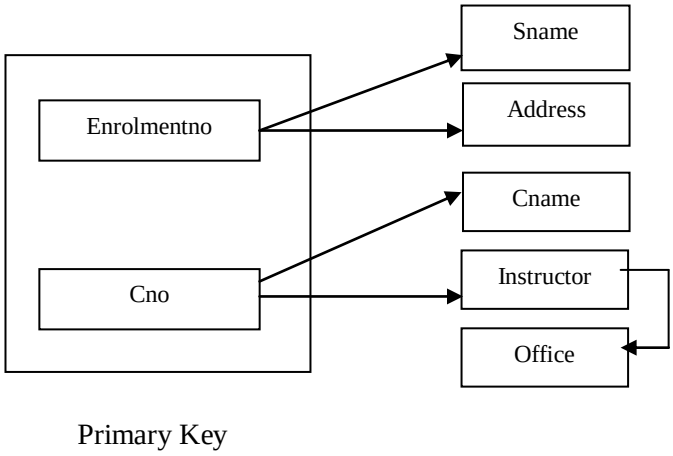


Figure 4: The dependencies of relation in Figure 3

Please note that the arrows in *Figure 4* are describing data inter-relationship. For example, enrolmentno column is unique for a student so if we know the enrolment no of a student we can uniquely determine his/her name and address. Similarly, the course code (Cno) uniquely determines course name (Cname) and Instructor (we are assuming that a course is taught by only one instructor). Please also note one important interrelationship in *Figure 4* that is, the Office (address) of an instructor is dependent on Instructor (name), assuming unique instructor names. The root cause of the presence of anomalies in a relation is determination of data by the components of the key and non-key attributes.

Normalisation involves decomposition of a relation into smaller relations based on the concept of functional dependence to overcome undesirable anomalies.

Normalisation sometimes can affect performance. As it results in decomposition of tables, some queries desire to join these tables to produce the data once again. But such performance overheads are minimal as Normalisation results in minimisation of data redundancy and may result in smaller relation sizes. Also DBMSs implements optimised algorithms for joining of relations and many indexing schemes that reduce the load on joining of relations. In any case the advantages of normalisaiton normally overweigh the performance constraints. Normalisation does lead to more efficient updates since an update that may have required several tuples to be updated, whereas normalised relations, in general, require the information updating at only one place.

A relation that needs to be normalised may have a very large number of attributes. In such relations, it is almost impossible for a person to conceptualise all the information and suggest a suitable decomposition to overcome the problems. Such relations need an algorithmic approach of finding if there are problems in a proposed database design and how to eliminate them if they exist. The discussions of these algorithms are beyond the scope of this Unit, but, we will first introduce you to the basic concept that supports the process of Normalisation of large databases. So let us first define the concept of functional dependence in the subsequent section and follow it up with the concepts of normalisation.

3.4 SINGLE-VALUED DEPENDENCIES

A database is a collection of related information and it is therefore inevitable that some items of information in the database would depend on some other items of information. The information is either single-valued or multivalued. The name of a person or his date of birth is single-valued facts; qualifications of a person or subjects that an instructor teaches are multivalued facts. We will deal only with single-valued facts and discuss the concept of functional dependency.

Let us define this concept logically.

Functional Dependency

Consider a relation R that has two attributes A and B. The attribute B of the relation is functionally dependent on the attribute A if and only if for each value of A, no more than one value of B is associated. In other words, the value of attribute A uniquely determines the value of attribute B and if there were several tuples that had the same value of A then all these tuples will have an identical value of attribute B. That is, if **t1** and **t2** are two tuples in the relation R where **t1(A) = t2(A),** then we must have **t1(B) = t2(B).**

Both, A and B need not be single attributes. They could be any subsets of the attributes of a relation R. The FD between the attributes can be written as:

R.A → R.B or simply A → B, if B is functionally dependent on A (or A functionally determines B). Please note that functional dependency does not imply a one-to-one relationship between A and B.

For example, the relation in *Figure 3*, whose dependencies are shown in *Figure 4*, can be written as:

Enrolmentno → Sname
Enrolmentno → Address
Cno → Cname
Cno → Instructor

Instructor→Office

These functional dependencies imply that there can be only one student name for each **Enrolmentno**, only one address for each student and only one subject name for each **Cno**. It is of course possible that several students may have the same name and several students may live at the same address.

If we consider **Cno → Instructor**, the dependency implies that no subject can have more than one instructor (perhaps this is not a very realistic assumption). Functional dependencies therefore place constraints on what information the database may store. In the example above, you may be wondering if the following FDs hold:

Sname → Enrolmentno **(1)**
Cname → Cno **(2)**

Certainly there is nothing in the given instance of the database relation presented that contradicts the functional dependencies as above. However, whether these FDs hold or not would depend on whether the university or college whose database we are considering allows duplicate student names and course names. If it was the enterprise policy to have unique course names than **(2)** holds. If duplicate student names are possible, and one would think there always is the possibility of two students having exactly the name, then (**1)** does not hold.

A simple example of the functional dependency above is when A is a primary key of an entity (e.g., enrolment number: Enrolment no) and B is some single-valued property or attribute of the entity (e.g., student name: Sname). **A → B** then must always hold. (Why?)

Functional dependencies also arise in relationships. Let C be the primary key of an entity and D be the primary key of another entity. Let the two entities have a relationship. If the relationship is one-to-one, we must have both **C → D** and **D → C** . If the relationship is many-to-one (Con many side), we would have **C → D** but not **D → C** . For many-to-many relationships, no functional dependencies hold.

For example, consider the following E-R diagram:

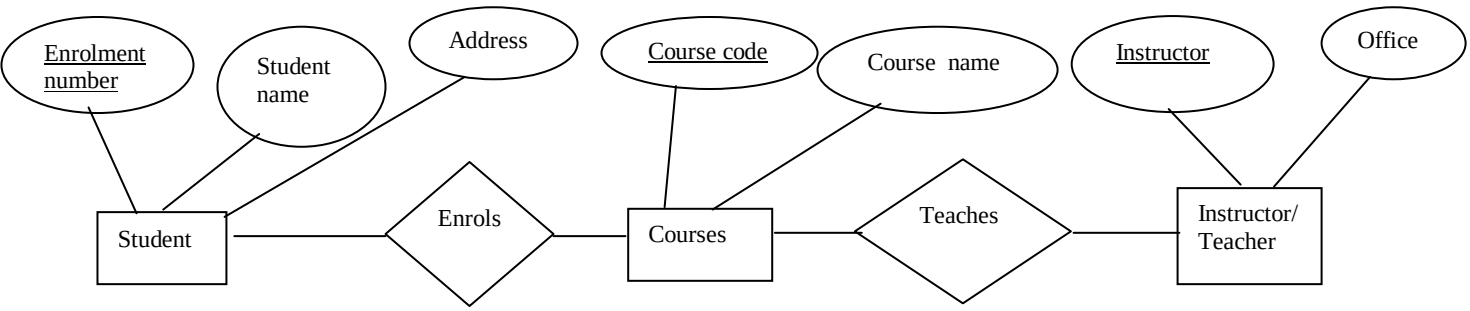


Figure 5: E-R diagram for student course Teacher

In the ER diagram as above, the following FDs exist:

FDs in Entities:

Student entity:

Enrolment number → Student name, Address

Course Entity

Course code → Course name

Instructor/ Teacher Entity:

Instructor → Office

FDs in Relationships:

Enrols Relationship: None as it is many to Many
Teaches Relationship:
Course code → Instructor
Instructor → Course code

The next question is: How do we identify the functional dependence in a database model?

Functional dependencies arise from the nature of the real world that the database models. Often A and B are facts about an entity where A might be some identifier for the entity and B some characteristic. Functional dependencies cannot be automatically determined by studying one or more instances of a database. They can be determined only by a careful study of the real world and a clear understanding of what each attribute means.

There are no thumb rules for determining FDs.

Functional dependence is an important concept and a large body of formal theory has been developed about it.

3.5 SINGLE VALUED NORMALISATION

Codd in the year 1972 presented three normal forms (1NF, 2NF, and 3NF). These were based on functional dependencies among the attributes of a relation. Later Boyce and Codd proposed another normal form called the Boyce-Codd normal form (BCNF). The fourth and fifth normal forms are based on multivalue and join dependencies and were proposed later. In this section we will cover normal forms till BCNF only. Fourth and fifth normal forms are beyond the scope of this course.
For all practical purposes, 3NF or the BCNF are quite adequate since they remove the anomalies discussed for most common situations. It should be clearly understood that there is no obligation to normalise relations to the highest possible level. Performance should be taken into account and sometimes an organisation may take a decision not to normalise, say, beyond third normal form. But, it should be noted that such designs should be careful enough to take care of anomalies that would result because of the decision above.

Intuitively, the second and third normal forms are designed to result in relations such that each relation contains information about only one thing (either an entity or a relationship). A sound E-R model of the database would ensure that all relations either provide facts about an entity or about a relationship resulting in the relations that are obtained being in 2NF or 3NF.

Normalisation results in decomposition of the original relation. It should be noted that decomposition of relation has to be always based on principles, such as functional dependence, that ensure that the original relation may be reconstructed from the decomposed relations if and when necessary. Careless decomposition of a relation can result in loss of information. We will discuss this in detail in the later section.

Let us now define these normal forms in more detail.

3.5.1 The First Normal Form (1NF)

Let us first define 1NF:

Definition: A relation (table) is in 1NF if

1. **There are no duplicate rows or tuples in the relation.**

2. **Each data value stored in the relation is single-valued**
3. **Entries in a column (attribute) are of the same kind (type).**

Please note that in a 1NF relation the order of the tuples (rows) and attributes (columns) does not matter.

The first requirement above means that the relation **must have a key**. The key may be single attribute or composite key. It may even, possibly, contain all the columns. The first normal form defines only the basic structure of the relation and does not resolve the anomalies discussed in section 3.3.

The relation STUDENT (Enrolmentno, Sname, Address, Cno, Cname, Instructor, Office) of *Figure 3* is in 1NF. The primary key of the relation is (Enrolmentno+Cno).

3.5.2 The Second Normal Form (2NF)

The relation of *Figure 3* is in 1NF, yet it suffers from all anomalies as discussed earlier so we need to define the next normal form.

Definition: A relation is in 2NF if it is in 1NF and every non-key attribute is fully dependent on each candidate key of the relation.

Some of the points that should be noted here are:

- A relation having a single attribute key has to be in 2NF.
- In case of composite key, partial dependency on key that is part of the key is not allowed.
- 2NF tries to ensure that information in one relation is about one thing
- Non-key attributes are those that are not part of any candidate key.

Let us now reconsider *Figure 4*, which defines the FDs of the relation to the relation STUDENT (Enrolmentno, Sname, Address, Cno, Cname, Instructor, Office). These FDs can also be written as:

Enrolmentno	→	Sname, Address	(1)
Cno	→	Cname, Instructor	(2)
Instructor	→	Office	(3)

The key attributes of the relation are (Enrolmentno + Cno). Rest of the attributes are non-key attributes. For the 2NF decomposition, we are concerned with the FDs (1) and (2) as above as they relate to partial dependence on the key that is (Enrolmentno + Cno). As these dependencies (also refer to *Figure 4*) shows that relation in not in 2NF and hence suffer from all the three anomalies and redundancy problems as many non-key attributes can be derived from partial key attribute. To convert the relation into 2NF, let us use FDs. As per FD (1) the Enrolment number uniquely determines student name and address, so one relation should be:
STUDENT1 (Enrolmentno, Sname, Address)

Now as per FD (2) we can decompose the relation further, but what about the attribute 'Office'?

We find in FD (2) that Course code (Cno) attribute uniquely determines the name of instructor (refer to FD 2(a)). Also the FD (3) means that name of the instructor uniquely determines office number. This can be written as:

Cno	→	Instructor	(2 (a)) (without Cname)
Instructor	→	Office	(3)
⇒ Cno	→	Office	(This is transitive dependency)

Thus, FD (2) now can be rewritten as:

Cno	→	Cname, Instructor, Office	(2′)

This FD, now gives us the second decomposed relation:

COU_INST (Cno, Cname, Instruction, Office)

Thus, the relation STUDENT has been decomposed into two relations:

STUDENT1 (Enrolmentno, Sname, Address)
COU_INST (Cno, Cname, Instruction, Office)

Is the decomposition into 2NF complete now?

No, how would you join the two relations created above any way? Please note we have super FDs as, because (Enrolmentno + Cno) is the primary key of the relation STUDENT:

Enrolmentno, Cno → ALL ATTRIBUTES

All the attributes except for the key attributes that are Enrolmentno and Cno, however, are covered on the right side of the FDs (1) (2) and (3), thus, making the FD as redundant. But in any case we have to have a relation that joins the two decomposed relations. This relation would cover any attributes of Super FD that have not been covered by the decomposition and the key attributes. Thus, we need to create a joining relation as:

COURSE_STUDENT (Enrolmentno, Cno)

So, the relation STUDENT in 2NF form would be:

STUDENT1 (Enrolmentno, Sname, Address)	2NF(a)
COU_INST (Cno, Cname, Instruction, Office)	2NF(b)
COURSE_STUDENT (Enrolmentno, Cno)	2NF(c)

3.5.3 The Third Normal Form (3NF)

Although, transforming a relation that is not in 2NF into a number of relations that are in 2NF removes many of the anomalies, it does not necessarily remove all anomalies. Thus, further Normalisation is sometimes needed to ensure further removal of anomalies. These anomalies arise because a 2NF relation may have attributes that are not directly related to the candidate keys of the relation.

Definition: A relation is in third normal form, if it is in 2NF and every non-key attribute of the relation is non-transitively dependent on each candidate key of the relation**.**

But what is **non-transitive** dependence?
Let A, B and C be three attributes of a relation R such that **A → B** and **B → C**. From these FDs, we may derive **A → C**. This dependence **A → C** is transitive.

Now, let us reconsider the relation 2NF (b)

COU_INST (Cno, Cname, Instruction, Office)

Assume that Cname is not unique and therefore Cno is the only candidate key. The following functional dependencies exists

Cno	→	**Instructor**	(2 (a))
Instructor	→	**Office**	(3)
Cno	→	Office	(This is transitive dependency)

We had derived Cno → Office from the functional dependencies 2(a) and (3) for decomposition to 2NF. The relation is however not in 3NF since the attribute 'Office' is not directly dependent on attribute 'Cno' but is transitively dependent on it and

should, therefore, be decomposed as it has all the anomalies. The primary difficulty in the relation above is that an instructor might be responsible for several subjects, requiring one tuple for each course. Therefore, his/her office number will be repeated in each tuple. This leads to all the problems such as update, insertion, and deletion anomalies. To overcome these difficulties we need to decompose the relation 2NF(b) into the following two relations:

COURSE (Cno, Cname, Instructor)
INST (Instructor, Office)

Please note these two relations and 2NF (a) and 2NF (c) are already in 3NF. Thus, the relation STUDENT in 3 NF would be:

STUDENT1 (Enrolmentno, Sname, Address)
COURSE (Cno, Cname, Instructor)
INST (Instructor, Office)
COURSE_STUDENT (Enrolmentno, Cno)

The 3NF is usually quite adequate for most relational database designs. There are however some situations where a relation may be in 3 NF, but have the anomalies. For example, consider the relation NEWSTUDENT (Enrolmentno, Sno, Sname, Cno, Cname) having the set of FDs:

Enrolmentno	→	Sname
Sname	→	Enrolmentno
Cno	→	Cname
Cname	→	Cno

The relation is in 3NF. Why? Please refer to the functional diagram for this relation given in *Figure 6*.

Error!

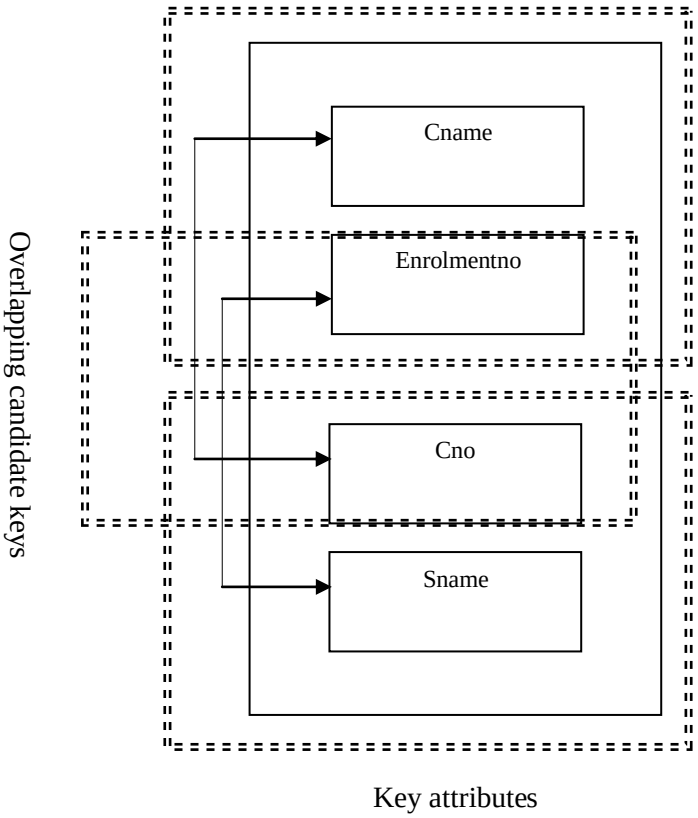


Figure 6: Functional Diagram for NEWSTUDENT relation

All the attributes of this relation are part of candidate keys, but have dependency between the non-overlapping portions of overlapping candidate keys. Thus, the 3NF may not eliminate all the redundancies and inconsistencies. Thus, there is a need of further Normalisation using the BCNF.

3.5.4 Boyce-Codd Normal Form (BCNF)

The relation NEWSTUDENT (Enrolmentno, Sno, Sname, Cno, Cname) has all attributes participating in candidate keys since all the attributes are assumed to be unique. We therefore had the following candidate keys.

(Enrolmentno, Cno)
(Enrolmentno, Cname)
(Sname, Cno)
(Sname, Cname)

Since the relation has no non-key attributes, the relation is in 2NF and also in 3NF. However, the relation suffers from the anomalies (please check it yourself by making the relational instance of the NEWSTUDENT relation).

The difficulty in this relation is being caused by dependence within the candidate keys.

Definition: A relation is in BCNF, if it is in 3NF and if every determinant is a candidate key.

- A determinant is the left side of an FD
- Most relations that are in 3NF are also in BCNF. A 3NF relation is not in BCNF if all the following conditions apply.

(a) The candidate keys in the relation are composite keys.
(b) There is more than one overlapping candidate keys in the relation, and some attributes in the keys are overlapping and some are not overlapping.
(c) There is a FD from the non-overlapping attribute(s) of one candidate key to non-overlapping attribute(s) of other candidate key.

Let us recall the NEWSTUDENT relation:

NEWSTUDENT (Enrolmentno, Sno, Sname, Cno, Cname)

Set of FDs:

Enrolmentno	→	Sname	(1)
Sname	→	Enrolmentno	(2)
Cno	→	Cname	(3)
Cname	→	Cno	(4)

The relation although in 3NF, but is not in BCNF and can be decomposed on any one of the FDs in (1) & (2); and any one of the FDs in (3) & (4) as:

STUD1 (Enrolmentno, Sname)
COUR1 (Cno, Cname)

The third relation that will join the two relation will be:
ST_CO(Enrolmentno, Cno)

Since this is a slightly difficult form, let us give one more example, for BCNF.

Consider for example, the relation:

ENROL(Enrolmentno, Sname, Cno, Cname, Dateenrolled)

Let us assume that the relation has the following candidate keys:

(Enrolmentno, Cno)
(Enrolmentno, Cname)
(Sname, Cno)
(Sname, Cname)

(We have assumed Sname and Cname are unique identifiers).

The relation has the following set of dependencies:

Enrolmentno	→	Sname
Sname	→	Enrolmentno
Cno	→	Cname
Cname	→	Cno
Enrolmentno, Cno	→	Dateenrolled

The relation is in 3NF but not in BCNF because there are dependencies. The relation suffers from all anomalies. Please draw the relational instance and checks these problems. The BCNF decomposition of the relation would be:

STUD1 (Enrolment no, Sname)
COU1 (Cno, Cname)
ENROL1 (Enrolmentno, Cno, Dateenrolled)

We now have a relation that only has information about students, another only about subjects and the third only about relationship enrolls.

Higher Normal Forms:

Are there more normal forms beyond BCNF? Yes, however, these normal forms are not based on the concept of functional dependence. Further normalisation is needed if the relation has Multi-valued, join dependencies, or template dependencies. These topics are beyond the scope of this unit you can refer to further readings for more detail on these. MCS-043 contains more discussion on these topics.

Check Your Progress 2

1) Consider the following relation
LIBRARY (member_id, member_name, book_code, book_name, issue_date, return_date)

The relation stores information about the issue and return of books in a Library to its members. A member can be issued many books. What are the anomalies in the relation above?

……

……

2) What are the functional dependencies in the relation above? Are there any constraints, specially domain constraint, in the relation?

……

……

3) Normalise the relation of problem 1.

……

……

……

……

3.6 DESIRABLE PROPERTIES OF DECOMPOSITION

We have used normalisation to decompose relations in the previous section. But what is decomposition formally? Decomposition is a process of splitting a relation into its projections that will not be disjoint. Please recall the Relational projection operator given in Unit 2, and remember no duplicate tuples are allowed in a projection.

Desirable properties of decomposition are:

- Attribute preservation
- Lossless-join decomposition
- Dependency preservation
- Lack of redundancy

3.6.1 Attribute Preservation

This is a simple and obvious requirement that involves preserving all the attributes that were there in the relation that is being decomposed.

3.6.2 Lossless-Join Decomposition

Let us show an intuitive decomposition of a relation. We need a better basis for deciding decompositions since intuition may not always be correct. We illustrate how a careless decomposition may lead to problems including loss of information.

Consider the following relation
ENROL (stno, cno, date-enrolled, room-no, instructor)

Suppose we decompose the above relation into two relations enrol and enrol2 as follows:

ENROL1 (stno, cno, date-enrolled)

ENROL2 (date-enrolled, room-no, instructor)

There are problems with this decomposition but we do not wish to focus on this aspect at the moment. Let an instance of the relation ENROL be:

St no	cno	Date-enrolled	Room-no	Instructor
1123	MCS-011	20-06-2004	**1**	Navyug
1123	MCS-012	26-09-2004	**2**	Anurag Sharma
1259	MCS-011	26-09-2003	**1**	Preeti Anand
1134	MCS-015	30-10-2005	**5**	Preeti Anand
2223	MCS-016	05-02-2004	**6**	Shashi Bhushan

Figure 7: A sample relation for decomposition

Then on decomposition the relations ENROL1 and ENROL2 would be:

ENROL1

St no	Cno	Date-enrolled
1123	MCS-011	20-06-2004
1123	MCS-012	26-09-2004
1259	MCS-011	26-09-2003
1134	MCS-015	30-10-2005
2223	MCS-016	05-02-2004

ENROL2

Date-enrolled	Room-no	Instructor
20-06-2004	**1**	Navyug
26-09-2004	**2**	Anurag Sharma
26-09-2003	**1**	Preeti Anand
30-10-2005	**5**	Preeti Anand
05-02-2004	**6**	Shashi Bhushan

All the information that was in the relation ENROL appears to be still available in ENROL1 and ENROL2 but this is not so. Suppose, we wanted to retrieve the student numbers of all students taking a course from Preeti Anand, we would need to join ENROL1 and ENROL2. For joining the only common attribute is Date-enrolled. Thus, the resulting relation obtained will not be the same as that of *Figure 7*. (Please do the join and verify the resulting relation).

The join will contain a number of spurious tuples that were not in the original relation. Because of these additional tuples, we have lost the right information about which students take courses from Preeti Anand. (Yes, we have more tuples but less information because we are unable to say with certainty who is taking courses from Preeti Anand). Such decompositions are called **lossy decompositions**. A nonloss or lossless decomposition is that which guarantees that the join will result in exactly the same relation as was decomposed. One might think that there might be other ways of recovering the original relation from the decomposed relations but, sadly, no other operators can recover the original relation if the join does not (why?).

We need to analyse why the decomposition is lossy. The common attribute in the above decompositions was Date-enrolled. The common attribute is the glue that gives us the ability to find the relationships between different relations by joining the relations together. **If the common attribute have been the primary key of at least one of the two decomposed relations, the problem of losing information would not have existed**. The problem arises because several enrolments may take place on the same date.

The dependency based decomposition scheme as discussed in the section 3.5 creates lossless decomposition.

3.6.3 Dependency Preservation

It is clear that the decomposition must be lossless so that we do not lose any information from the relation that is decomposed. Dependency preservation is another important requirement since a **dependency is a constraint** on the database. If all the attributes appearing on the left and the right side of a dependency appear in the same relation, then a dependency is considered to be preserved. Thus, dependency preservation can be checked easily. Dependency preservation is important, because as stated earlier, dependency is a constraint on a relation. Thus, if a constraint is split over more than one relation (dependency is not preserved), the constraint would be difficult to meet. We will not discuss this in more detail in this unit. You may refer to the further readings for more details. However, let us state one basic point:
"A decomposition into 3NF is lossless and dependency preserving whereas a decomposition into BCNF is lossless but may or may not be dependency preserving."

3.6.4 Lack of Redundancy

We have discussed the problems of repetition of information in a database. Such repetition should be avoided as much as possible. Let us state once again that redundancy may lead to inconsistency. On the other hand controlled redundancy

sometimes is important for the recovery in database system. We will provide more details on recovery in Unit 3 of Block 2.

3.7 RULES OF DATA NORMALISATION

Let us now summarise Normalisation with the help of several clean rules. The following are the basic rules for the Normalisation process:

1. **Eliminate Repeating Groups:** Make a separate relation for each set of related attributes, and give each relation a primary key.
2. **Eliminate Redundant Data:** If an attribute depends on only part of a multi-attribute key, remove it to a separate relation.
3. **Eliminate Columns Not Dependent On Key:** If attributes do not contribute to a description of the key, remove them to a separate relation.
4. **Isolate Independent Multiple Relationships:** No relation may contain two or more **1:n** or **n:m** relationships that are not directly related.
5. **Isolate Semantically Related Multiple Relationships:** There may be practical constrains on information that justify separating logically related many-to-many relationships.

Let's explain these steps of Normalisation through an example:

Let us make a list of all the employees in the company. In the original employee list, each employee name is followed by any databases that the member has experience with. Some might know many, and others might not know any.

Emp-ID	Emp-Name	Database-Known	Department	Department-Loc
1	Gurpreet Malhotra	Oracle,	A	N-Delhi
2	Faisal Khan	Access	A	N-Delhi
3	Manisha Kukreja	FoxPro	B	Agra
4	Sameer Singh	DB2, Oracle	C	Mumbai

For the time being we are not considering department and Department-Loc till step 3.

3.7.1 Eliminate Repeating Groups

The query is, "Find out the list of employees, who knows DB2".
For this query we need to perform an awkward scan of the list looking for references to DB2. This is inefficient and an extremely untidy way to retrieve information.

We convert the relation to 1NF. Please note that in the conversion Department and Department-loc fields will be part of Employee relation. The Emp-ID in the Database relation matches the primary key in the employee relation, providing a foreign key for relating the two relations with a join operation. Now we can answer the question by looking in the database relation for "DB2" and getting the list of Employees. Please note that in this design we need not add D-ID (Database ID). Just the name of the database would have been sufficient as the names of databases are unique.

Employee Relation	
Emp-ID	**Emp-Name**
1	Gurpreet Malhotra
2	Faisal Khan
3	Manisha Kukreja
4	Sameer Singh

Database Relation		
D-ID	**Emp-ID**	**Database-name**
1	2	Access
2	4	DB2
3	3	FoxPro
4	1	Oracle
4	4	Oracle

3.7.2 Eliminate Redundant Data

In the "Database Relation" above, the primary key is made up of the Emp-ID and the D-ID. The database-name depends only on the D-ID. The same database-name will appear redundantly every time its associated D-ID appears in the Database Relation. The database relation has redundancy, for example D-ID value 4 is oracle is repeated twice. In addition, it also suffers insertion anomaly that is we cannot enter Sybase in the relation as no employee has that skill.

The deletion anomaly also exists. For example, if we remove employee with Emp-ID 3; no employee with FoxPro knowledge remains and the information that D-ID 3 is the code of FoxPro will be lost.

To avoid these problems, we need **second normal form**. To achieve this, we isolate the attributes depending on both parts of the key from those depending only on the D-ID. This results in two relations: "Database" which gives the name for each D-ID, and "Emp-database" which lists the databases for each employee. The employee relation is already in 2NF as all the EMP-ID determines all other attributes.

Employee Relation	
Emp-ID	**Emp-Name**
1	Gurpreet Malhotra
2	Faisal Khan
3	Manisha Kukreja
4	Sameer Singh

Emp-database Relation	
Emp-ID	**D-ID**
2	1
4	2
3	3
1	4
4	4

Database Relation	
D-ID	**Database**
1	Access
2	DB2
3	FoxPro
4	Oracle

3.7.3 Eliminate Columns Not Dependent On Key

The Employee Relation satisfies -

First normal form - As it contains no repeating groups.
Second normal form - As it doesn't have a multi-attribute key.

The employee relation is in 2NF but not 3NF. So we consider this table only after adding the required attributes.

Employee Relation			
Emp-ID	**Emp-Name**	**Department**	**Department-Loc**
1	Gurpreet Malhotra	A	N-Delhi
2	Faisal Khan	A	N-Delhi
3	Manisha Kukreja	B	Agra
4	Sameer Singh	C	Mumbai

The key is Emp-ID, and the Dept-Name and location describe only about Department, not an Employee. To achieve the third normal form, they must be moved into a separate relation. Since they describe a department, thus the attribute Department becomes the key of the new "Department" relation.

The motivation for this is the same for the second normal form: we want to avoid update, insertion and deletion anomalies.

Employee-List	
Emp-ID	**Emp-Name**
1	Gurpreet Malhotra
2	Faisal Khan
3	Manisha Kukreja
4	Sameer Singh

Department-Relation		
Dept-ID	**Department**	**Department Loc**
1	A	N-Delhi
2	B	Agra
3	C	Mumbai

The rest of the Relation remains the same.

The last two steps: Isolate Independent Multiple Relationships and Isolate Semantically Related Multiple Relationships, converts the relations to higher normal form and are therefore not discussed here. They are not even required for the current example.

Check Your Progress 3

1) What is the need of dependency preservation?

……

……

……

……

……

2) What is a lossless decomposition?

……

……

……

……

3) What are the steps for Normalisation till BCNF?

……

……

……

……

3.8 SUMMARY

This unit converts the details of Database integrity in detail. It covers aspects of keys, entity integrity and referential integrity. The role of integrity constraints is to make data more consistent.

A **functional dependency (FD)** is a many-to-one relationship between two sets of attributes of a given relation. Given a relation R, the FD A → B (where A and B are subsets of the attributes of R) is said to hold in R if and only if, whenever two tuples of R have the same value for A, and they also have the same value for B.

We discussed Single valued Normalisation and the concepts of **first, second, third,** and **Boyce/Codd normal forms.** The purpose of Normalisation is to avoid **redundancy**, and hence to avoid certain **update insertion and detection anomalies.**

We have also discussed the desirable properties of a decomposition of a relation to its normal forms. The decomposition should be attribute preserving, dependency preserving and lossless.

We have also discussed various rules of data Normalisation, which help to normalise the relation cleanly. Those rules are eliminating repeating groups, eliminate redundant data, eliminate columns not dependent on key, isolate independent multiple relationship and isolate semantically related multiple relationships.

3.9 SOLUTIONS/ANSWERS

Check Your Progress 1

1)

Relation	Candidate Keys	Primary key
SUPPLIERS	SNO, SNAME*	SNO
PARTS	PNO, PNAME*	PNO
PROJECTS	PROJ NO, JNAME*	PROJNO
SUP_PAR_PROJ	(SNO+PNO+PROJNO)	SNO+PNO+PROJNO

* Only if the values are assumed to be unique, this may be incorrect for large systems.

2) SNO in SUPPLIERS, PNO in PARTS, PROJNO in PROJECTS and (SNO+PNO+PROjNO) in SUP_PAR_PROJ should not contain NULL value. Also no part of the primary key of SUP_PAR_PROJ) that is SNO or PNO or PROJNO should contain NULL value. Please note SUP_PAR_PROJ relation is violating this constraint in the 12^{th} tuple which is not allowed.

3) Foreign keys exist only in SUP_PAR_PROJ where SNO references SNO of SUPPLIERS; PNO references PNO of PARTS; and PROJNO references PROJNO of PROJECTS. The referential integrity necessitates that all matching foreign key values must exist. The SNO field of SUP_PAR_PROJ in last tuple contains S6 which has no matching SNO in SUPPLIERS (valid values are from S1 to S5). So there is a violation of referential integrity constraint.

4) The proposed referential actions are:

For Delete and update, we must use RESTRICT, otherwise we may lose the information from the SUP_PAR_PROJ relation. Insert does not create a problem, only referential integrity constraints must be met.

Check Your Progress 2

1) The database suffers from all the anomalies; let us demonstrate these with the help of the following relation instance or state of the relation:

Member ID	Member Name	Book Code	Book Title	Issue Date	Return Date
A 101	Abishek	0050	DBMS	15/01/05	25/01/05
R 102	Raman	0125	DS	25/01/05	29/01/05
A 101	Abishek	0060	Multimedia	20/01/05	NULL
R 102	Raman	0050	DBMS	28/01/05	NULL

Is there any data redundancy?

Yes, the information is getting repeated about member names and book details. This may lead to any update anomaly in case of changes made to data value of the book. Also note the library must be having many more books that have not been issued yet. This information cannot be added to the relation as the primary key to the relation is: (member_id + book_code + issue_date). (This would involve the assumption that the same book can be issued to the same student only once in a day).

Thus, we cannot enter the information about book_code and title of that book without entering member_id and issue_date. So we cannot enter a book that has not been issued to a member so far. This is insertion anomaly. Similarly, we cannot enter member_id if a book is not issued to that member. This is also an insertion anomaly.

As far as the deletion anomaly is concerned, suppose Abishek did not collect the Multimedia book, so this record needs to be deleted from the relation (tuple 3). This deletion will also remove the information about the Multimedia book that is its book code and title. This is deletion anomaly for the given instance.

2) The FDs of the relation are:

member_id → member_name (1)

book_code → book_name (2)

book_code, member_id, issue_date → return_date (3)

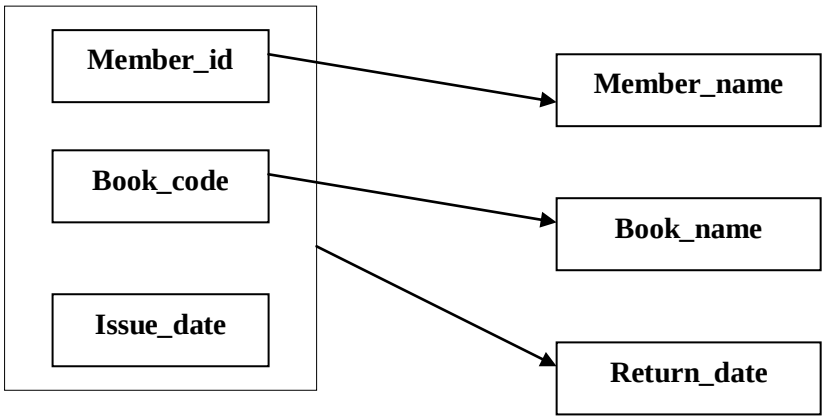
Why is the attribute issue_date on the left hand of the FD above?
Because a student for example Abishek can be issued the book having the book_code 0050 again on a later date, let say in February after Raman has returned it. Also note that return_date may have NULL value, but that is allowed in the FD. FD only necessitates that the value may be NULL or any specific data that is consistent across the instance of the relation.

Some interesting domain and procedural constraints in this database are:

- A book cannot be issued again unless it is returned
- A book can be returned only after the date on which it has been issued.
- A member can be issued a limited/maximum number of books at a time.

You will study in Book 2 Unit 1, how to put some of these constraints into domains using SQL.

3) The table is in 1NF. Let us draw the dependency diagram for the table.



1NF to 2NF:

The member_name and book_name attributes are dependent on part of the key so the decomposition based on specific FDs would be:

MEMBER(member_id, member_name) [Reason: FD (1)]
BOOK (book_code, book_name) [Reason: FD (2)]
ISSUE_RETURN (member_id, book_code, issue_date, return_date)
[Reason: FD (3)]

2NF to 3 NF and BCNF:

All the relations are in 3NF and BCNF also. As there is no dependence among non-key attributes and there is no overlapping candidate key.

Please note that the decomposed relations have no anomalies. Let us map the relation instance here:

MEMBER

Member - ID	Members Name
A 101	Abhishek
R 102	Raman

BOOK

Book - Code	Book – Name
0050	DBMS
0060	Multimedia
0125	OS

ISSUE_RETURN

Member - ID	Book - Code	Issue - Date	Return – Date
A 101	0050	15/01/05	25/01/05
R 102	0125	25/01/05	29/01/05
A 101	0060	20/01/05	NULL
R 102	0050	28/01/05	NULL

i) There is no redundancy, so no update anomaly is possible in the relations above.

ii) The insertion of new book and new member can be done in the BOOK and MEMBER tables respectively without any issue of book to member or vice versa. So no insertion anomaly.

iii) Even on deleting record 3 from ISSUE_RETURN it will not delete the information the book 0060 titled as multimedia as this information is in separate table. So no deletion anomaly.

Check Your Progress 3

1) Dependency preservation within a relation helps in enforcing constraints that are implied by dependency over a single relation. In case you do not preserve dependency then constraints might be enforced on more than one relation that is quite troublesome and time consuming.

2) A lossless decomposition is one in which you can reconstruct the original table without loss of information that means exactly the same tuples are obtained on taking join of the relations obtained on decomposition. Lossless decomposition requires that decomposition should be carried out on the basis of FDs.

3) The steps of Normalisation are:

   1. Remove repeating groups of each of the multi-valued attributes.
   2. Then remove redundant data and its dependence on part key.
   3. Remove columns from the table that are not dependent on the key that is remove transitive dependency.
   4. Check if there are overlapping candidate keys. If yes, check for any duplicate information, and remove columns that cause it.