Topic 6 ▶ Normalisation

LEARNING OUTCOMES

When you have completed this Topic you should be able to:

1. Discuss importance of the normalisation in the database design.
2. Discuss the problems related to data redundancy.
3. Explain the characteristics of functional dependency which describes the relationship between attributes.
4. Apply the functional dependency concept in normalisation.
5. Discuss the characteristics of the three normal forms.
6. Employ normalisation process up to third normal forms in the design of a database.

TABLE OF CONTENTS

▶ INTRODUCTION

In this Topic 6, we introduce the concept of normalisation and explain its importance in the database design. Next, we will present the potential problems in the database design which is also referred to as update anomalies. One of the main goals of normalisation is to produce a set of relations that is free from update anomalies. Then, we go into the key concept that is fundamental to understanding the normalisation process which is functional dependency. Normalisation involves a step by step process or normal forms. This Topic will cover discussion of the normalisation process up to the third normal form.

6.1 THE PURPOSE OF NORMALISATION

What is normalisation?

Normalisation is a technique used when designing a database. Normalisation involves a multi-step process with aim to reduce data redundancy and to help eliminate data anomalies that can result from such redundancy. Normalisation works through a series of stages, described as normal forms: the first three stages are referred to as: first normal form (1NF); second normal form (2NF); and third normal form (3NF).

The concept of normalisation was first developed and documented by E. F. Codd (1972). There are two goals of the normalisation process:

- eliminating redundant data (for example, storing the same data in more than one table) and
- ensuring data dependencies make sense (only storing related data in a table).

Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored. If our database is not normalised it can be inaccurate, slow, inefficient, and it might not produce the

data we expect. Not to mention if we have a normalised database, queries, forms, and reports are much easier to design!

SELF-CHECK 6.1

1. Define normalisation.
2. Identify two purposes of normalisation.

6.2 HOW NORMALISATION SUPPORTS DATABASE DESIGN

Normalisation involves the analysis of functional dependencies between attributes (or data items). It helps us decide which attributes should be grouped together in a relation.

Why Normalisation?

Normalisation is about designing a "good" database i.e. a set of related tables with a minimum of redundant data and no update, delete or insert anomalies. Normalisation is a "bottom up" approach to database design. The designer interviews users and collects documents - reports etc. The data on a report can be listed and then normalised to produce the required tables and attributes.

Normalisation is also used to repair a "bad" database design, i.e. given a set of tables that exhibit update, delete & insert anomalies the normalisation process can be used to change this set of tables to a set that does not have problems.

SELF-CHECK 6.2

1. Briefly explain how normalisation supports database design.
2. Is normalisation a "bottom-up" or "top-down" approach to database design? Briefly explain.

6.3 DATA REDUNDANCY AND UPDATE ANOMALIES

Data redundancy refers to an issue related to the duplication of unnecessary data within a database. The redundant data utilises lot of space unnecessarily and also may creates problem when updating the database, also called update anomalies, which may leads to data inconsistency, and inaccuracy.

As mentioned earlier, the main aim of database design is to eliminate data redundancy. To eliminate data redundancy, you must take special care to organise the data in your database. Normalisation is a method of organising your data as it helps you to decide which attributes should be grouped together in a relation.

To illustrate the problem associated with data redundancy that causes update anomalies, lets compare the Product and Supplier relations shown in Figure 6.1 with the alternative format that combine these relation into a single relation called Product-Supplier as shown in Figure 6.2. For the Supplier relation, supplier number (*SuppNo*) is the primary key, and for Product relation, product number (*ProductNo*) is the primary key. For the Product-Supplier relation, *ProductNo* is chosen as the primary key.

Supplier

SuppNo	SName	TelNo	ContactPerson
S8843	ABX Technics	56334532	Teresa Ng
S9884	SoftSystem	55212233	Fatimah
S9898	ID Computers	77617709	Larry Wong
S9990	ITN Suppliers	56345505	Tang Lee Huat

Product

ProductNo	Name	UnitPrice	SuppNo
P2344	17 inch Monitor	200	S8843
P2346	19 inch Monitor	250	S8843
P4590	Laser Printer	650	S9884
P5443	Color Laser Printer	750	S9898
P6677	Color Scanner	350	S9990
P7700	3 in 1 Printer	400	S9990

Figure 6.1: Supplier and Product Relation

Product-Supplier

ProductNo	Name	UnitPrice	SuppNo	SName	TelNo	ContactPerson
P2344	17 inch Monitor	200	S8843	ABX Technics	56334532	Teresa Ng
P2346	19 inch Monitor	250	S8843	ABX Technics	56334532	Teresa Ng
P4590	Laser Printer	650	S9884	SoftSystem	55212233	Fatimah
P5443	Color Laser Printer	750	S9898	ID Computers	77617709	Larry Wong
P6677	Color Scanner	350	S9990	ITN Suppliers	56345505	Tang Lee Huat
P7700	3 in 1 Printer	400	S9990	ITN Suppliers	56345505	Tang Lee Huat

Figure 6.2: Product-Supplier Relation

You should notice that in the Product-Supplier relation the details of the supplier are included for every each product. These supplier details (*SName, Telno* and *contactPerson* attributes) are unnecessarily repeated for every product that is supplied by the same supplier, and this leads to data redundancy. For instance, the product numbers '*P2344'* and '*P2346'* have the same supplier, thus the same supplier details for both products are repeated. These supplier details attributes are also considered as repeating group. On the other hand, in the Product relation, only the supplier number is repeated for the purpose of linking each product to a supplier, and in the Supplier relation the details of each supplier appears only once.

A relation with data redundancy as shown in the Figure 6.2, may result in problem called update anomalies, comprises of insertion, deletion and modification anomalies. In the following section, we illustrate each of these anomalies using the Product-Supplier relation.

6.3.1 Insertion Anomalies

Insert anomalies exist when adding a new record will cause unnecessarily data redundancy or when there is unnecessarily constraint places on a task of adding new record.

There are two types of insertion anomalies:

- Product-Supplier Relation
  Since the information about product and supplier are combined together in a single relation, to add a new supplier is not possible without entering values into attributes for products such as product number. This is because the product number is the primary key of the relation, and based on entity integrity rule, a null value is not allowed for a primary key. In other words, we cannot add new supplier unless we assigned a product to that new supplier. This kind of problem is an example of insert anomaly.

- Type of Insertion Anomaly
  When we want to insert a new product that is supplied by existing supplier, we need to ensure that the details of the supplier (repeating group) are accurately entered and consistent with existing stored values. For instance, to insert a new product supplied by '*S9990'*, we must ensure that details of supplier '*S9990'* are accurately entered and consistent with values for supplier '*S9990'* in other tuples of the Product-Supplier relation. In a properly normalised database, such as shown in Figure 3.1, the insertion anomaly can be avoided as we need only to enter the supplier number for each product in the product relation and the details of the supplier are entered only once in the Supplier relation.

6.3.2 Deletion Anomalies

A deletion anomaly exists when deleting a record would remove a record not intended for deletion. In this case when we want to delete a product from the Product-supplier relation, the details about the supplier would also be removed from the database. There is a possibility that we are deleting the only tuple that we have for a particular supplier. For instance, if we want to delete a product *'P5443'*, the details on supplier '*S9898'* would also be removed from the database. As a result we lost the whole information of this supplier because the supplier '*S9898'* only appears in the tuple that we removed. In a properly normalised database, this deletion anomaly can be avoided as the information about supplier and product is stored in separate relations and they are link together using the supplier number. Therefore, when we delete a product number '*P5443'* from Product relation, the details about the supplier '*S9898'* from the Supplier relation are not affected.

6.3.3 Modification Anomalies

An update anomaly exists when a modifying a specific value necessitates the same modification in other records or tables.

Redundant information not only wastes storage but makes updates more difficult since, for example, changing the name of contact person for supplier '*S9990'* would require that all tuples containing supplier *S9990'* need to be updated. If for some reason, all tuples are not updated, we might have a database that has two different names of contact person for supplier *S9990'*. This difficulty is called the *modification anomaly*. Since our example is only dealing with small relation, it does not seem to be a big problem. However, its effect would be very significant when we are dealing with a very large database.

Similar to the insertion and deletion anomaly, we may avoid the modification anomaly by having a properly normalised database. In our examples, these update anomalies arise primarily because the Product-subject relation has information about both product and supplier. One solution to deal with this problem is to decompose the relation into two smaller relations, the Product and Supplier.

Before we discuss the details of the normalisation process, let's look at the functional dependency concept, which is an important concept to the normalisation process.

SELF-CHECK 6.3

1. Briefly explain data redundancy.
2. Give one example how data redundancy can cause update anomalies.
3. Briefly differentiate between insertion anomalies, deletion anomalies and modification anomalies.

6.4 FUNCTIONAL DEPENDENCIES

Functional dependency is an important concept underlying the normalisation process. Functional Dependency describes the relationship between attributes (columns) in a relation.

In this section we explain the characteristics and the type of functional dependency that are important for normalisation process. For our discussion on this concept, we will refer to the CustomerOrdering relational schema as shown in Figure 3.3(a) and the details of the relation is in Figure 6.3(b).

CustomerOrdering (CustNo, CustName, TelNo, OrderNo, OrderDate, ProductNo, ProdName, UnitPrice, QtyOrdered)

Figure 6.3(a): CustomerOrdering Relation

CustNo	CustName	TelNo	Order No	Order Date	Product No	Prod Name	Unit Price	Qty Ordered
C3340	Bakar Nordin	017-6891122	6234	16-Apr-2007	P2346	19 inch Monitor	250	4
C1010	Fong Kim Lee	012-5677118	1120	23-Jan-2007	P4590	Laser Printer	650	2
C1010	Fong Kim Lee	012-5677118	1120	23-Jan-2007	P6677	Color Scanner	350	2
C1010	Fong Kim Lee	012-5677118	1120	23-Jan-2007	P2344	17 inch Monitor	200	3
C2388	Jaspal Singh	013-3717071	4399	19-Feb-2007	P2344	17 inch Monitor	200	2
C2388	Jaspal Singh	013-3717071	4399	19-Feb-2007	P5443	Color laser printer	750	5
C4455	Daud Osman	017-7781256	9503	02-May-2007	P2344	17 inch Monitor	200	10

Figure 6.3(b): CustomerProduct Relation

6.4.1 Characteristics of Functional Dependencies

Before we look into the normalisation process, let us first understand the concept and characteristics of functional dependence, which is crucial in understanding the normalisation process. As mentioned earlier, functional dependency describes the relationship between attributes in a relation, in which one attribute or group of attributes determines the value of another.

For a simple illustration of this concept, lets for example, we have a relation with attributes A and B. B is functionally dependent on A, if each value of A is associated with exactly one value of B. This dependency between A and B is written as 'A→B'.

We may think of how to determine functional dependency like this: Given a value for attribute A, can we determine the single value for B? If B relies on A, then A is said to functionally determine B. The functional dependency between attribute A and B is represented diagrammatically in Figure 6.4.

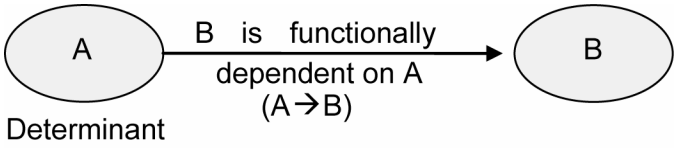

Figure 6.4: Functional Dependency between A and B

Attribute or group of attributes on the left hand side of the arrow of a functional dependency is referred to as **determinant**. In our example above, A is the determinant. Thus we may say 'A functionally determines B'.

Now let's look at the CustomerOrdering relation as shown in the Figure 6.3(a), to find the functional dependencies. First, we consider the attributes *CustNo* and *CustName.* It is true that for a specific *CustNo*, it can only be associated with one value of *custName*. In other words, the relationship between *CustNo* and *CustName* is 1:1 (for each customer number, there is only one name). Thus we can say that *CustNo* determines *CustName* or *CustName* is functionally dependent on *CustNo*. This dependency can be written as *CustNo* → *CustName.*

Let's try another example; the relationship between *CustNo* and *OrderNo.* Based on the CustomerOrdering relation, a customer may make more than one order. Thus, a *CustNo* may be associated with more than one *OrderNo*. In other words, the relationship between *CustNo* and *OrderNo* is 1:M, as illustrated in Figure 6.5(a). In this case, we can say that *OrderNo* is not functionally dependent on *CustNo*.

Now let's examine the opposite direction of the relationship. Does *CustNo* functionally dependent on *OrderNo*? Does a specific *OrderNo* can only be associated with only one value of *CustNo*. In this case, we can say that each *OrderNo* is associated with only one *CustNo*, as illustrated by Figure 6.5(b). Thus, *OrderNo* determines *CustNo*, or *CustNo* is functionally dependent on *OrderNo*, which can be written as *OrderNo* → *CustNo*.

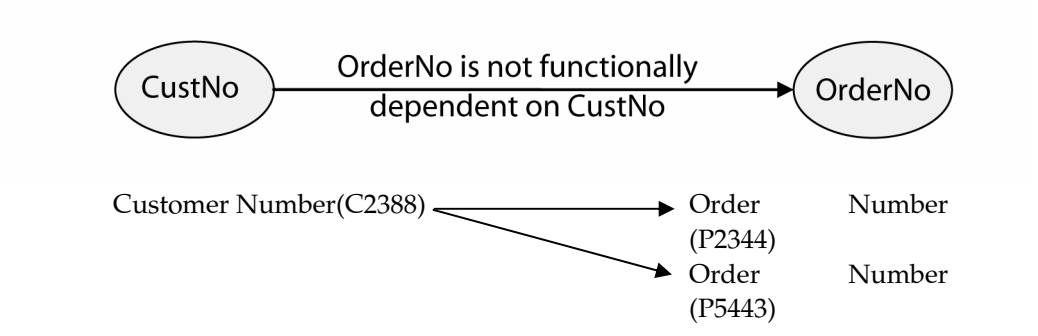

Figure 6.5(a): OrderNo is not functionally dependent on CustNo

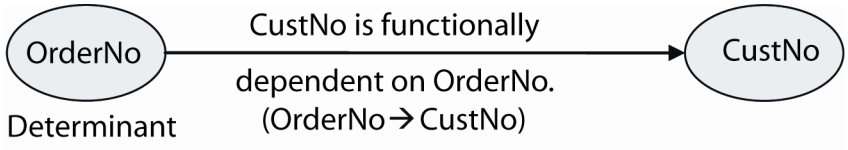

Figure 6.5(b): *CustNo* functionally dependent on *OrderNo*

Additional characteristics of functional dependency that are important for normalisation process are listed below.

- **Full Functional dependency:** Indicates that if A and B are attributes (columns) of a relation, B is fully functionally dependent on A if B is functionally dependent on A but not on any proper subset of A. E.g. *OrderNO → CustNo*

- **Partial Functional Dependency:** Indicates that if A and B are attributes of a relation, B is partially dependent on A if there is some attribute that can be removed from A and yet the dependency still holds.

  Say for example the following functional dependency that exists in the ConsumerOrdering relation: E.g. (*OrderNo, ProductNo) → CustNo. CustNo* is functionally dependent on a subset of A(*OrderNo, ProductNo*), namely *OrderNO*.

- **Transitive Functional Dependency:** A condition where A, B and C are attributes of a relation such that if A is functionally dependent on B and B is functionally dependent on C then C is transitively dependent on A via B.

Say for example, consider the following functional dependencies that exists in the ConsumerOrdering relation:

OrderNo→CustNo,
OrderNo → CustName
CustNo → CustName

So, *OrderNo* attribute funcitionally determines *CustName* via *CustNo* attribute.

6.4.2 Identifying Functional Dependencies

Identifying functional dependency can be difficult and confusing if we do not understand the meaning of each attributes and the relationship between the attributes. This information should be gathered first from users or owner of the system to be build, before the functional dependency can be identified. Examining the user's requirement specification and business model and rules of the enterprise will provide a good source of this information.

Now let's list down all the possible functional dependencies for the CustomerOrdering relation. We will get a list of functional dependencies as listed in Figure 6.6 below:

CustNo → CustName, TelNo
OrderNo → CustNo,CustName, TelNo, OrderDate
ProductNo → ProdName, UnitPrice
OrderNo,ProductNo → QtyOrdered, CustNo, CustName, TelNo, OrderDate, ProductName, UnitPrice

Figure 6.6: List of Functional Dependency

We may write the functional dependencies by grouping them together based on their determinants as given above, or we may list each of them separately (E.g *CustNo →custName, CustNo → TelNo*). There are five determinants in the CustomerOrdering relation: *CustNo, OrderNo, ProductNo, and (OrderNo, ProductNo)*. We have to ensure that for each functional dependency, the left hand side determinant is associated to only a single value of the right hand side attribute/s.

6.4.3 Identifying the Primary Key for a Relation using Functional Dependencies

In our previous discussion, we have identified a list of functional dependencies for the CustomerOrdering relation by analysing the relationship between attributes. Besides identifying the determinants, functional dependency can also assist us in specifying the integrity constraint, and thus help to identify the primary key for a relation. Before we can select a primary key, we need to identify the possible candidate keys.

In order to find the candidate key(s), we must identify the attribute (or group of attributes) that uniquely identifies each tuple in a relation. Therefore, to identify the possible choices of candidate keys, we should examine the determinants for each functional dependency. Then we select one of them (if more than one) as the primary key. All attributes that are not the primary key attribute are referred to as non-key attributes. These non-key attributes must be functionally dependent on the key.

Now let us identify the candidate keys for relation CustomerOrdering. We have identified the functional dependencies for this relation as given in Figure 6.6. The determinants for these functional dependencies are: *CustNo, OrderNo, ProductNo,* and *(OrderNo,ProductNo)*. From this list of determinants, the

(OrderNo, ProductNo) is the only determinant that uniquely identifies each tuple in the relation. It is also true that all attributes (besides the *OrderNo* and *ProductNo*) are functionally dependent on the determinants with combination of attributes *OrderNo* and *Product (OrderNo,ProductNo)*. Thus it is the candidate key and the primary key for CustomerOrdering relation.

In this section we have shown the importance of the functional dependency in assisting us identifying the primary key for a given relation. Understanding of this concept is the fundamental of the normalisation process which to be discussed next.

6.5 THE PROCESS OF NORMALISATION

Database Normalisation is the process of organising and decomposing an inefficient structured relation into smaller, and more efficient structured relations. In other words, the process of normalisation involves determining what data should be stored in each relation with aims to minimised data redundancy and update anomalies. It makes use of functional dependencies that exist in a relation and the primary key or candidate keys in analysing the relations.

The normalisation process involves a series of steps and each step is called a normal form. Three normal forms were initially proposed called First normal Form (1NF), Second normal Form (2NF), and Third normal Form (3NF). Subsequently R.Boyce and E.F.Codd introduced a stronger definition of 3NF called Boyce-Codd Normal Form (BCNF). With the exception of 1NF, all these normal forms are based on functional dependencies among the attributes of a relation. Higher normal forms that go beyond BCNF were introduced later such as Fourth Normal Form (4NF) and Fifth Normal Form (5NF). However these later normal forms deal with situations that are very rare. In this Topic, we will only cover the first three normal forms. Figure 3.7 illustrate the process of normalisation up to third normal form.

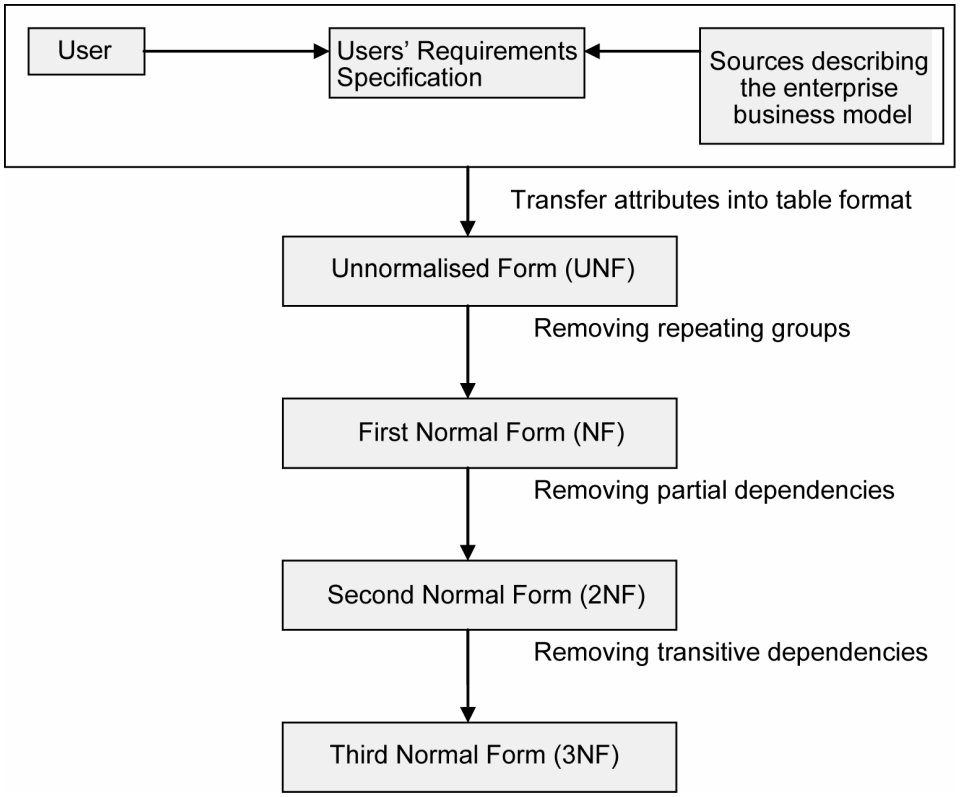

Figure 6.7: Diagrammatic illustration of the Normalisation Process (adopted from Conolly and Begg (2005).

As illustrated in the Figure 6.7, all information gathered about attributes is transferred into table format. This table is described as being in Unnormalised Form (UNF). From here, we need to go through a step by step test of each normal form until it produce a set of relations that fulfill requirements for the third normal form. The definition of each of the unnormalised form (UNF) and first, second and third normal forms can be found in Figure 6.8.

Unnormalised Form (UNF)	A table that contains one or more repeating groups
First Normal Form (1NF)	A relation in which the intersection of each row and coloumn contains one and only one value (atomic value)
Second Normal Form (2NF)	A relation that is in First Normal Form and every non-primary key attributes is fully functionally dependent on the primary key
Third Normal Form (3NF)	A relation that is in First and Second Normal Form and in which no non-primary key attributes is transitively dependent on the primary key

Figure 6.8: Definition for Normal Forms

The detail of the process will be discussed in the following section. Let's assume that we have transferred all the required attributes from the Users specification requirement into the table format and referred it as CustomerOrdering table as shown in Figure 6.9. We are going to use the CustomerOrdering table to illustrate the normalisation process.

Cust No	Cust Name	TelNo	Order No	Order Date	Product No	Prod Name	Unit Price	Qty Ordered
C3340	Bakar Nordin	017-6891122	6234	16-Apr-2007	P2346	19 inch Monitor	250	4
C1010	Fong Kim Lee	012-5677118	1120	23-Jan-2007	P4590	Laser Printer	650	2
					P6677	Color Scanner	350	2
					P2344	17 inch Monitor	200	3
C2388	Jaspal Singh	013-3717071	4399	19-Feb-2007	P2344	17 inch Monitor	200	2
			4399	19-Feb-2007	P5443	Color laser printer	750	5
C4455	Daud Osman	017-7781256	9503	02-May-2007	P2344	17 inch Monitor	200	10

Figure 6.9: Unnormalised CustomerOrdering Relation

6.5.1 First Normal Form (1NF)

First, let's check the CustomerOrdering table and identify whether it is considered as unnormalised or already in the 1NF. Based on the definition given in Figure 6.8, a table is unnormalised if it contains one or more repeating groups. In other words, the table contains a multi-valued attributes, or an attribute or a group of attributes that have more than one value for an instance of an entity.
In order for us to transform the unnormalised table to normalised table, some steps need to be performed, which are:

- Nominate an attribute or group of attributes to act as the key for the unnormalised table.
- Identify the repeating groups(s) in the unnormalised table which repeats for the key attribute(s).

If the table contains repeating groups or multi-valued attributes, then we need to remove these repeating groups. This can be done using any of these two approaches:

1. By entering appropriate data into the empty columns of rows containing the repeating data (fill in the blanks by duplicating the non-repeating data, where required). Or
2. By placing the repeating data along with a copy of the original key attribute(s) into a separate relation.

Then, after performing one of the above approach, we need to check whether the relation is in the First normal form (1NF). We have to follow the following rules:

- Identify the key attribute
- Identify the repeating groups
- Place the repeating groups into a separate relation along with a copy of its determinants.

The process above must repeat to all the new relations created for the repeating attributes to ensure that all relations are in 1NF.

For our example, let's use the first approach by entering appropriate value to each cell of the table. Then we will select a primary key for the relation and check for repeating groups. If there is repeating group then, we have to remove the repeating group to a new relation.

First step is to check whether the table is an unnormalised or is already in the INF. We will use CustomerOrdering table to illustrate the normalisation process. First we select a primary key for the table, which is *CustNo*. Then we need to find for repeating groups or a multi-valued attributes. We can see that *ProductNo, ProductName, UnitPrice* and *QtyOrdered* have more than one value for *CustNo* = 'C1010' and 'C2388'. So these attributes are repeating groups and thus the table is unnormalised.

As illustrated in Figure 6.7, our next step is to transform this unnormalised table into 1NF. First we need to make the table into a normalised relation. Let's apply the first approach in which we need to fill up all the empty cells with a relevant value as shown in Figure 6.10. Each cell in the table now has an atomic value.

CustNo	CustName	TelNo	Order No	Order Date	Product No	Prod Name	Unit Price	Qty Ordered
C3340	Bakar Nordin	017-6891122	6234	16-Apr-2007	P2346	19 inch Monitor	250	4
C1010	Fong Kim Lee	012-5677118	1120	23-Jan-2007	P4590	Laser Printer	650	2
C1010	**Fong Kim Lee**	**012-5677118**	**1120**	23-Jan-2007	P6677	Color Scanner	350	2
C1010	**Fong Kim Lee**	**012-5677118**	**1120**	23-Jan-2007	P2344	17 inch Monitor	200	3
C2388	Jaspal Singh	013-3717071	4399	19-Feb-2007	P2344	17 inch Monitor	200	2
C2388	**Jaspal Singh**	**013-3717071**	**4399**	19-Feb-2007	P5443	Color laser printer	750	5
C4455	Daud Osman	017-7781256	9503	02-May-2007	P2344	17 inch Monitor	200	10

Figure 6.10: A Normalised Table

The Next step is to check if the table we just created is in 1NF. Firstly, we need to identify the primary key for this table and then check for repeating groups. The best choice would be to look at the list of functional dependency that you have identified. From the functional dependency list, we can say that the combination of *OrderNo* and *productNo (OrderNo, ProductNo)* functionally determines all the non-key attributes in the table. This means that the value of each (*OrderNo, ProductNo*) is associated with only a single value of all other attributes in the table and (*OrderNo, ProductNo*) also uniquely identifies each of the tuple in the relation. Thus, we can conclude that (*OrderNo, ProductNo*) is the best choice as the primary key, since the relation will not have any repeating group. Therefore this relation is in 1NF.

CustNo	CustName	TelNo	**Order No**	Order Date	**Product No**	Prod Name	Unit Price	Qty Ordered
C3340	Bakar Nordin	017-6891122	6234	16-Apr-2007	P2346	19 inch Monitor	250	4
C1010	Fong Kim Lee	012-5677118	1120	23-Jan-2007	P4590	Laser Printer	650	2
C1010	Fong Kim Lee	012-5677118	1120	23-Jan-2007	P6677	Color Scanner	350	2
C1010	Fong Kim Lee	012-5677118	1120	23-Jan-2007	P2344	17 inch Monitor	200	3
C2388	Jaspal Singh	013-3717071	4399	19-Feb-2007	P2344	17 inch Monitor	200	2
C2388	Jaspal Singh	013-3717071	4399	19-Feb-2007	P5443	Color laser printer	750	5
C4455	Daud Osman	017-7781256	9503	02-May-2007	P2344	17 inch Monitor	200	10

Figure 6.11: First Normal Form CustomerOrdering Relation

6.5.2 Second Normal Form (2NF)

For relations to be in 2NF they must first be in 1NF. They must also have no partial dependencies. A partial dependency occurs when the primary key is made up of more than one attribute (i.e. it is a composite primary key) and there exists an attribute (which is a non-primary key attribute) that is fully functionally dependent on only part of the primary key.

These partial dependencies can be removed by removing all of the partially dependent attributes into another relation along with a copy of the determinant attribute (which is part of the primary key in the original relation)

Let's now transform the table in Figure 6.11 to 2NF. The first step is to examine whether the relation has partial dependency. Since the primary key choosen for the relation is a combination of 2 attributes, therefore we should check for partial dependency. From the list of functional dependencies, attributes *ProdName* and *UnitPrice* is also full functionally dependent on part of the primary key which is the *ProductNo*, and the *CustNo, custName, TelNo and OrderDate* are full functionally dependent on part of the primary key which is the OrderNo. Thus this relation is not in 2NF and we need to remove these partial dependent attributes into a new relation along with a copy of their determinants. Therefore, we have to remove *ProductName* and *UnitPrice* into a new relation, along with its determinant which is *ProductNo*. We also need to remove *CustNo, CustName, TelNo and OrderDat*e into another new relation along with the determinant *OrderNo*. After performing this process, 1NF CustomerOrdering relation now breaks down into three relations which can be named as Product, Order and OrderProduct, as listed in figure 6.12.

ProductNo	Name	UnitPrice
P2344	17 inch Monitor	200
P2346	19 inch Monitor	250
P4590	Laser Printer	650
P5443	Color Laser Printer	750
P6677	Color Scanner	350

OrderNo	OrderDate	CustNo	CustName	TelNo
1120	23-Jan-2007	C1010	Fong Kim Lee	012-5677118
4399	19-Feb-2007	C2388	Jaspal Singh	013-3717071
6234	16-Apr-2007	C3340	Bakar Nordin	017-6891122
9503	02-May-2007	C4455	Daud Osman	017-7781256

Order No	Product No	Qty Ordered
6234	P2346	4
1120	P4590	2
1120	P6677	2
1120	P2344	3
4399	P2344	2
4399	P5443	5
9503	P2344	10

Figure 6.12: 2NF Relations Derived from CustomerProduct Relation.

Since we make changes to the original relation and have created two new relations, we need to check and ensure that each of these relations is in 2NF. Based on the definition of 2NF, these relations must first check for 1NF test for repeating groups, then check for partial dependency. All these relations are in 1NF as none of them has repeating group. For relations Order and Product, we may skip the partial dependency test as the primary key only has one attribute. Thus both of the relations are already in 2NF. For the OrderProduct relation, there is only one non-key attribute which is *QtyOrdered*, and this attribute is fully functionally dependent on (*OrderNo, ProductNo*). Thus this relation is also in 2NF.

6.5.3 Third Normal Form (3NF)

Getting a relation to 3NF involves removing any transitive dependencies. Therefore, a relation in 3NF must be in 1NF and 2NF and it must have no non-primary key attributes which are transitively dependent upon the primary key. In other words, we must check for functional dependency between two non-key

attributes. Thus, we may conclude that if 2NF relations only have one non-key attribute then the relation is also in 3NF.

If there is a transitive dependency, we must remove the transitive dependency attribute/s or attribute/s with a non-key determinant, to a new relation along with a copy of its determinants.

Now, let's look at all the three 2NF relations as shown in Figure 6.12. Since we are looking for a functional dependency between two non-key attributes, we can say that the relation OrderProduct is already in 3NF. This is because this relation only has one non-key attribute which is the *QtyOrdered*. We need to check for relation Product and Order, as both of these relations have more than one non-key attributes.

Let's check the Product relation. There is no transitive dependency in this relation. Thus we can say that this relation is also in 3NF. Next, we check the Order relation. Based on our functional dependency list, we can see that *CustNo* functionally determines *CustName* and *TelNo*. Thus, *CustName* and *TelNo* are transitive attributes and need to be removed from the Order relation into a new relation along with a copy of the determinant. From completing this process, we derive one additional relation named as Customer relation. For the newly created relation, we need to restart the process to check for 1NF. The primary key for this new relation is normally the determinant of the transitive attribute/s, which is *CustNo*. The relation has no repeating group, thus it is in 1NF. It is also in 2NF since its primary key consists of only one attribute. It also has no transitive dependency, and thus, the Customer relation is already in 3NF.

Now let's check the other three relations. All of them have no transitive dependency. Therefore we conclude that these relations are in 3NF, as shown in Figure 6.13.

Customer

CustNo	CustName	TelNo
C1010	Fong Kim Lee	012-5677118
C2388	Jaspal Singh	013-3717071
C3340	Bakar Nordin	017-6891122
C4455	Daud Osman	017-7781256

Product

ProductNo	Name	UnitPrice
P2344	17 inch Monitor	200
P2346	19 inch Monitor	250
P4590	Laser Printer	650
P5443	Color Laser Printer	750
P6677	Color Scanner	350

Order

OrderNo	OrderDate	CustNo
1120	23-Jan-2007	C1010
4399	19-Feb-2007	C2388
6234	16-Apr-2007	C3340
9503	02-May-2007	C4455

Order Product

OrderNo	ProductNo	QtyOrdered
Null	Null	null
6234	P2346	4
1120	P4590	2
1120	P6677	2
1120	P2344	3
4399	P2344	2
4399	P5443	5
9503	P2344	10

Figure 6.13: 3NF Relations Derived from the CustomerProduct Relation

SUMMARY

- Normalisation is a process of organising your data and breaking it into smaller relations that are easier to manage. The primary reason we normalise a database is to prevent redundant data such that can eliminate update anomalies.

- Data redundancy refers to an issue related to the duplication of unnecessary data within a database. The redundant data utilises lot of space unnecessarily and also may create problems when updating the database, also called update anomalies, which may leads to data inconsistency, and inaccuracy.

- One of the most important concepts underlying the normalisation process is Functional dependency. Functional Dependency describes the relationship between attributes (columns) in a relation.

- Normalisation works through a series of stages, described as normal forms: the first three stages are referred to as: first normal form (1NF); second normal form (2NF); and third normal form (3NF).

- The First Normal Form (1NF) eliminates duplicate attributes from the same relation, creates separate relation for each group of related data, and identifies each tuple with a unique attribute or set of attributes (the primary keys).

- The Second Normal Form (2NF) will remove subsets of data that apply to multiple rows of a table, place them in separate tables, and create relationships between these new relations and the original relation by copying the determinants of the partial dependency attributes to the new relations.

- The Third Normal Form (3NF) will remove columns that are not dependent upon the primary key which is the functional dependency between the two non-key attributes.

DISCUSSION

1. Refer to the following figure :

XYZ COLLEGE
CLASS LIST
SPRING SEMESTER 2007

COURSE CODE: IT123
COURSE TITLE: INTRODUCTION TO DATABASE
LECTURER'S NAME: MR ALEX LIM
LECTURER'S LOCATION: A 203

STUDENT ID	NAME	MAJOR	GRADE
200701	SAM	COMP SC	A
200702	LINDA	INFO TECH	B
200703	ANNE	COMP SC	B
200704	BOB	COMP SC	A

2. By referring to the above figure, convert this user view to a set of 3NF relations. Assuming the following:
   (i) a lecturer has a unique location
   (ii) a student has a unique major
   (iii) a course has a unique title

REFERENCES

Connolly, M. & Begg, C. (2005). *Database systems – A practical approach to design, implementation and management*. (4th ed.). Harlow, Essex, England: Addison-Wesley (Pearson Education Limited).

Rob, P. & Coronel, C. (2004). *Database Systems: Design, Implementation, & Management*. Boston: Thomson Course Technology.