1. Normalization and Functional dependency

Formal Definitions of the Normal Forms

1 Normal Form (1NF)

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

1. There are no duplicated rows in the table.
2. Each cell is single-valued (i.e., there are no repeating groups or arrays).
3. Entries in a column (attribute, field) are of the same kind.

Note: The order of the rows is immaterial; the order of the columns is immaterial.
Note: The requirement that there be no duplicated rows in the table means that the table has a key (although the key might be made up of more than one column—even, possibly, of all the columns).

2nd Normal Form (2NF)

Def: A table is in 2NF if it is in 1NF and if all non-key attributes are dependent on all of the key

(Functional dependency: a particular relationship between two attributes. For any relation R, attribute B is functionally dependent on attribute A if, for every valid instance of A, that value of B. The functional dependence of B on A is represented as A B.)

Solution:

- Create separate tables for sets of values that apply to multiple records.
- Relate these tables with a foreign key.
- Records should not depend on anything other than a table's primary key (a compound key, if necessary).

Note: Since a partial dependency occurs when a non-key attribute is dependent on only a part of the (composite) key, the definition of 2NF is sometimes phrased as, "A table is in 2NF if it is in 1NF and if it has no partial dependencies."

3rd Normal Form (3NF)

Def: A table is in 3NF if it is in 2NF and if it has no transitive dependencies.

Solution:

- Eliminate fields that do not depend on the key.
- Values in a record that are not part of that record's key do not belong in the table. In general, any time the contents of a group of fields may apply to more than a single record in the table, consider placing those fields in a separate table.

For example, in an Employee Recruitment table, a candidate's university name and address may be included. But you need a complete list of universities for group mailings. If university information is stored

in the Candidates table, there is no way to list universities with no current candidates. Create a separate Universities table and link it to the Candidates table with a university code key.

Example

First Normal Form (1NF)

Consider the example of the EMPLOYEE relation – now let us say that employees complete training Courses. This information might be stored in a relation named EMPLOYEE, that looks like the table below.

EMPLOYEE

Emp_ID	**Name**	**Department**	**Salary**	*Courses*
100	Marta Negash	Information Technology	100000	**Course** / **Date_Completed**: C++ / 30/11/2007; Report Writing / 15/10/2007
101	Alemu Ayalew	Administration	100000	**Course** / **Date_Completed**: Report Writing / 20/6/2007
102	Anuwar Ali	Finance	90000	**Course** / **Date_Completed**: Report Writing / 20/6/2007; Investments / 15/7/2007
103	Teka Lema	Information Technology	88000	**Course** / **Date_Completed**: Investments / 15/7/2007
105	Fatuma Kedir	Marketing	120000	**Course** / **Date_Completed**: Surveys / 30/8/2007; Survey Analysis / 10/9/2007

Figure 2 – EMPLOYEE relation - not in any normal form

This relation is not in any normal form because a normal-form relation must have only *simple values* in each row-column intersection. A simple value is one single value that does not have further components. The **values in the Courses attribute are made up of one or more rows** in another relation. This is not a simple value.

To make the relation into a first normal form relation, we must make each value in each column be a simple value. So, for example, the row for Emp_ID 100 becomes two rows:

Emp_ID	**Name**	**Department**	**Salary**	*Course*	*Date_Comleted*
100	Marta Negash	Information Technology	100000	C++	30/11/2007
100	Marta Negash	Information Technology	100000	Report Writing	15/10/2007

If we do the same for each row in the relation in Figure 2 above, the resulting relation will look like the one shown in Figure 3 below. All the values in each column are simple values – so this relation is in **first normal form**.

EMPLOYEE1

Emp_ID	**Name**	**Department**	**Salary**	*Course*	**Date_Completed**
100	Marta Negash	Information Technology	100000	C++	30/11/2007
100	Marta Negash	Information Technology	100000	Report Writing	15/10/2007
101	Alemu Ayalew	Administration	100000	Report Writing	20/6/2007
102	Anuwar Ali	Finance	90000	Report Writing	20/6/2007
102	Anuwar Ali	Finance	90000	Investments	15/7/2007
103	Teka Lema	Information Technology	88000	Investments	15/7/2007
105	Fatuma Kedir	Marketing	120000	Surveys	30/8/2007
105	Fatuma Kedir	Marketing	120000	Survey Analysis	10/9/2007

Figure 3 – EMPLOYEE1 relation with sample data – in first normal form

It is always possible to normalise a non-normal relation in this way.
In table EMPLOYEE2, each row in this table is unique for the combination of Emp_ID and Course. However, for some employees, the Emp_ID, Department and Salary values appear in more than one row – for Emp_ID 100, 102 and 105. So, if the salary for Emp_ID 100 changes, it needs to be recorded in 2 rows. Hence, there is still redundant data in this relation.
Moving onto the higher normal forms will remove the data redundancy.
However, we can see at this stage that the reason there is redundant data in this relation is that it contains data about two different entities – Employee and Course. The principles of normalization can be used to divide the relation into two relations – EMPLOYEE and EMP COURSE.
The EMPLOYEE relation is as in Figure 1. The other is shown below – EMP_COURSE. In the EMP_COURSE relation, the primary key is a composite of the Emp_ID and the Course.
EMP COURSE

Emp_ID	**Course**	**Date_Completed**
100	C++	30/11/2007
100	Report Writing	15/10/2007
101	Report Writing	20/6/2007
102	Report Writing	20/6/2007
102	Investments	15/7/2007
103	Investments	15/7/2007
105	Surveys	30/8/2007
106	Survey Analysis	10/9/2007

Figure 4 – EMP_COURSE relation, showing the Course and Date_Completed for each Employee-Course combination

Before looking at the higher normal forms, we must first understand the concept of **functional dependency**.

Functional Dependence

A **functional dependency** is a particular relationship between two attributes.

If a particular value of one attribute (A) in a relation uniquely determines the value of another attribute (B) in the same relation, then there is a functional dependency between attributes A and B.

This means that if we know the value of A, then we can determine a unique value for B. In this case, B is ***functionally dependent*** on A. A functional dependency can be expressed using an arrow i.e. this one can be shown as A → B. Or, in other words, A *determines* B.

In the EMPLOYEE example, shown in Figure 1 above, if we take a particular Emp_ID value, then we know that there is only one possible value for the Name attribute. For example, the Emp_ID value 100 always has a Name value of Marta Negash'. We can express this as:

Emp_ID → Name

A given Emp_ID value has *only one corresponding* Name value.

An attribute can be functionally dependent on two or more attributes.

Take for example, the EMP_COURSE relation shown in Figure 4. In this relation, the value of the Date_Completed attribute cannot be determined by the Emp_ID alone, nor can it be determined by the Course value alone – because the Date_Completed is a characteristic of an Employee taking a Course. The Date_Completed depends on the combination of the Emp_ID and Course values.

Note that the instances or rows in a relation (i.e. the sample data) do not prove the existence of a functional dependency. Knowledge of the system, obtained through the requirements analysis, is needed to understand the data and thus to identify functional dependencies.

Note also that a functional dependency A → B does not imply that the value of one attribute can be computed from the value of the other – it means that there can be only one value of B for each value of A. In addition, the reverse is not always true – i.e. there is not always only one value of A for each value of B e.g. there can be two Employees with the same Name, and they would have different Emp_ID values.

Second Normal Form (2NF)

A relation is in ***second normal form (2NF)*** *if every non-primary-key attribute is functionally dependent on the whole primary key. Thus no non-primary-key attribute is functionally dependent on part, but not all, of the primary key.*

The relation EMPLOYEE1, in Figure 3, is in 1NF but is not in 2NF. This relation can be written as follows, showing that the primary key consists of Emp_ID and Course.

EMPLOYEE1(Emp_ID,Name,Department,Salary,Course,Date_Completed)

The functional dependencies here are:
Emp_ID → Name,Department,Salary
Emp_ID,Course → Date_Completed

Name, Department and Salary are functionally dependent on Emp_ID, which is part of the primary key. So, these are non-primary-key attributes that are functionally dependent on part of the primary key.

We can say that second normal form is satisfied if any of the following apply:

1. The primary key consists of only one attribute (for example, the attribute Emp_ID in the EMPLOYEE relation shown in Figure 1)
2. No non-primary-key attributes exist in the relation.
3. Every non-primary-key attribute is functionally dependent on the full set of primary key attributes.

To convert a relation to second normal form, we need to remove the functional dependencies that violate the rules. To do this, the relation needs to be decomposed into new relations.
The new relations are based on the attributes that determine other attributes – also called the *determinants* – the determinants are the primary keys of the new relations.

So, if we take the functional dependencies, the new relations are as follows – primary keys are underlined:

EMPLOYEE (Emp_ID, Name, Department, Salary)
EMP_COURSE (Emp_ID, Course, Date_Completed)

Another way to look at this is to say that we take the functional dependency that violated the 2NF constraint i.e.
Emp_ID → Name,Department,Salary