BCPR203 – Database Management Systems

Normalisation

So, what is normalisation? Basically, it's the process of efficiently organizing data in a database. There are two goals of the normalisation process: eliminate redundant data (i.e. storing the same data in more than one table) and ensure data dependencies make sense (only storing related data in a table i.e. each entity has its own table). Both of these are important as they reduce the amount of space a database consumes and ensure that data is logically stored.

There are a series of guidelines for ensuring that databases are normalised. These are referred to as normal forms and are numbered from one (the lowest form of normalisation, referred to as first normal form or 1NF) through five (fifth normal form or 5NF). In practical applications, you'll often see 1NF, 2NF, and 3NF along with the occasional 4NF. Fifth normal form is very rarely seen and won't be discussed here.

Even though we will look at the process of normalisation it is important to understand that they are guidelines and guidelines only. Occasionally, it becomes necessary to stray from them to meet practical business requirements (usually related to database performance).

Functional Dependency

Definition:

A column (attribute) B is functionally dependent on another column A (or possibly a collection of columns) if each value for A in the database is associated with exactly one value of B.

If you are given a value for A in the database, do you know that it will be associated with exactly one value of B? If so, B is functionally dependent on A. If B is functionally dependent on A, you can also say that A functionally determines B.

Example: Rep Table

RepNum	LastName	FirstName	Street	City	Commission	PayClass	Rate
20	Kaiser	Valerie	624 Randall	Grove	$20,542.50	1	0.05
35	Hull	Richard	532 Jackson	Sheldon	$39,216	2	0.07
65	Perez	Juan	1626 Taylor	Fillmore	$23,487	1	0.05

In the Rep table above, LastName is functionally dependent on RepNum. If you are given a value of 20 for RepNum, for example, you know that you will find a *single* LastName, in this case Kaiser, associated with it. RepNum **determines** the LastName, the LastName **depends** on the RepNum.

First Normal Form

First normal form (1NF) sets the very basic rules for an organized database:

- Eliminate duplicative columns from the same table(i.e. no repeating groups of columns).
- Remove multivalued attributes into separate fields or, more usually, create another table for this attribute.
- Create separate tables for each group of related data and identify each row with a unique column or set of columns (the primary key).

Second Normal Form

Second normal form (2NF) looks at the process of removing duplicative data:

- Must be in 1NF
- No partial dependencies (dependencies that rely on *part* of the primary key).
- Create relationships between these new tables and their predecessors through the use of foreign keys.

Third Normal Form

Third normal form (3NF) goes one step further:

- Must be in 2NF
- No transitive dependencies (dependencies that rely on fields that are not part of the primary key)
- Create relationships between these new tables and their predecessors through the use of foreign keys.

Example

Let's begin by creating a sample set of data. Imagine we are working on a system to keep track of employees working on certain projects.

Table 1:

Project number	Project name	Employee number	Employee name	Rate category	Hourly rate
1023	Madagascar travel site	11	Vincent Radebe	A	$60
		12	Pauline James	B	$50
		16	Charles Ramoraz	C	$40
1056	Online estate agency	11	Vincent Radebe	A	$60
		17	Monique Williams	B	$50

There are problems with the above table. Tables in relational databases, which would include most databases you'll work with, are in a simple grid, or table format. Here, each project has a set of employees. So we couldn't even enter the data into this kind of table – if you wanted to add another person to the Madagascar project how can you do this?. And if we tried to use null fields to cater for the fields that have no value, then we cannot use the project number, or any other field, as a primary key (a primary key is a field, or list of fields, that uniquely identify one record). There is not much use in having a table if we can't uniquely identify each record in it. This table is not in 1NF because there is no primary key.

Another possible format is to have one row for each project.

Table 1B:

Proj num	Project name	Emp num1	Emp name1	Rate1	Hourly rate1	Emp num2	Emp name2	Rate2	Hourly rate2	Emp num3	Emp name3	Rate3	Hourly rate3
1023	Madagascar travel site	11	Vincent Radebe	A	$60	12	Pauline James	B	$50	16	Charles Ramoraz	C	$40
1056	Online estate agency	11	Vincent Radebe	A	$60	17	Monique Williams	B	$50				

This table has a primary key and no multi-valued attributes but it is not in 1NF because it has a repeating group of columns (EmpNum to HourlyRate). This is not a good database design. What would happen if a project had ten people working on it?

Table 2: The following table is in 1NF

employee_project table

Project number	Project name	*Employee number*	First name	Last name	Rate category	Hourly rate
1023	Madagascar travel site	11	Vincent	Radebe	A	$60
1023	Madagascar travel site	12	Pauline	James	B	$50
1023	Madagascar travel site	16	Charles	Ramoraz	C	$40
1056	Online estate agency	11	Vincent	Radebe	A	$60
1056	Online estate agency	17	Monique	Williams	B	$50

Notice that the project number cannot be a primary key on its own. It does not uniquely identify a row of data. So, our primary key must be a combination of project number and employee number. Together these two fields uniquely identify one row of data. Note that the employee name has also been split into Firstname and Lastname. The employee name is a composite attribute (i.e. it is composed of more than one part). Composite attributes are usually, but not always, split into their component parts, e.g. address is split into street, suburb, city, postcode, country. Note: This is **not** the same thing as multi-valued. There is only one address not many addresses.

So, now our data can go in table format, but there are still some problems with it. We store the information that code 1023 refers to the Madagascar travel site 3 times! – data redundancy. Besides the waste of space, there is another serious problem. Look carefully at the data below.

Table 3:

employee_project table

Project number	**Project name**	***Employee number***	**First name**	**Last name**	**Rate category**	**Hourly rate**
1023	Madagascar travel site	11	Vincent	Radebe	A	$60
1023	Madagascar travel site	12	Pauline	James	B	$50
1023	Madagascat travel site	16	Charles	Ramoraz	C	$40
1056	Online estate agency	11	Vincent	Radebe	A	$60
1056	Online estate agency	17	Monique	Williams	B	$50

Did you notice anything strange in the data above? Madagascar is misspelt in the 3rd record. Now imagine trying to spot this error in a table with thousands of records! By using the structure above, the chances of the data being corrupted increases drastically.

The solution is simply to take out the duplication. What we are doing formally is looking for partial dependencies, ie fields that are dependent on a part of a key, and not the entire key. Since both project number and employee number make up the key, we look for fields that are dependent only on project number, or on employee number as these are the two primary keys.

We identify two fields. Project name is dependent on project number only (employee_number is irrelevant in determining project name), and the same applies to employee name, hourly rate and rate category, which are dependent on employee number. So, we take out these fields, as follows:

employee_project table

Project number	***Employee number***
1023	11
1023	12
1023	16
1056	11
1056	17

Clearly we can't simply take out the data and leave it out of our database. We take it out, and put it into a new table, consisting of the field that has the partial dependency, and the field it is dependent on. So, we identified employee name, hourly rate and rate category as being dependent on employee number. The new table will consist of employee number as a key, and employee name, rate category and hourly rate, as follows:

Employee table

Employee number	**First name**	**Last name**	**Rate category**	**Hourly rate**
11	Vincent	Radebe	A	$60
12	Pauline	James	B	$50
16	Charles	Ramoraz	C	$40
17	Monique	Williams	B	$50

And the same for the project data.

Project table

Project number	**Project name**
1023	Madagascar travel site
1056	Online estate agency

Note the reduction of duplication. The text "Madagascar travel site" is stored once only, not for each occurrence of an employee working on that project. The link is made through the key, the project number. Obviously there is no way to remove the duplication of this number without losing the relation altogether, but it is far more efficient storing a short number repeatedly, than a large piece of text.

We're still not perfect. There is still room for anomalies in the data. Look carefully at the data below.

Table 4:

Employee table

Employee number	**First name**	**Last name**	**Rate category**	**Hourly rate**
11	Vincent	Radebe	A	$60
12	Pauline	James	B	$50
16	Charles	Ramoraz	C	$40
17	Monique	Williams	B	$40

The problem above is that Monique Williams has been awarded an hourly rate of $40, when she is actually category B, and should be earning $50 (In the case of this company, the rate category - hourly rate relationship is fixed. This may not always be the case). Once again we are storing data redundantly: the hourly rate - rate category relationship is being stored in its entirety for each employee.

The solution, as before, is to remove this excess data into its own table. Formally, what we are doing is looking for transitive relationships, or relationships where a non-key attribute is dependent on another non-key relationship. Hourly rate, while being in one sense dependent on Employee number (we probably identified this dependency earlier, when looking for partial dependencies) is actually dependent on Rate category. So, we remove it, and place it in a new table, with its actual key, as follows.

Employee table

Employee number	**First name**	**Last name**	**Rate category**
11	Vincent	Radebe	A
12	Pauline	James	B
16	Charles	Ramoraz	C
17	Monique	Williams	B

Rate table

Rate category	**Hourly rate**
A	$60
B	$50
C	$40

It is now impossible to mistakenly assume rate category "B" is associated with an hourly rate of anything but $50. These relationships are only stored in one place - our new table, where it can be ensured they are accurate.

The end product of the normalisation process is now four tables where we started with one and each table is in 3NF.

RATE TABLE

Rate category	**Hourly rate**
A	$60
B	$50
C	$40

EMPLOYEE TABLE

Employee number	**First name**	**Last name**	**Rate category**
11	Vincent	Radebe	A
12	Pauline	James	B
16	Charles	Ramoraz	C
17	Monique	Williams	B

PROJECT TABLE

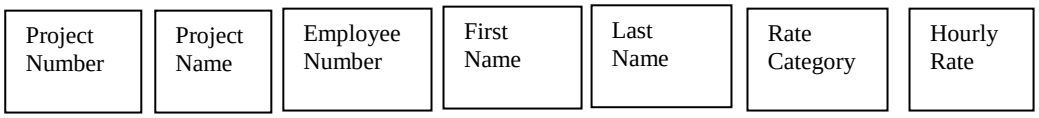
Project number	**Project name**
1023	Madagascar travel site
1056	Online estate agency

PROJECTEMPLOYEE TABLE

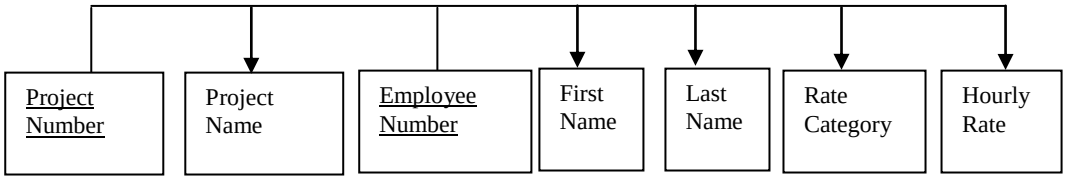
Project number	***Employee number***
1023	11
1023	12
1023	16
1056	11
1056	17

Dependency Diagrams

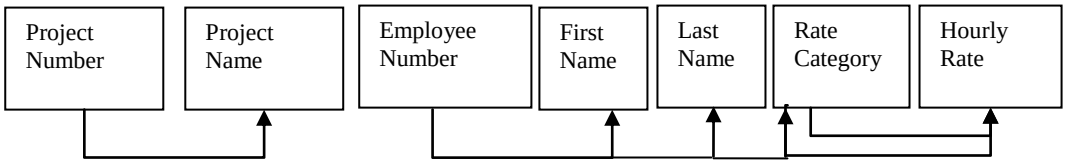
Let's now draw a dependency diagram representing this process. First of all we start off with all of the fields we are concerned with.



Then we need to assign a primary key or keys. In this case we need two fields to give us this uniqueness – Project Number and Employee Number. We show the primary key by an underline. (If we know the Project Number and the Employee Number then we know the Project Name, First Name, Last Name, Rate Category and Hourly Rate)

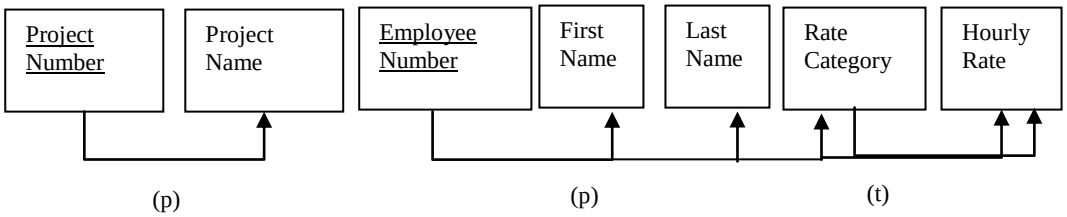


Then we draw a line below showing the dependencies.



- If I know the Project Number I know the Project Name.
- If I know the Employee Number I know the First Name, Last Name and Rate Category
- If I know the Rate category I know the Hourly Rate.

We then need to look at which dependencies are partial (p) dependencies (those that depend on part of the primary key) and transitive (t) dependencies (those that depend on a non-primary key attribute).



1NF, 2NF and 3NF

First Normal Form (1NF)

We can write this out like this: (one table that would have all the data – but this is not fully normalised)

ProjectNumber, EmployeeNumber, ProjectName, FirstName, LastName, RateCategory, HourlyRate

Second Normal Form (2NF)

To get to 2NF we need to remove the partial dependencies. We will create a separate table that contains only the primary key's as these fields must always stay together.

ProjectNumber, EmployeeNumber,

And we remove the partial dependencies and create a table for each of these dependencies.

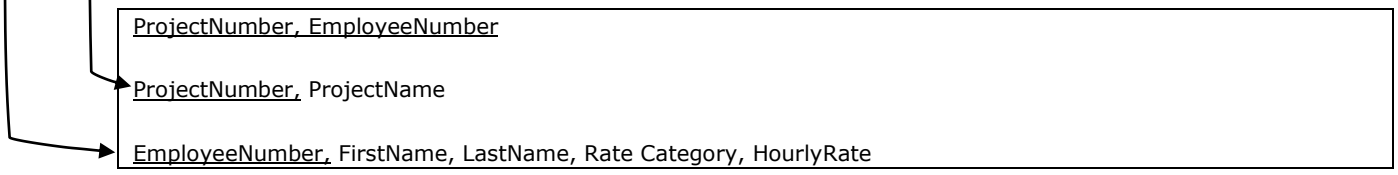
ProjectNumber, ProjectName

EmployeeNumber, FirstName, LastName, RateCategory, HourlyRate

Any fields that are left over remain in the primary key table. So the primary key table will now look like this.

ProjectNumber, EmployeeNumber

We now have three tables and we are in 2NF



Third Normal Form (3NF)

In 3NF we need to remove the transitive dependencies.

We now create a fourth table to remove the transitive dependency.

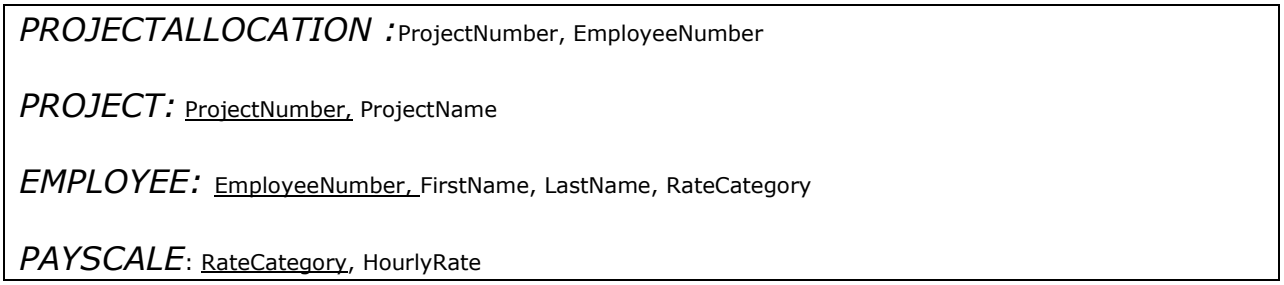
RateCategory, HourlyRate

This table will require a primary key, which we underline.

RateCategory, HourlyRate

We now have four tables which are in 3NF.

We would then give each table an appropriate name (e.g. ProjectAllocation, Project, Employee and PayScale) and normalisation is done.



Extracts from:

http://databases.about.com/library/weekly/aa080501a.htm

Pratt, P.J., Adamski, J.J., *Concepts of Database Management* [2005], Thomson Learning, USA.