Normalization

1. Introduction

Let us consider the following table.

Tab. 2.1
An example of non-normalized table

Team's Member	Salary	Project Team	Project' Budget	Role in the team
Red	1500€/month	Alfa	1000000€	Designer
Red	1500€/month	Beta	500000€	Designer
Red	1500€/month	Gamma	1500000€	Chief Designer
Green	2500€/month	Alfa	1000000€	PM
Green	2500€/month	Gamma	1500000€	PM
White	5000€/month	Beta	500000€	Controller
Brown	800€/month	Alfa	1000000€	Junior

It is easy to see that *this table is conceptually wrong*. Some of the problems related to it are the following ones:

- *The salary of Red is repeated in three records. This is a clear example of* **data redundancy**. Note that also the name Red is repeated in three records, but this repetition is licit, as it is needed to make clear that Red participates (or has participated) to three different projects;
- *If the salary of Red varies* (for career advancement) *the new value should be modified in all three records*. This is called **updating anomaly**;
- *If Red resigns* (or is fired) *all three records relative to him should be deleted*. For this reason, *all the information related to him* (i.e., his name, his salary and the projects he was involved in) *would get lost*. This is called **deletion anomaly**;
- *If a new employee is hired, it is not possible to include his data in the database, until he or she is not assigned to a specific project's team*. This is called as **insertion anomaly**.

To avoid these anomalies and to assess the consistency and the correctness of a relational database, a set of checks, based on the concept of **functional relation**, must be made. Formally *these checks are called normalizations rules* and they are *needed to ascertain* that:

- *Each field contains a single value;*
- *Each table has a primary key;*
- *Repetitions and data redundancy have been avoided;*
- *The value of a field does not depend on the value of another field.*

To assure that the above-mentioned requirements have been met *six rules* (or more formally six normal forms) can be used; a *RDB that complied with all the normal forms is said to be normalized* or to be in normal form. However, the last three normal forms are too stringent and so, *in practice only the first three are used*.

2. First Normal Form - 1st NF

As we know, to assure referential integrity, each record of table must be unique and, for this reason, *each table must have a Primary Key (PK)*. This obvious concept is generally referred to as the **Zero Normal Rule**.

The **first normal rule**, also called **atomic form**, **assures that a field of a record cannot contain a multiplicity of values**. In other words, *a field cannot contain a vector or any array structure*. Specifically, if a table is in first normal form, then there is no possibility to subdivide the table's fields any further. The reasons why one should not use any fields containing more than one value is due to the following main reasons:

- Data filtering, data sorting, and data search cannot be efficiently performed in case of fields containing multiple values;
- It is not possible to define relations among data if there are fields containing multiple values.

To clarify this concept let us consider Tab. 2.2

Tab. 2.2
First normal rule is not respected

Id	Name	Surname	Address
1	Frank	Red	Carnaby Street, London, UK
2	Paolo	Verdi	Via Curzio, Padova, Italy
3	John	White	Trafalgare Square, London, UK
…	…	…	…

In this case the Address field violates the 1st NR as it contains three values one for the street/square, one for the city and one for the Country. This makes almost impossible to sort, filter and search the records, based on the Address field. For example, it would be difficult to show only British people.

Obviously, to correct this table, *it is sufficient to split the address field into three distinct fields* that are: Street, City and Country.

The following table shows a situation that is even trickier. Let us consider the protocols that can be made in a town hall. If the town hall has n offices, a protocol may pass through all the n offices, but this is a limiting condition; most of the time a protocol will go through many offices, let say m, with m lower than n.

If we want to code in a DB the sequence of offices that must be visited by a protocol, we could make a table like Tab. 2.3:

Tab. 2.3
First normal rule is not respected

Id	Protocol	Office
1	P1	O1, O2, O3
2	P2	O2, O1
3	P3	O4, O1, O5
…	…	…

In this case the problem concerns the Office field. In the case of Tab. 2.8 it was easy to understand that the Address field had to be split into three distinct fields. In this case the solution is not so obvious. *How many fields should we add? Also, if we split the Offices field into many fields, how can we code the order with which offices must be visited?*

A possible solution is shown by Tab. 2.4, where *n* fields (*n* is the total number of offices of the town hall) have been added. The first field corresponds to the first visited office, the second field correspond to the secondo visited office and so on. Thus, if the value O_j appears in the ith field it means that the *j*-th office is the *i*-th to be visited. Obviously if a protocol visits *m* offices (with $m < n$) then all the fields from the $(m + 1)$-th to the *n*-th will be null.

Tab. 2.4
The modified Table for Protocols' Iter

ID	Protocol	Office 1	Office 2	Office 3	…	Office n
1	P1	O1	O2	O3	Null	Null
2	P2	O2	O1	Null	Null	Null
3	P3	O4	O1	O5	Null	Null
…	…	…	…	…	…	

This solution could work, but it still has problems:

- If a protocol visits an office more than once *n* fields may not be enough to describe all the routing of the protocol;
- Records are very long and most of the fields are null;
- It is not easy to find all the protocol that visits an office (since the filter should be made on each one of the *n* Office fields).

Also, and perhaps more important, if we wanted to treat Protocols and Offices as Meta-Data (i.e., in addition to the name we want to associate additional information concerning a protocols and offices), then Tab. 2.4 would be useless[1].

In this case *we should create two tables PROTOCOLS and OFFICES that, clearly, are in MTM relation*. The bridge table needed to split the MTM relation is perfect to define the administrative process followed by each protocol. To this aim, it is sufficient to create records with three fields: *(i) Protocols_ID, (ii) Office_ID and (iii) Position*. The first two are the forward keys that, together, generate the primary key of the table. The last field is used to define the position of an office, within the routing of a certain procedure.

For instance, a record such as {P1, O3, 6} means the sixth office visited by Protocol #1 is Office #3.

3. Second Normal Form - 2nd NF

«A RDB is said to be in second normal form if and only if every non-prime attribute of a Table (i.e., a field that does not belong to any primary key) depends, completely, on the primary key and it does not depends on one side (or part) of the primary key».

In other words, the **second normal form states that**, in a table that is already in first normal form, **all the fields must depend, exclusively, on the value of the primary key**. This ensures that the PK represents, uniquely, each record in the table. Additionally, this rule also implies that **calculated fields are not to be registered physically**. For instance, Table 2.5 displays a table, namely ORDERS, that is in first normal form; indeed, it has a primary key and there are no fields that contain more than one value.

Tab. 2.5
A table that is not in second normal form

Order_Id	Quantity	Unit Price	Invoice
1	10	100€	1000€
2	50	500	25000€
3	5	1000	5000€
…	…	…	…

However, this table is conceptually wrong. *The problem is due to the presence of the Invoice field*. This field depends both on the unit price and on the quantity purchased fields (i.e., invoice = unit price × quantity), but this fields do not form a primary key. The table, therefore, is not in second normal form: the invoice field must be removed from the table and, if needed, it must be calculated at run-time through a query, using the mathematical functions that can be invoked inside the Structured Query Language (SQL).

[1] *Can you guess why?*

The second normal form also applies to those tables that have a Primary Key composed by more than one field (bridge tables are a typical example); indeed, according to the definition *"a field must depend on the whole PK and not only on one side of it"*. Let us consider Table 2.6 that forms a bridge between the ITEMS and the WAREHOUSES tables. As previously discussed, ITEMS and WAREHOUSES are, typically, in a MTM relation because an item may be stored in several warehouses and a warehouse can store different items. Thus, the bridge table must have two Forward Keys, one related to Item_Id and one to Warehouse_Id, which, taken together, also form the Primary Key.

Tab 2.6
A table that is not in second normal form

War_Id (FK)*	Item_Id (FK)*	W_Name	I_Name	On Hand	Safety Stock
1	1	Central	Pr#1	1000	500
1	2	Central	Pr#2	2000	200
…	…	…	…	…	….
2	1	North-East Coast	Pr#1	800	100
…	…	…		…	…

It is evident that:

- The name of a warehouse depends, only, on War_Id, which is a part of the Primary Key;
- The name of an item depends, only, on Item_Id, which is a part of the Primary Key;
- The available stock (i.e., on hand) depends both on War_Id and on Item_ID that, together form the Primary Key;
- The safety stock depends both on War_Id and on Item_ID that, together form the Primary Key.

Consequently, to comply with the 2nd NF, W_Name and I_Name must be removed and placed (as additional fields) in the WAREHOUSES and ITEMS table, respectively. Conversely, it is correct to place On Hand and Safety Stock in the bridge table.

4. Third Normal Form - 3rd NF

The third normal form (also called *Boyce and Codd normal form*, from the name of its creators) state that: *«A RDB is in 3rd NF if and only if all the redundancies based on functional dependencies have been removed».*

In other words, a RDB is in 3rd normal form provided that, for each functional dependency of the form $X \rightarrow Y$, the following conditions hold:

- $X \rightarrow Y$ is a trivial functional dependency (i.e., $X \subseteq Y$);
- X is a super key of a Table.

To understand the meaning of this formulation we need to specify what is meant by super-key and by functional dependency.

A **super-key is a set of attributes within a table, whose values can be used to uniquely identify a record**. A **candidate key** is a minimal set of attributes necessary to identify a record; this is also called a **minimal super-key.** Due to what we have said before, *in a standard table a minimal super-key is made of a single field, whereas in a bridge table a minimal super-key is made, at least, of two fields*.

In order to define a functional dependency, let us consider Table T made of *n* fields and two non-empty subsets X and Y of those fields. Then we say that, in T, **there is a functional dependency X → Y (i.e., X functionally determines Y) if for each pair of record r_i and r_j of T with the same values of the attributes in X, then r_1 and r_2 have the same values also for the attributes in Y.**

For instance, if a table T contains *m* = 100 records each one made *n* = 7 fields, four subsets could be defined as follows (see Fig. 2.1):

- X contains the first three fields;
- Y contains the last two fields;
- K contains the first two fields;
- Z contains the fifth field.

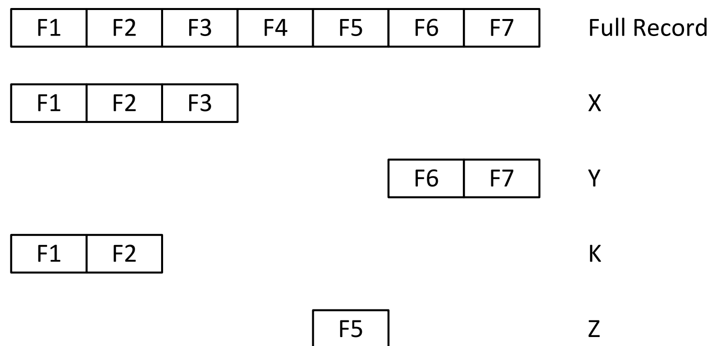

Fig. 2.1 *A record and possible sub-sets*

A functional dependency of the form X → K certainly exists, because K ⊂ X i.e., this is a trivial functional dependency. Conversely, it is much more interesting to see if a functional dependency X → Y can be identified. To this aim *all the possible couples of records r_i and r_j, with $i \in \{1,...,100\}$ and $j \in \{1,...,100\}$, must be considered: a functional dependency exists if and only if any time that the first three fields of r_i have the same values of r_j, then also the last two fields of r_i and r_j coincide*, as shown by Fig. 2.1.

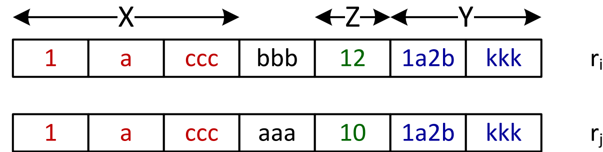

Fig. 2.2 *An example of functional dependency* $X \rightarrow Y$

Lastly, from Fig. 2.2 we can see that a functional dependency of the form $X \rightarrow Z$ does not exist, because there exists at least a couple of records r_i and r_j having the same values of the first three fields but that differ for the value of the fifth field.

To explain how the 3rd NF can be used, let us consider once again the example of Table 2.1. Specifically, it is easy to see that there *are many non-trivial functional dependencies*, including the following three:

- *Team's member* $\rightarrow$ *Salary;*
- *Project* $\rightarrow$ *Budget;*
- *{Team's Member, Project}* $\rightarrow$ *Role;*

What is important to stress is the fact that *all the anomalies discussed in the introduction section can be traced back to the presence of the first two functional dependencies*. Conversely, *the last one does not cause any anomaly. The reason lies in the fact that the third Functional Dependency has a super-key on its left side, but the other ones do not*. In other words, neither {Team's member} nor {Project} are primary keys of the Table; actually, only the fields {Team's Member, Project} form a primary key of the table.

Thus, accordingly to the third normal form of Boyce and Codd, *stating that the left part of a functional dependency must be a super-key, the solution is as follows*:

- *Two tables must be created, respectively MEMBERS and PROJECTS;*
- *{Team's member} and {Project} should be used, respectively, as the primary keys of the MEMBERS and PROJECTS tables;*
- *MEMBERS and PROJECTS are in a MTM relation (a member can participate to several projects and each project has a team composed by many members) and so a bridge table, namely PROJECTS' MEMBERS, must be created. Also, this table should use the couple of forward keys {Team's Member, Project} as primary key.*

Summarizing the first functional dependency corresponds to the MEMBERS table, the second one corresponds to the PROJECTS table and the last one corresponds to the bridge table PROJECTS' MEMBERS.

Thus, as clearly shown by the example, from a practical point of view we could argue that the third normal form extends to the entire RDB what the first two rules require for each Table. In practice **redundancies are avoided by decomposing the database in as many tables as the (non-trivial) functional dependencies that can be identified.**

5. Relaxed Third Normal Form

Without going in excessive details, we note that the *Boyce and Codd normal form is very stringent to be realized; thus, in practice, a less stringent one is frequently used*. According to Codd, this "relaxed" form can be enunciated as follows: *a Table T (of a RDB) is in 3rd NF if and only if both the following conditions hold:*

- *T is already in 2nd NF;*
- *Every non-prime attribute of T is non-transitively dependent on every key of T.*

Where a **Non-Prime attribute** of T is an attribute that does not belong to any candidate key of T and a **Transitive Dependency** is a functional dependency in which X $\rightarrow$ Z (X determines Z) indirectly, by virtue of X $\rightarrow$ Y and Y $\rightarrow$ Z.

An equivalent 3rd NF definition given by Zaniolo states that: *a table is in 3rd NF if and only if, for each of it functional dependencies X $\rightarrow$ A, at least one of the following conditions holds:*

- *X contains A i.e., X $\rightarrow$ A is trivial functional dependency;*
- *X is a super-key;*
- *Every element of the set difference (A - X) is a prime attribute (i.e., each attribute in (A - X) is contained in some candidate key).*

Please note that is exactly the third condition that differentiates the relaxed 3rd NF from the more stringent Boyce and Codd normal form.

It is interesting to note how only in rare cases a relaxed 3rd NF table does not meet the requirements of the Boyce and Codd Normal Form. *A 3rd NF table that does not have multiple overlapping candidate keys is guaranteed to be in Boyce Codd NF.* Conversely, a 3rd NF table with two or more overlapping candidate keys may or may not be in Boyce Codd Normal Form.

Table 2.7 is an example of a 3rd NF table that does not meet Boyce and Codd requirements.

Tab. 2.7

A table in 3rd NF that does not comply with Boyce and Codd requirements

		Today's Court Bookings		
Court	**Date**	**Start Time**	**End Time**	**Rate Type**
1	05/10/2017	09:30	10:30	Saver
1	05/10/2017	11:00	12:00	Saver
1	05/10/2017	14:00	15:30	Standard
2	05/10/2017	10:00	11:30	Premium-B
2	05/10/2017	11:30	13:30	Premium-B
2	05/10/2017	15:00	16:30	Premium-A
…	…	…	…	…

Each row in the table represents a court booking at a tennis club that has one hard court (Court 1) and one grass court (Court 2). Also, a booking is coded adding to the Court Id (Court field) the date and time in which the Court is reserved (Date Start Time and End Time fields, respectively).

Additionally, each booking has a Rate Type associated with it and there are four distinct rate types:

- *Saver* – Valid for Court 1 if bookings are made by members;
- *Standard* - Valid for Court 1 if bookings made by non-members;
- *Premium_A* – Valid for Court 2 if bookings are made by members;
- *Premium_B* – Valid for Court 2 if bookings are made by non-members.

The table's super-keys are[2]:

- **S1 = {Date, Court, Start Time}**
- **S2 = {Date, Court, End Time}**
- **S3 = {Date, Rate Type, Start Time}**
- **S4 = {Date, Rate Type, End Time}**
- S5 = {Date, Court, Start Time, End Time}
- S6 = {Date, Rate Type, Start Time, End Time}
- S7 = {Date, Court, Rate Type, Start Time}
- S8 = {Date, Court, Rate Type, End Time}
- ST = {Date, Court, Rate Type, Start Time, End Time}, the trivial super-key made of all fields

However, only S1, S2, S3 and S4 are candidate keys (i.e., minimal super-keys) because e.g. S1 $\subset$ S5, so S5 cannot be a candidate key.

Now, recall that the 2nd NF prohibits partial functional dependencies of non-prime attributes (i.e., an attribute that does not occur in any key) on candidate keys, and that the 3rd NF prohibits transitive functional dependencies of non-prime attributes on candidate keys.

In Tab. 2.6 there are no non-prime attributes: that is, all attributes belong to some candidate key. Therefore the table adheres to both the 2nd NF and 3rd NF. Yet Tab. 2.6 does not adhere to Boyce and Codd requirements, due to the Rate Type $\rightarrow$ Court functional dependencies in which the left side part (Rate Type) is neither a candidate key nor a superset of a candidate key.

Thus, to be compliant with Boyce and Codd requirements, an additional Table RATES, in OTM relation with the COURTS table, should be created.

[2] *Even though Start Time and End Time attributes have no duplicate values for each of them, we still must admit that in some other days, two different bookings on court 1 and court 2 could start at the same time or end at the same time. This is the reason why {Start Time} and {End Time} cannot be considered as the table's super-keys.*