

Normalisation is a process of efficiently organizing data in a database. Knowing the principles of normalization and applying them to your daily database design isn't all that complicated and could drastically improve the performance of your DBMS. This section introduces the concept of normalization and takes a brief look at the most common normal forms.

**The purpose of normalization is to:**

- efficiently organize data
- eliminate redundant data
- avoid storing unrelated data in a table
- reduce the amount of space consumed by a database
- ensure faster processing of information, improves access rate



By following the rules of normalization, we will be able to store data in a logical manner.

Database academics have developed a series of guidelines for ensuring that databases are normalized. These are referred to as normal forms and are numbered from one (the lowest form of normalization, referred to as first normal form or 1NF) to five (fifth normal form or 5NF). We will concentrate on 1NF, 2NF and 3NF since 4NF and 5NF are rarely used. A **primary key** is a field that uniquely identifies records in a table. The primary key field cannot contain duplicates. A **foreign key** is a field in a table that is not a primary key field. A foreign key field may contain duplicates. All entries in the field of the foreign key must be contained in the table where that field is a primary key field. **Anomalies** refer to redundant data or any occurrences that weakens the integrity of your data due to irregular or inconsistent data.

### First Normal Form (1NF)

sets the very basic rules for an organized database.

- eliminates duplicate data in rows from the same table
- ensures the table has a primary key.
- all non key fields depend partly or fully on the primary key field

### What do these rules mean when contemplating the design of a database?

These rules simply mean that there must be no duplicated data within the same row of a table and within the column of the primary key field. Also, all fields must somewhat depend on or fully depend on the field chosen as a primary key field.

### Second Normal Form (2NF)

Further addresses the concept of removing duplicate data.

- meets all requirements of the first normal form (1NF).
- remove subsets of data that apply to multiple rows of a table and place them in separate tables.
- create relationships between these new tables and their predecessors through the use of primary and foreign keys.
- remove columns / fields that are not fully dependant on the Primary key.

These rules can be summarized in a simple statement: 2NF attempts to reduce the amount of data in a table by extracting subsets of data, placing it in new tables and creating relationships between these tables.

### Third Normal Form (3NF)

- meets all the requirements of the 1NF and 2NF.
- no transitive dependencies, that is, a field should not depend on another field that is not a primary key field.
- removes all derived fields that are not completely dependant on the primary key.

### Let us work through an example to illustrate the process of normalization.

During the July vacation, Rainbow University hosts tuition sessions for Grade 12 learners. Students may choose a subject that runs for a period of two weeks. The University requires information of all students that attend these courses for the purposes of area statistics and fees.

**The information required by the university is as follows:**

Student Number, Name, Surname, ID, City, Code, Subject enrolled for, Lecturer, Rate per Hour per Subject, Number of Hours per subject and Fees.

(Each subject will differ with regards to the rate per hour and number of hours)

### First normal form (1NF)

Create separate tables for related data to avoid repeating groups of information.

Set primary and foreign keys.

**StudentTable** (SdtNo, City, Subj)

**DetailsTable** (SdtNo, ID, Name, Surname)

**AreaTable** (City, Code)

**SubjectTable** (Subj, Lect, RatePerHr, NoOfHrs, Fees)



An example of duplicated data would mean to put information from AreaTable into StudentTable. There would be many students from a particular city and this would mean for every student number, you would be repeating this information. Rather have an area table that contains a single entry of the city with the code and create a one to many relationship from AreaTable to StudentTable.

### Second normal form (2NF)

All requirements for 1NF must be met. Remember by assigning fields from one table into another table, where the fields are not directly related to the primary key field would be defying the rules for 2NF. All non key fields must be fully dependant on the primary key field. Let us take a look at SubjectTable (Subj, Lect, RatePerHr, NoOfHrs, Fees)

Subj is the primary key field. Are all other fields fully dependant on the Subj field? The field "Lect" is safe as it is fully dependant on "Subj", the field "RatePerHr" is safe as it fully depends on the "Subj" selected by the student, the field "NoOfHrs" is safe as it fully depends on the "Subj" selected by the student. There is a problem with the field "Fees" as it is not fully dependant on the RatePerHr and the NoOfHrs fields. The identification of the fields to be removed or adjusted is done in NF2. The actual removal or adjustment of the fields is done in NF3. This field must be removed from the table.

### Third normal form (3NF)

Removes the field called Fees from the table. The table will now contain the following fields. SubjectTable (Subj, Lect, RatePerHr, NoOfHrs)

All requirements for 1NF and 2NF must be met. There must be no transitive dependencies. What does that mean? Non-key fields must not depend on other non key fields for a value. If the Subject and fees were required and if Fees was still part of the SubjectTable, The query would be as follows:

```
SELECT Subj, RatePerHr * NoOfHrs as Fees
FROM SubjectTable
```

After normalisation, we would have removed fees and our query would be

```
SELECT Subj, Fees FROM SubjectTable
```

This will achieve the same results without violating normalization rules.

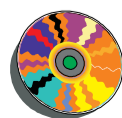
## CREATING TABLES IN A DATABASE

### A Data Dictionary

The conceptual design of tables in a database is presented in the form of a data dictionary. This is a useful feature for both the programmer and the user as it clearly describes the type of contents each field can contain. The following characteristics of a field may be used to create a data dictionary.

### Specify the Table Name.

**Selecting Field Names:** Choose a field name that clearly describes the information the field will contain.



**Data Types:** Databases offer a variety of data types in Design View. By clicking on the drop down arrow next to the field name, one would gain access to the different data types available. The purpose of the data type is to restrict entry into that field.

If you need to store a value eg. 0347, rather use the 'Text' data type instead of 'Number' as Number will erase the first 0 in the value.

**Field Size:** The width of a field is often used with Text fields. Eg. The size of the field to store ID numbers should be restricted to 13.

**Input Mask:** specifies the exact format of the information to be entered into fields of a table. The purpose of the input mask is to force users to enter data in a specific way to maintain data integrity rules.

**Validation Rules:** allow the database designer more control over the type of information entered into the fields of the table.

Although the input mask specifies the format of the data, they do not allow complete control over the exact information required. A validation rule specifies the accepted range of values. Eg An input mask could specify a string of 3 digits, but a validation rule could specify that the numbers must be in the range 350 899

A **Validation Text** property alerts the user of any mistakes during entry.

**Default Value:** is the value that automatically appears in the field when a new record is created

Eg A Library Database could have a field for outstanding fines with a default value set to 0.00. For every new record created, this value will automatically appear in this field

## FORMULATING AN INPUT MASK:

### Required Entry

SYMBOL	DESCRIPTION	EXAMPLE
O	A digit from 0 - 9 must be entered	0000 requires an entry of a 4 digit number Eg 1234
L	A letter from A - Z must be entered <i>Not case sensitive</i>	LLLLL requires an entry of 5 letters Eg Abcde
A	A digit or letter must be entered	AAA requires an entry of 3 letters or digits Eg A3B, B29,7YX
&	Any character or a space must be entered	&&&& requires an entry of 4 characters or spaces Eg A 4b, D*3%, #SBC

### Optional Entry

SYMBOL	DESCRIPTION	EXAMPLE
9	A digit from 0 - 9 or a space. Operational signs are not allowed eg + - * /	9999 allows up to 4 digits or spaces Eg 123, 5 61, 6543
#	The same as '9', further allows for the plus and minus signs.	##### allows up to 5 digits or spaces with or without signs. Eg -67, 786, +8999
?	A letter from A - Z. <i>Not case sensitive</i>	????? allows up to 5 characters Eg XYZ, Maria, abcd
a	A letter, digit or space.	aaaa allows up to 4 letters or digits Eg 74b, KL8, moon
C	Any character or space	CCCCCC allows up to 7 characters or spaces like G Force, I * U, L O L

### OPERATORS USED TO FORM AN INPUT MASK

.,:;- /	Decimal placeholder, thousand, date and time.	###.## allows for a max of 3 digits to the left and two to the right. Eg 27.99, 765.89, 21.6
<	Causes all characters following to be converted to lowercase.	<LLLL will ensure all 4 letters are stored in lowercase.
>	Causes all characters following to be converted to uppercase.	>LLL will ensure all 3 letters are stored in uppercase.
!	Causes all characters to be filled from the rightmost. Useful for monetary values	9999 will store the value 789 as 789_. 9999! will store as _789.