

There are many rules that help in designing a good relational database. Among them are called the **normal forms**. There are altogether 5 normal forms. For our purpose, we only need to know the first three.

1. The First Normal Form (1NF)

A table is in 1NF if each row is unique and each field stores a single value (rather than a collection of values).

All of the tables we have seen and designed so far are in 1NF. Below is an example of a table that is not in 1NF. Can you see why?

CUSTOMER_ID (key)	LAST_NAME	FIRST_NAME	PHONES
JS1	Smith	John	310-456-4022 (W) 310-444-8712 (H)
PH1	Pocahontas		310-432-2813 (H)

For the above table, it is difficult to look up phone numbers and to make any change to them. This is one of the reasons why the table design is bad. How do we fix this problem? There are at least two ways to fix this problem.

One way is design a table with two phone fields instead of one as follows.

CUSTOMER_ID (key)	LAST_NAME	FIRST_NAME	WorkPhone	HomePhone
JS1	Smith	John	310-456-4022	310-444-8712
PH1		Pocahontas		310-432-2813

The above table design is certainly in 1NF. However it is not a very “flexible” design. Can you foresee any problem with the above design? What is tomorrow, Pocahontas has fax number? How can we enter this information? We will need to redesign the above table by adding another Fax field, won’t we? Redesigning tables after data are entered is a very costly action because data may be lost and will need to be re-entered.

The above table design does not reflect the *one-to-many relationship* between customers and phone numbers: a customer can have zero or more phone numbers. To fix this problem, we need at least two tables: one table just for the customers with customer ID as key but without any phone number and one table containing phone numbers with customer ID as a foreign key and a phone type field.

In class exercise: build a database called FirstNForm with the following tables.

tblCustomer table

CUSTOMER_ID (key)	LAST_NAME	FIRST_NAME
JS1	Smith	John
PH1		Pocahontas

tblPhone table

PhoneNum	Customer_ID (foreign key)	PhoneType
310-456-4022	JS1	Work
310-444-8712	JS1	Home
310-432-2813	PH1	Home

Set PhoneNum together with Customer_ID as a **pair key**. This is an example of what is called a **composite key**.

Use the Lookup Wizard to connect the common field Customer_ID together. In the Relationship diagram, set the link between tblCustomer and tblPhone to one-to-many.

Now, if tomorrow Pocahontas obtains a fax number, all we have to do is add the fax number with PH1 as Customer_ID and Fax as PhoneType to the tblPhone table. We do not have to redesign any table at all.

To avoid typing the PhoneType each time a new phone is added, we can create a third table that contains only the phone type as key and connect it to the tblPhone table via the common field PhoneType, as follows.

tblPhoneType

PhoneType (key)
Home
Work
Fax

2. The Second Normal Form (2NF)

A table is in 2NF if

- it is in 1NF, and
- each row has a primary key (which can be a combination of several fields), and
- each non-key field depends on the entire primary key and not on only parts of the key.

Below is a table that is **not** in 2NF.

COURSE	SECTION	INSTRUCTOR	COURSE_NAME
Cosc 250	1	Nguyen	Computer Science for Business
Cosc 250	2	Warford	Computer Science for Business
Cosc 250	3	Nguyen	Computer Science for Business
Cosc 250	4	Zimmerman	Computer Science for Business
Cosc 480	1	Nguyen	Programming Languages

The primary key for the above table is the combination (COURSE, SECTION). The Instructor field depends fully on this compound key. However, the COURSE_NAME field depends only on Course. As a result, the COURSE_NAME field has what is called the problem of data redundancy. In the above table, the course name “Computer Science for Business” is repeated unnecessarily in many rows. It is possible to mistyped this same course name is more than one row giving rise to inconsistencies in data.

Again the above table design does not properly reflect the *one-to-many relationship* between instructors and courses: an instructor can teach many courses. To fix this problem, we break up the above table into separate tables related via the COURSE field.

tblCourse Table

COURSE	COURSE NAME
Cosc 250	Computer Science for Business
Cosc 480	Programming Languages

tblSection Table

COURSE	SECTION	INSTRUCTOR
Cosc 250	1	Nguyen
Cosc 250	2	Warford
Cosc 250	3	Nguyen
Cosc 250	4	Zimmerman
Cosc 480	1	Nguyen

3. The Third Normal Form (3NF)

A table is in 3NF if

- it is in 2NF, and
- there is no transitive dependency, that is, each non-key field depends directly on the primary key field(s) only.

Below is a sample of a table that is **not** in 3NF. Can you see why?

EMP_ID	EMP_NAME	JOB_CODE	JOB_TITLE	DATE_HIRED	JOB_DESCRIPTION
A120	Jones	1	Programmer	9/17/95	Write computer code.
A721	Harpo	1	Programmer	7/17/93	write computer code.
B270	Garfunkel	2	Analyst	1/12/95	Perform cost analysis.
C273	Selsi	3	Designer	5/21/94	Design graphics.

In the above table, the primary key is EMP_ID. However, JOB_TITLE and JOB_DESCRIPTION depend on JOB_CODE and not on EMP_ID. This causes the problem of data redundancy in JOB_TITLE and JOB_DESCRIPTION. As a result, this is a bad design.

The above design does not reflect the one-to-many relationship between Job and Employee: a job

can be held by many employees; for example, the Programmer job is held by two employees. To fix this problem, we break the above table into two separate tables related to each other via the JOB_CODE field.

tblEmployee Table

EMP_ID	EMP_NAME	JOB_CODE	DATE_HIRED
A120	Jones	1	9/17/95
A721	Harpo	1	7/17/93
B270	Garfunkel	2	1/12/95
C273	Selsi	3	5/21/94

Jobs Table

JOB_CODE	JOB_TITLE	JOB_DESCRIPTION
1	Programmer	Write computer code.
2	Analyst	Perform cost analysis.
3	Designer	Design graphics.

What kind of relationship link should you create between these two tables?

4. Many-to-many relationship

What if in a company, an employee can hold multiple jobs and a job can be held by many employees? For example, an employee can be both a programmer and an analyst, and there are many employees holding the programmer job. This is an example of what is called a many-to-many relationship between two entities. Here, we say that there is a many-to-many relationship between employee and job.

To represent a many-to-many relationship between two entities, we need three tables:

- One table for the first entity,
- One table for the second entity,
- One table that is connected to both of the above two tables.

Here are the table designs that that represent the many-to-many relationship between employee and job.

Design of tblEmployee

Field Name	Data Type	Description
EmployeeID	Text	key
LastName	Text	
FirstName		

Design of tblJob

Field Name	Data Type	Description
JobCode	Text	key
JobName	Text	
JobDescription	Text	

Design of tblEmployeeJob (this is the table that connects the tblEmployee table and the tblJob table together).

Field Name	Data Type	Description
EmployeeID	Text	Foreign key from tblEmployee
JobCode	Text	Foreign key from tblJob
DateHired	Date/Time	

There should be a one-to-many relationship between tblEmployee and tblEmployeeJOB via the common field EmployeeID and a one-to-many relationship between tblJob and tblEmployeeJOB via the common field JobCode.

In class exercise: build the above database as specified by the above.

Another exercise: A student takes many classes and a class can have many students. A student has a StudentID that is a key, a first and last name and a gender (yes for male and no for female). A class has an classID that is a key and a course name. Design a database to represent this many-to-many relationship.