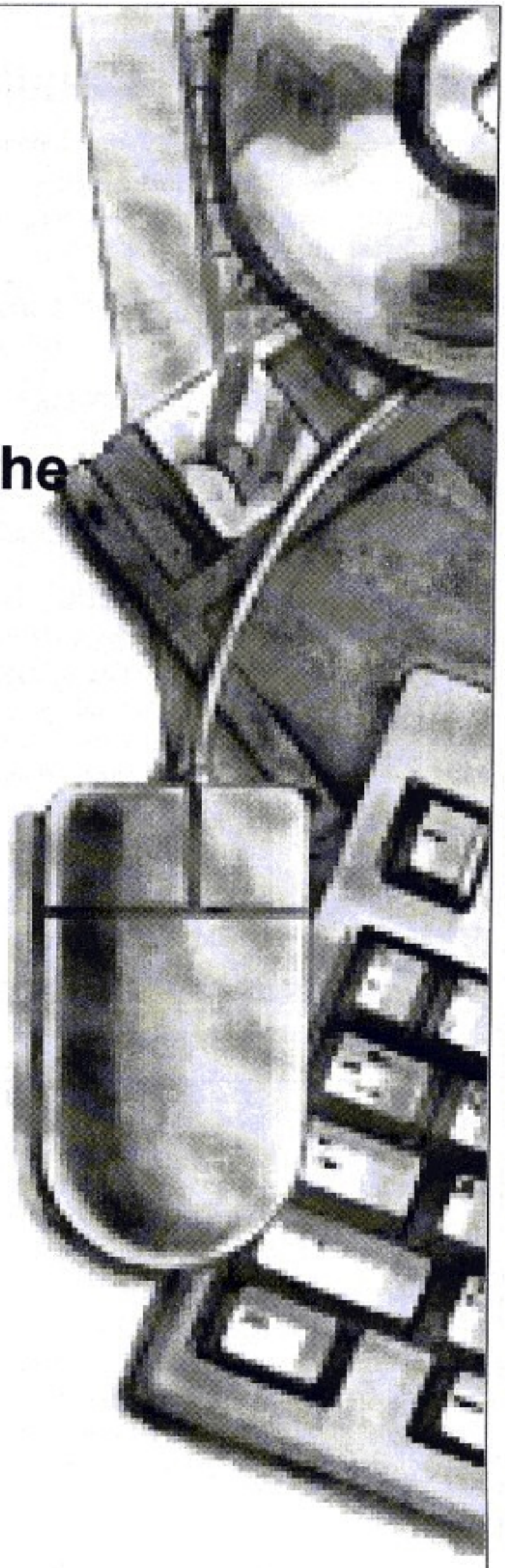
Module 3: Completing the Data Model



Performance Objectives

In this module, you gain confidence in your ability to:

- Use normal forms to refine the data model
- Establish table relationships
- Test and document the data model

Module Skills and Their Importance

After completing this module, you will be able to:

- Use normal forms to refine the data model
  You can use the normal forms or rules to achieve the optimum structure for the database.
- Establish table relationships
  You can store data more efficiently and optimize the performance of your database.
- Test and document the data model
  You can verify performance and ensure that your design is the optimum for your objectives. You also provide documentation for future users and administrators of your database.

What You Will Accomplish

The skills learned in this module prepare you to refine a data model using First through Fifth Normal Form. You learn the skills needed to establish relationships between tables in the database. You learn the importance of testing the data model for accuracy and providing thorough documentation.

Using Normal Forms to Refine the Data Model

After you develop a rough draft of your relational database, you might want to reorganize the structure. The preferred way to organize your data is *normalization*. To normalize your data, you must know the items you are tracking and how the data will be used. The goals of normalization are to enter a minimum amount of data, avoid duplicating data, and ensure the integrity of the data.

NEW TERM

normalization The process of organizing data into one or more tables.

The normal forms are tests that your database must pass. Each normal form deals with a specific design problem that could compromise the integrity of your database design. The first three normal forms were defined in the early 1970's by the pioneering database theoretician, E.F. Codd. Since that time more advanced levels of normalization have been defined. The higher normal forms are more rigorous tests of your database design.

First Normal Form

A table in *First Normal Form* means that each field in the table contains the smallest logical value, and that for each record, there should be only one entry for each type of information. No fields repeat the same type of information as other fields. In other words, your table should contain no repeating groups of fields like Month 1, Month 2, Month 3, for example.

NEW TERM

First Normal Form When all fields contain the smallest logical value and do not repeat the same type of information in multiple fields.

When a single attribute (field) of a table can contain multiple entries for each record of the table, the temptation is to create more than one column or field for that attribute, or to repeat the table records enough times to list each entry in a single column. Either solution causes data entry and retrieval problems. This is discussed in many-to-many relationships at the end of this module.

If the same type of information is listed in many different columns, it makes it difficult to search for a particular piece of information because it might be listed in one of several different fields. Sorting and calculating are difficult as well, since you might have to combine data in different columns to create an accurate calculation.

■ Example of a First Normal Form violation

Suppose our airline wanted to keep track of regular maintenance on the aircraft on a monthly basis, and designed the following table.

Maintenance			
Serial#	**Month1**	**Month2**	**Month3**
4123	oil change	none	oil change
3872	oil change	oil change	altimeter check
5991	oil change, fuel filter	new wing	none

The person who built this table envisioned a list of airplanes, with the maintenance records for each airplane attached. This works fine in a filing cabinet, with a folder for each airplane, but as a database table it has some problems. To find out when oil changes were performed on all airplanes, the user must look through all of the Month# fields for oil change entries. To find which aircraft received a new wing, a query would have to search every column of the table for the new wing maintenance, a difficult query to formulate. Finally, storing two separate maintenance items in one field, as in the last row, is a disorganized way to store data. Whenever you search for oil changes, the filter entry is also displayed.

■ Applying First Normal Form

It is advisable to design the table so that each field deals with a unique category of information, making the table longer and narrower. This means that some fields that weren't repeating before will repeat, for example, Serial#. This may mean you need to designate a different primary key. If designating a different primary key causes too many other repeated entries in the table, dividing the data into two separate tables might be necessary.

In First Normal Form, the preceding table would probably end up looking like the following table.

Maintenance			
Job#	**Serial#**	**Job Done**	**Date of Job**
101	4123	oil change	1/25
102	3872	oil change	1/27
103	5991	fuel filter	1/31
104	5991	oil change	2/12
105	5991	new wing	2/27
106	3872	altimeter check	3/1
107	5991	new filter	3/15

With this table design, finding the plane with the new wing means searching in only the Job Done column. The fuel filter for airplane #5991 is now a separate entry and will not show up in any searches for oil changes. Another benefit of this table design is that the "none" entries are eliminated. Only meaningful data is entered. If no maintenance was performed, you don't have to update the database.

Splitting the data into separate tables isn't necessary in this case, since only the Serial# is repeated, and that is necessary for identification. If this table also listed airplane makes, names, or other non-maintenance information, it would be appropriate to store that information in a separate table where it would not be repeated.

Second Normal Form

A table in *Second Normal Form* means that each field in the table relates directly to the primary key field, and to the entire primary key if the key consists of a combination of fields.

NEW TERM

Second Normal Form When each field in the table relates directly to the primary key field of the table.

Data that doesn't relate to the primary key (the topic of the table) at all, doesn't belong in that table. If the data relates to only part of a combined primary key, the data must be repeated every time the field it relates to is repeated. This means that the information is entered unnecessarily, and if this information needs to be edited, it must be edited in every place. If the data doesn't relate to the primary key at all, it belongs in a different table.

■ Example of a Second Normal Form violation

If the preceding Maintenance table contained more information about each aircraft than just the Serial#, the person who enters the maintenance information has to repeat a lot of aircraft-specific information.

Maintenance					
Job#	**Serial#**	**Make**	**Model**	**Job Done**	**Date**
101	4123	Boeg	7002	oil change	1/25
102	3872	Boeg	7003	oil change	1/27
103	5991	ABus	E300	filter	1/31
104	5991	ABus	E300	oil change	2/12
105	5991	ABus	E300	new wing	2/27
106	3872	Boeg	7003	altimeter check	3/1
107	5991	ABus	E300	new filter	3/15

Every time maintenance is performed on plane #5991, someone must enter the fact that the plane is an ABus E300. Sometimes people try to avoid this repetition by leaving the repeated information blank for adjacent lines, for example, leaving out "ABus" and "E300" from the fourth and fifth data lines. This strategy would work only if the table was always sorted by Serial#, so that all of the 5991 entries would be grouped. As soon as the table is sorted into a different order, though, for example, by the Job Done field, the whole system becomes disorganized.

■ Applying Second Normal Form

Split the table up so that the repeated information (Make and Model) can be listed in a separate table just once. The preceding table should be split into two tables: one for the aircraft information, listing Serial#, Make, Model, and any other aircraft-specific data, and a second table listing maintenance-related information such as the Serial# (to link with the first table), the Job Done, and the date.

Third Normal Form

In *Third Normal Form*, non-key fields should not be derived from other non-key fields. To change your table into Third Normal Form, you eliminate any fields that could be derived or calculated from other existing fields.

NEW TERM

Third Normal Form When none of the fields in the table are derived from fields other than the primary key field(s).

In some sources, Third Normal Form might be described similarly to Second Normal Form: fields in a table should describe the key field of the table and not another field. In other words, if you can calculate the contents of one field based on another field, then it is an elaboration on the original field, which is probably not the key field of the table.

If one field can be determined from other fields in the table, it is inefficient for someone to enter that data; better to have the computer calculate it or look it up. Having data that must be calculated by hand from another part of the table also causes problems when updating information. Not only do you have to update the base data, but then you have to update the dependent information as well.

■ Example of a Third Normal Form violation

Our airline might have the following ticketing information table to track money brought in through ticket sales.

Ticketing			
Flight#	**Ticket Price**	**Tax**	**Total Price**
987	$200	$ 16.20	$ 216.20
765	$175	$ 14.18	$ 189.18
707	$45	$ 3.65	$ 48.65
007	$372	$ 30.13	$ 402.13

Here, the Tax and Total Price fields can be calculated from the ticket price. The way the table is set up now, the user must calculate the tax and the total every time a ticket is entered. If a ticket price is changed, someone must recalculate each entry to update the other fields.

■ Applying Third Normal Form

Remove the fields that can be derived from other fields. Use a query or a report in your database program to calculate those values, instead of storing them permanently with the rest of the data. You save storage space as well as data-entry time.

It's important not to take this rule of normalization to an extreme. Addresses are the most common example. It is possible to figure out the city and state by simply knowing the zip code, however, you would have to create separate zip code lookup tables to "calculate" this information whenever a full address is needed. This is really only productive for very large databases; where the space saved by not listing every city and state offsets the space taken up by listing the city and state of every zip code in the United States.

Using Higher Normal Forms

It is acceptable to stop the normalization process at Third Normal Form. You have most likely eliminated the worst sources of database design error. There are some advantages to continuing the normalization process with higher normal forms. For example, you can sometimes pick up a structural flaw that was missed in previous normalization. When using higher normal forms, remember that in most applications normalization is less than complete.

Fourth Normal Form

For Fourth Normal Form, there must be no independent one-to-many relationships between the primary key and other fields in the table. A problem with Fourth Normal Form is often characterized by blank spots in fields. In other words, there might be a field in the table that will never store data because the field is not relevant for some items being tracked by the table. It is also often characterized by table names that have "and" in the title, where someone tried to combine two separate topics into one table.

The following table was designed to list which equipment goes in each hangar, and which technicians are permanently assigned to each hangar.

Hangar Equipment & Technicians				
Hangar	Technician	Rating	Equip#	Equipment
12	Mac Couteau	A1	RF756	Refueling truck
12			AW932	Arc welder
12			FL109	Forklift
14	Thomas Hu	A2	RF621	Refueling truck
14	Louise May	A1	FL902	Forklift
14	Vern Smith	A3		

At first glance, this table lists technicians and the equipment they use. But on further examination, you see that three pieces of equipment are listed for Hangar 12, but only one permanent technician. If the technician uses all of the equipment, the technician should be listed next to each piece. A technician is listed for Hangar 14, but no equipment is listed next to his name.

Each hangar can have more than one technician, and more than one piece of equipment, but those two facts don't necessarily have anything to do with one another. One technician might use more than one of the pieces of equipment in a hangar. The result is blank spaces and repeated hangar numbers, which we might have used as a primary key.

■ Applying Fourth Normal Form

You might split the table into multiple tables, for example, one table listing technicians for each hangar, and another table listing equipment for each hangar. Or, you might list the hangar number for each piece of equipment in a master equipment list, and list the hangar number for each technician in a list of technicians.

Fifth Normal Form

Fifth Normal Form eliminates any non-key field repetition from the tables. No information should be repeated anywhere unless the information is part of a primary key field, or is a foreign key and therefore is necessary for table relationships. Putting a table into Fifth Normal Form usually involves splitting a large table into two or more tables with fewer fields. By putting your tables into Fifth Normal Form, data can be updated in just one place, without searching for other occurrences of the information.

Other Details to Check

Here are some other points to check in your data model to prevent future problems:

■ Field names

1. Fields that refer to the same thing should be named the same in all tables.
   If an employee field is EMP-ID in one table, the same employee field should be called EMP-ID in other tables. This makes it obvious that the two fields contain the same data.

2. Fields that do not refer to the same thing should not have the same name.
   A QTY field that lists the quantity of airplane parts in stock should be differentiated from a QTY field that refers to the quantity of pilot uniforms ordered, for example QTYpts and QTYuni.

■ Redundant information

1. The same information stored in different ways.
   A common example is a table containing an age field and a birth date field. If you know a person's birth date, you can calculate the person's age (see Third Normal Form). It is sometimes difficult to recognize this type of redundant information because of the different formats: age might be a number, while birth date might be a date-formatted field.

2. Mutually exclusive information.
   For example, an airline employee can choose from three health plans. You might be tempted to show this in a table by making a field for each health plan, and in each person's record you might place an X in the appropriate field. This causes a First Normal Form violation because you have three fields referring to one thing: the person's health plan. Since a person is only in one health plan (meaning that health plans are mutually exclusive), it is more efficient to have one column in the table in which you simply enter which plan was chosen.

■ Adjectives in table names

1. Table names that contain the same noun with different descriptive adjectives can often be combined into one table. This makes it easier to search and query.

Establishing Table Relationships

A database becomes a relational database when you designate relationships between the different tables of the database. Each table might track different things, but the things that are tracked relate to one another. For example, in addition to the table tracking employees, you might have a table listing departments, for example, reservations, maintenance, and flight crew. Departments and employees are related because the employees are the people who staff the departments.

There are three types of relationships between tables: *one-to-one*, *one-to-many*, and *many-to-many*.

NEW TERMS

one-to-one relationship A relationship between two tables in which each record in table A is related to only one record in table B, and each record in table B relates to only one record in table A.

one-to-many relationship A relationship between two tables in which each record in table A relates to multiple records in table B. Each record in table B only relates to one record in table A.

many-to-many relationship A relationship between two tables in which each record in table A is related to multiple records in table B, and vice versa.

■ One-to-one relationships

In a one-to-one relationship, one item in the first (or "left") table relates to just one item in the second (or "right") table. Often two tables with a one-to-one relationship can be combined into a single larger table. They might be kept as two separate tables for convenience or security reasons. Some information might be looked up more often than other information, or some data might be restricted.

An example of a one-to-one relationship between two tables could be as follows. Suppose your airline maintains a table of employees' home addresses, in addition to the Employees table, to more conveniently schedule flight destinations.

Employee Home Addresses				
Name	**Address**	**City**	**State**	**Zip**
Joel Ripple	123 Main St.	Auburn	WA	98002
Lucy Redhawk	321 Main St.	Remington	IN	47977

This table and the Employees table would have a one-to-one relationship. One employee in the Employees table corresponds to one home address, and one address in the Employee Home Addresses table corresponds to one employee.

■ One-to-many relationships

In a one-to-many relationship, one entry in the first (or "left") table relates to many entries in the second (or "right") table.

The following Departments and Employees tables are an example of a one-to-many relationship. One department has many employees, but one employee works in only one department.

Departments	
Dept Code	**Dept Name**
MT	Maintenance
FC	Flight Crews
TR	Ticketing & Reservations

Employees				
Name	**Job Title**	**Dept**	**Hire Date**	**Salary**
Joel Ripple	engineer	MT	4/1/80	$60, 000
Lucy Redhawk	pilot	FC	7/4/90	$75,000

The Dept field in the Employees table designates the employees that work in each department, thus relating the information in the two tables.

■ Many-to-many relationships

In a many-to-many relationship, one entry in the first table relates to many entries in the second table, but the reverse is true as well: one entry in the second table also relates to many entries in the first table. In our airline example, you might have a table listing airplanes, and another table listing pilots. Each plane might be flown by several different pilots over time, and each pilot might fly many different planes. These two tables have a many-to-many relationship.

Pilots			
PilotID	**Name**	**Rank**	**Flight Hrs**
RJ	Roy Jones	2nd Class	150
LR	Lucy Redhawk	1st Class	275
GH	Gina Herrero	1st Class	235

Airplanes		
Serial#	**Make**	**Model**
4123	Boeg	7002
3872	Boeg	7003
5991	ABus	E300

Many-to-many relationships can cause problems in relational databases. In order for the database to effectively answer the question, "Which pilots fly which planes?", you must limit the query to searching for pilots for one specific plane, or planes flown by one specific pilot. If you don't limit the question and ask for all the planes that all the pilots fly, the result is "they all fly all the planes," which isn't very helpful.

One solution would be to add a field to the Pilots table listing the airplanes they fly. Since a pilot flies many planes, you need to have multiple Airplane fields, one for each plane flown for each pilot's record, for example, Plane1, Plane2, and Plane3.

Pilots					
PilotID	**Name**	**Rank**	**Plane1**	**Plane2**	**Plane3**
RJ	Roy Jones	2nd Class	4123		
LR	Lucy Redhawk	1st Class	5991	4123	
GH	Gina Herrero	1st Class	4123	5991	3872

If you added only one new Serial# field, you would have to list the different airplanes on multiple rows of the table, forcing you to repeat all the other information on each pilot in the multiple rows as well.

Pilots				
PilotID	**Name**	**Rank**	**Hrs**	**Serial#**
RJ	Roy Jones	2nd Class	150	4123
LR	Lucy Redhawk	1st Class	275	5991
LR	Lucy Redhawk	1st Class	275	4123
GH	Gina Herrero	1st Class	235	4123
GH	Gina Herrero	1st Class	235	5991
GH	Gina Herrero	1st Class	235	3872

The first solution, multiple Serial# fields in the Pilots table, makes querying the table to locate pilots for a specific airplane very difficult, since the query must search many columns for the airplane number. It also causes problems when you encounter one pilot who flies four planes, because you must restructure your table to add a new column, and restructure any queries you had previously created to take the new column into account. Also, any pilots who fly fewer planes have blank space in their records, which adds to the size of your database.

The second solution is inefficient because of the repetitive pilot information. This adds unnecessary bulk to a database, and is difficult for the person completing the data entry. Also, you no longer have a simple list of individual pilots to refer to when you just need to look up a name.

In this situation, you can't just leave out the repeated information in subsequent rows and assume people know that it's "supposed to be" the same as the previous row. For example, you can't leave Gina Herrero's initials, name, and rank out of the second and third lines of information about her.

Pilots				
PilotID	**Name**	**Rank**	**Hrs**	**Serial#**
RJ	Roy Jones	2nd Class	150	4123
LR	Lucy Redhawk	1st Class	275	5991
				4123
GH	Gina Herrero	1st Class	235	4123
				5991
				3872

As soon as this table is sorted into a different order, for example, by Serial# instead of by PilotID, the entries with the blanks in them no longer make sense.

■ The Solution

To solve this problem, you must create a third table to link the original two. This table is called a bridge table, or a weak entity. This table contains one entry for each possible combination of pilot and plane, and breaks the many-to-many relationship into two one-to-many relationships. One airplane in the Airplanes table has many listings in the Fliers table, and one pilot in the Pilots table has many listings in the Fliers table.

Fliers	
Serial#	**Pilot ID**
4123	LR
4123	GH
4123	RJ
3872	GH
5991	GH
5991	LR

This table also gives the database a single table of valid answers to the question, "Which pilot flies which planes?" It provides a comprehensive list of all valid combinations. Determining a particular combination such as, "Does Lucy Redhawk fly the ABus?" is now a fairly simple question. All you do is determine whether such a record exists in this table.

Creating this table involves some repetition of pilots and airplanes, so the Fliers table refers to plane numbers and pilot codes. All other information about planes and pilots would be looked up in the original Airplanes table and Pilots table.

Testing and Documenting the Data Model

Testing is an important step in the design process since it brings you back to the initial goal of the database.

When possible, gather actual data from database users as your test data. If the database users can test your database for you, so much the better, provided that they know that it is indeed a test and their comments are requested. User comments can stem from their personal knowledge of what helps them to get their jobs done, and user testing is a true test of the system. Often during tests, you discover additional areas that must be incorporated, or areas that are unnecessary. When possible, do an imaginary run-through with test data before you begin the computer part of the work.

Document your database as you work on it, as well as when it is finished. Add to your database description and write down instructions and assumptions for the procedures performed by your database. Write brief descriptions of each report, form, or view, explaining what it is for, what it does, or what it looks like. That way, if you have to modify your database weeks, months, or years later, you have documentation to refer to that can remind you of how you created the database.

Documentation is also useful to the database users while they learn to use the system, and to anyone who might have to modify the system if you aren't available.

Creating and Using a Data Dictionary

You have decided what tables you want in your database, what fields you want in each table, and how you want to relate the tables. You've even decided which field in each table is the primary key for that table.

Writing down what you've come up with so far forms the basis of your *data dictionary*. The data dictionary is your blueprint for the database system you eventually build. The exact structure of the data dictionary is not important so long as the dictionary is legible and understandable; what is important is that all the pertinent information is included.

NEW TERM

data dictionary A systematic listing of the tables, fields, and other details of a database plan.

Your data dictionary should list all the tables that the database includes, and all the fields in each table. Mark which fields are primary keys, and how tables relate to one another. You might want to mark which fields are foreign keys, as well.

Along with the names of the fields and tables, include any special information about that table or field, such as any assumptions about how data is entered into the item in question, and in what format the data appears. Think about the following things for each field. Any answers to these questions must be noted with that field in your data dictionary.

1. What type of data goes in this field? Text? Numbers?
2. Will this field be calculated automatically? If so, should it be eliminated from the storage part of the database?
   Refer to Third Normal Form on page 37.
3. Should there be a default entry for this field?
4. Should there be any limitations on what can be entered in this field? On how the data is entered?

A partial sample data dictionary might look like the following:

Pilot Table			
Pilot Initials	Text	2 chars	KEY
Pilot Name	Text	50 chars	Full Name
Ranking	Text	10 chars	default: second class
Hours	Number		not less than 20
Airplanes Table			
Airplane#	Text	5 chars	KEY
Make	Text	15 chars	required field
Model	Text	10 chars	required field
Seats	Number		default based on Make & Model

It might be convenient to include your data dictionary as a table in your database. The data dictionary table doesn't link with any other tables in the database or affect any of the other data, but it can be used as a reference.

A data dictionary table in your database might be organized as a single table with the following fields. Each record is one field from your database design.

Field Name	Table	Key?	Data Type	Size	Details

You can use the dictionary as a blueprint for your own database setup work, or for someone else to follow in creating the rest of your database for you. You can also make it available to future users of your database and use it to review the goals and tasks in your statement of purpose to ensure expectations are fulfilled.

Moving the Idea to the Computer

You complete your database design, test it (at least mentally) to make sure nothing obvious has been left out, and start writing your preliminary data dictionary. The next step is to start the computer and begin creating the actual database. You first create the tables that you planned and outlined in your data dictionary, and then create the data entry forms and reports for printing. You may want to add some programmed procedures, customized menus, or buttons to help guide your database users through their tasks. The exact procedures you use to do these things depends on the DBMS software that you've chosen. With such a detailed plan to begin with, setting up the computer side of your database is much easier.

Review Checklist

You must be able to complete the following tasks on your own. As you review these topics, place a check mark next to those you can do. If you do not feel confident about an item on the following list, let your instructor know.

- ❑ Use normal forms to refine the data model
- ❑ Establish table relationships
- ❑ Test and document the data model