PRINCIPLES OF EFFECTIVE DATABASE DESIGN

Table of Contents

Page

Last updated: July 29, 2004

LESSON 1: WHAT IS A DATABASE?

OVERVIEW

A database is simply a computerized record-keeping system; a set of structured, interrelated data. Databases are maintained to make information available on demand to one or more users or applications. A database is also something created using a database software application or tool. It differs from a spreadsheet primarily in terms of purpose (although there is an increasing blur between these two types of applications, due to the continual refinement of both database and spreadsheet application software).

Databases can store large amounts of text data and typically display 'one record's worth' at a time. Spreadsheets, by contrast, are used to perform calculations quickly and display large amounts of numeric data, although many people use them to keep lists of information. When debating on whether to use a database or spreadsheet for a particular task, it's best to examine the purpose of your task. Are you dealing with lists of numeric data requiring calculations or do you need to track multiple occurrences of data such as names and addresses? If the latter is the case, perhaps a database is what you need.

A database can be small, containing only a few types of data such as Name, Rank and Serial Number, or enormous such as those used by State and Federal Agencies or large corporations. Regardless of size, all databases share certain basic qualities. For instance, every database:

- Had to be 'built'.
- Required some degree of forethought (i.e., design).
- Has at least two table components for the purpose of storing data.
- Allows for the creation, deletion, or reading of table data via records.

Beyond these basic similarities, databases can range tremendously in terms of purpose, design, and interface.

Some examples of databases in wide use today are:

- Airline reservations
- University student records
- Human Resource databases
- Product Inventories
- Medical records
- Computerized library

NOTES

- Personal address book, appointment calendar
- Computerized map
- Expert system for medication interaction advice

In fact, the dissimilarities are the very reasons why we address issues such as *how to design a database* or *which database tool should I use*. These issues are important to us because we all wish to do it 'right' the first time. The best strategy for gaining experience in creating databases, however, is just to create one!

There are definite phases in developing a database that you should consider. The first stage is the **Requirements** stage where content, usage and performance are scoped out. You will want to interview users to find out their needs, to get a clear picture of the purpose of the database, what kind of data needs to be stored and for how long, etc.

The second phase is the **Conceptual design** stage where you will draw a schematic of your database design. In other words, you will create a model showing the different field names and how they relate to each other.

LESSON 2: IDENTIFYING REQUIREMENTS

OVERVIEW

We will create a database to help the Maine Widget Company. Maine Widget has lots of different products that it sells to many different companies in Maine and throughout the Northeast. They are a high volume supplier to other woodworking companies. Most of their products are parts which are ordered by furniture companies who use them to make finished pieces. Maine Widget also buys their raw lumber from a handful of wood harvesters, mostly in-state operations, although they also buy wood from New Brunswick, Canada. They have enjoyed a lot of growth over the years.

Maine Widget called us to help with a problem. They are experiencing difficulties tracking their inventory and what they sell to customers. They would like us to create reports from a database so that they can tell who is buying what product, and what raw material they need to order to supply the demand for widgets.

GOALS:

1. Create a database to track sales of widgets by customer.
2. Produce reports to give Maine Widget status on all customer accounts.

STEPS:

1. Brainstorm a list of questions about the problem. The answers to these questions should help you define the requirements for the database.

    a. What's the purpose of the database?
    b. What subjects must be covered?
    c. How are the subjects related to each other?
    d. What descriptive information about each subject must be captured?
    e. Who will be using the database?
    f. What kinds of reports must be generated?

    Detailed lists and answers to these questions will help define the data to be stored in the database.

2. Start from the desired *output* and work backwards. Design reports that will give the information needed to answer the questions thought of in step #1.

3. Design the screen layouts for entering information to your database.

NOTES

EXERCISE 1:

1) Make a list of the questions that you think Maine Widget will need answers for in order to straighten out their problem: they need to know which customers are buying what products.

2) Design a report for customer information, product information, and sales by customer on a word processor of your choice or on a piece of paper. Think of all the information you would like to display and that Maine Widget needs in order to make decisions about the questions you thought of in step #1.

NOTES

LESSON 3: CONCEPTUAL DESIGN - THE STORYBOARD

OVERVIEW

Using the questions, reports, and screens you have, compose a *storyboard* describing the problem. This is a short narrative explaining Maine Widget's problem. When you are done, you will want to:

1. Italicize the *adjectives*.
2. Underline the nouns.
3. Bold the **verbs**.

The nouns will become tables. The verbs become the relationships between the tables. The adjectives will become fields.

Arrange words in the following order: *adjectives (fields)*, nouns (tables), and **verbs (relationships)**.

EXERCISE 2:

1) Compose a storyboard and identify *adjectives*, nouns, and **verbs**. Complete the table below.

Adjectives (fields)	Nouns (tables)	**Verbs (Joins)**

NOTES

LESSON 4: CONCEPTUAL DESIGN - CONSTRUCTING TABLES

OVERVIEW

Now that you've got your questions asked, reports and screens designed, and a storyboard, the next step is to construct tables which will hold pieces of the information you need -- fields.

♦ Fields

Using the questions asked, reports, screens, and storyboard list the field names you will need.

♦ Primary Key

Primary keys are fields that identify each record uniquely in the database. Each primary key must be chosen for its ability to be unique for every record that will eventually populate the table. For example, in a **Person** table, one might first designate the field, *Last Name* as the Primary key, because people are usually referred to by their last names, instead of other descriptors such as hair color or age. However, in a database (as in real life) it is very likely that more than one person will share the same last name. Instead let's use the Social Security Number to uniquely represent a person. Unfortunately, in recent years, this too has fallen short in its ability to be unique due to the 'recycling' of Social Security Numbers within the fifty states.

To resolve this issue, we may wish to create an artificial primary key, essentially a computer generated numbering system that will automatically enter a unique number for each record. In the database world, such primary key fields are oftentimes referred to as *artificial or counter primary keys.* Once you decide to create an artificial primary key, a new attribute must be created and should be given a name such as *Person ID,* for ease of recognition.

Another option is to use more than one field as a primary key. This is called a **compound key.** For example, the combination of the field, Social Security Number and the field, Date of Birth would uniquely identify a person.

Generally speaking, it is best to find a *natural* (not artificial) field that you can always be assured is unique, and designate *it* as the primary key. But if you cannot be 100% certain that the natural field will contain unique information for each record over time, then an artificial key is the best solution.

♦ Normalization of Data

One of the most important aspects of relational database design theory is *Data Normalization*.

Benefits when you 'normalize' your data:

- Eliminate redundant information.
- Make searches for information efficient and easy to build.
- Make updating, adding, and deleting information easy and accurate.
- Create an overall design that will be easy to maintain over time.

Cost when you 'normalize' your data:

- Slows down application.

Unfortunately, an in-depth discussion of all the normal forms is far beyond the scope of this class. However, this is a good time to gain a basic understanding of at least the first three normal forms. These are the most critical in nonscientific information management systems, the basic information database.

♦ First Normal Form

First Normal Form is when each field contains the smallest meaningful value and the table does not contain repeating groups of fields. For instance, a field called *Name*, which contains both the last name and first name can be further broken down into two more succinct fields, *Last Name* and *First Name*.

EXERCISE 3:

1) Using the questions asked, reports, and storyboard list the field names you will need.

2) Identify or create primary keys.

3) Using the list of fields, separate them into tables so that they are in first normal form.

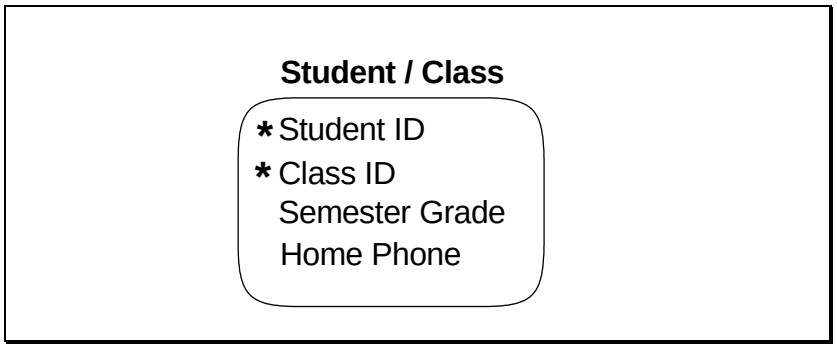
NOTES

♦ Second Normal Form

Second Normal Form is important only when you're dealing with *Compound Keys*. In this situation, each field must depend on the **entire** Key including all fields which comprise the Compound Key.

For example, the following Compound Key consists of *Student ID* and *Class ID*. This is because each record pertains to a particular Student's School Attendance, and both fields are necessary to uniquely identify each record. In other words, each student can attend more than one class and each class can have more than one student. We have to 'match up' the individual student with the individual class.

Let's examine the table below:



Does *Home Phone* pertain to the Whole Key? The answer is **No**. *Home Phone* pertains only to the *Student ID*, not the *Student ID* and the *Class ID*. Therefore, *Home Phone* should not be in the **Student / Class** table.

EXERCISE 4:

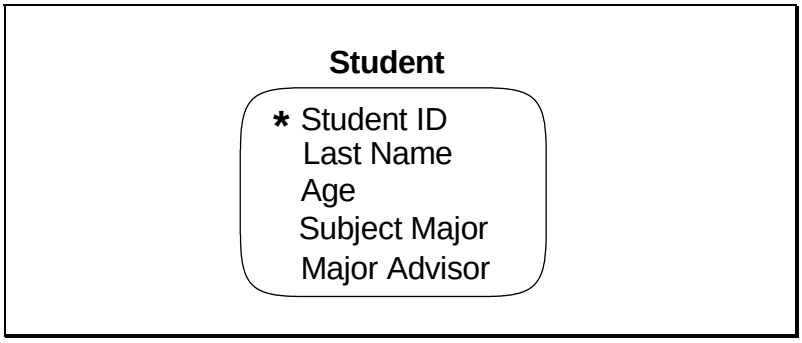
1) Are there any compound keys in our work for Maine Widget? If so, identify them and make sure they're in 2nd normal form.

NOTES

♦ Third Normal Form

Third Normal Form involves examining each field in your design and asking the question, *"Is this field dependent on the key?*" For example, if you plan to use the *Social Security Number* as the key field, then every other field in the **Person** table should be dependent on the *Social Security Number*. Such fields might include, *hair color* or *eye color*, because each **Person** (identified by his Social Security Number) has his own hair color and eye color.

Third Normal Form applies to tables with *single* key fields. To check for Third Normal Form, ask yourself, *"Does every other non-key field describe the key field, or does it describe another field?"* Let's examine the following table:



In this example, *Major Advisor* is dependent on *Subject Major*, not *Student ID*. Therefore, we can conclude that the **Student** entity is not in Third Normal Form. To remedy this situation, we could move both fields, *Subject Major* and *Major Advisor* to a new table called **Major**.

EXERCISE 5:

1) Are your tables in third normal form? If not, make sure they're in third normal form.

NOTES

◆ The "Myth"

Every good database is at least third normal form.

Now that we've discussed data normalization, let's do a 'reality check'.

It would be comforting to believe that once you achieve third normal form with your design, you're home free. Unfortunately, this is almost never true. Why? Because a perfectly normalized database may not be practical in terms of screen design and performance and also because your database software may not react well to a true third normal form design. However, you should *strive* to attain third normal form with your design, then selectively *'denormalize'* to achieve your required performance standards.

EXERCISE 6:

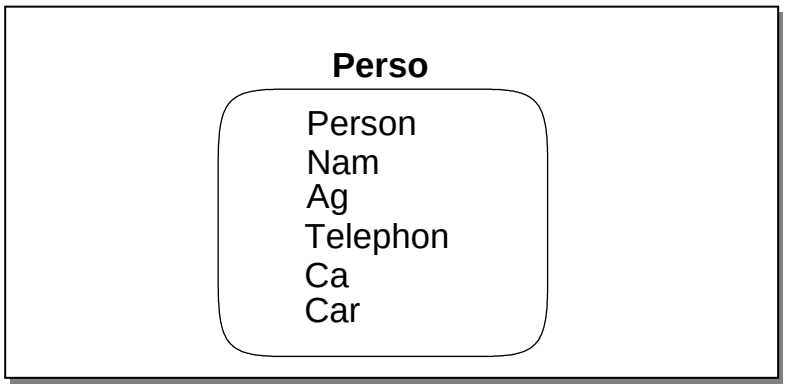
1) Think about your needs, and 'denormalize' the tables if necessary.

♦ A Common Sense Approach

After analyzing the recommended data normalization methods to achieve third normal form, this is the last check to make sure your database is designed properly.
First, make a list of tables and fields. Once you've accomplished this, go back and review your decisions. Try a substitution game using this phrase:

"The *Field Name* of the **Table Name** is"

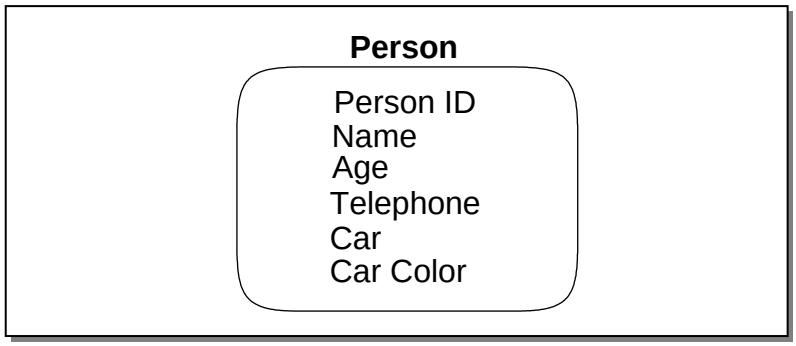
and see if it makes sense. For example: In a **Person** table, you may have assigned a field called *Car Color*. Using substitution, ask yourself if this phrase makes good sense: "The *Car Color* of the **Person** is...



Attribute Example

Clearly, the answer is "No" because car color is a way to describe a car, not a person. Therefore, you probably need a **Car** table as well.

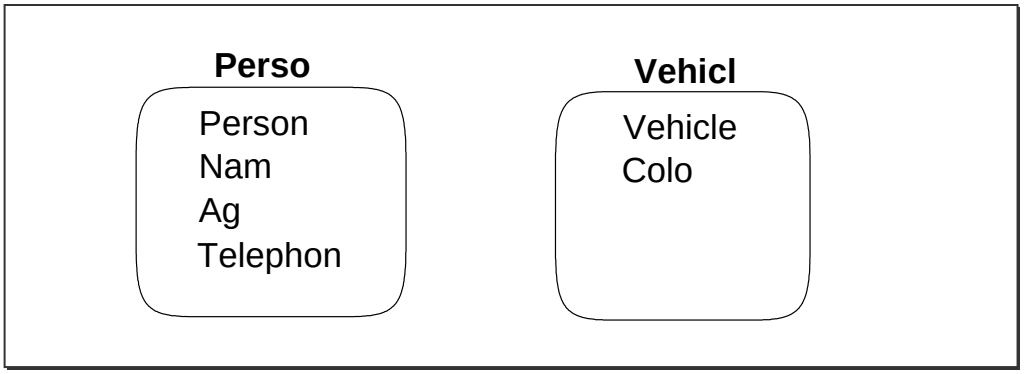
A similar check is to go through your field list and ask yourself, does this field describe the table or another field. Let's revisit our original example:



Attribute Example

Question: " Does *Car Color* describe the table **Person** or the field *Car*?

Answer: Car Color describes **Car**, not **Person**. So, make a second table called **Car** [or should we use the more generic noun, **Vehicle,** since some employees may have trucks or RV's...?] and move the field *Color* to the **Vehicle** table.



Attribute Example

EXERCISE 7:

1) In the Maine Widget database, look at the tables and ask the **"The *Field Name* of the *Table Name* is..."**question for each field.

NOTES

LESSON 5: CONCEPTUAL DESIGN - FIELD PROPERTIES

♦ Naming Conventions

All database components (tables, fields, queries, forms, and reports) should be given *meaningful* names. Granted, it is more time consuming than just arbitrarily assigning names such as "Table1", "Query5" or "Report3a" which is usually done automatically by the database tool you use to design your database. The few extra seconds it takes to assign meaningful names is well worth the effort!

You may wish to use a singular noun to name tables. For example, use "Person" instead of "People". This makes it easier to use sentences such as "Each **Person** may own one or more **Vehicle**s".

Do not give objects names you have to change. For example, if you name a screen, "**Training for 1996,"** you may have to change the name of the form after December 31, 1996. Name the form, "**Training for This Year."**

♦ Data Types and Field Sizes

OVERVIEW

Good News! Selecting a data type for each field is a relatively easy task, as the newest database products such as Lotus Approach and MSAccess have done most of the work for you. Certain data types have become industry standards as well, so you'll find many of the same data types in most database products. For example, Lotus Approach has a separate data type for each of the following commonly used data types**:** *Date, Time, Text,* and *Memo.* Microsoft Access varies only slightly in its offering, combining Date and Time into Date/Time.

Other Issues:

1. Use **Text** when none of the other types will work.

2. The data type **memo** is good for descriptions and comments. On an 8 ½ by 11 inch piece of paper about 80 characters can fit across the page horizontally (in portrait mode). If you consider left and right margins, you actually have 60 to 65 characters. The same thing is true with your screen. In a normal font style and size, about 80 characters fit across the screen. Again, if you consider margins, the number of characters decreases to between 60 and 65. If your data can be held in 65 characters or less, use **text**, otherwise use **memo.**

3. Use **numeric** when you need to perform mathematical calculations. Think about the data carefully. Sometimes the **numeric** data type is very tempting to use, however it is not appropriate. Even though zip codes and phone numbers contain numbers, you will never perform any type of mathematical operation on them. Using the data type, **numeric,** for zip codes and phone numbers actually causes problems. The zip code 04330 would be changed to 4330 because the leading zero would be dropped. The phone number 626-1234 may be changed to -608 (because 626 minus 1,234 is equal to negative 608) or an error message may appear. Basically, if you're unsure as to whether a particular data item contains 'only numbers' or 'numbers and letters', select **text**. In Lotus Approach, the default size for a **numeric** data type is 10.2. This means 10 numbers can be stored to the left of the decimal, and 2 numbers to the right of the decimal. MSAccess uses **byte, integer,** etc. for **number** field sizes. See help in MSAccess for further information.

4. Try to determine the maximum length of each text field, but don't quibble over a 'character or two'. For instance, if you're not sure whether to allocate 23 or 24 spaces for a particular name, go ahead and select 25 or 30.

5. The field size for **calculated, memo, Boolean, pictureplus, time,** and **date** data types are fixed in Lotus Approach. In MSAccess, the fixed field sizes are **memo**, **date/time**, **OLE object** and **yes/no**.

(We go over this information in depth in our database I and II courses.)

NOTES

♦ Default Values

OVERVIEW

In some cases, it may be useful for you to have information entered in a field automatically. For instance, you might be constructing a database of names and addresses of coworkers. Since probably everyone lives in Maine, it would make sense to have the field *STATE* filled automatically by a default value of *ME*. It saves data entry time.

(We go over this information in depth in our database I and II courses.)

EXERCISE 8:

1) Name the tables and reports. For the fields in the tables you've created, determine the data type, field sizes, and default values you will need in your database.

♦ Data Validation

OVERVIEW

To ensure that data entry is correct, it is helpful to have the computer check for the right information in a field before the data is committed or written to the database. For instance, you might want to make sure that you write a constraint for the field *SALARY* if salary ranges are between $10,000 and $100,000. That way, no one could accidentally enter a salary of $200,000. Or, if a field is formatted for social security numbers, you might want to have the computer check to make sure that entries conform to 999-99-9999 where 9 stands for any digit and the hyphen stands for itself. In both cases, if what is entered is incorrect information, the computer will not allow the person to leave the field during data entry until the correct information is typed in.

EXERCISE 9:

1) Think about the fields in the tables for Maine Widget and decide if data validation is useful or even necessary in any field.

2) If so, write the formula or constraint.

♦ Formulas

OVERVIEW

It is relatively easy in most database packages to have a computed field. You can set up a formula to produce information in another field very easily. For instance, you might want to see a field *Product Total Cost*. If your database contains the fields *Unit Price* and *Quantity Sold*, you can multiply them together to produce *Product Total Cost.* Another example is to use a formula to count the number of records in your query, or as you add records during data entry, have the computer increment by an interval each time a record is entered.

(We go over this information in depth in our database II course.)

EXERCISE 10:

1) Construct the formulas for any computed fields in your tables.

NOTES

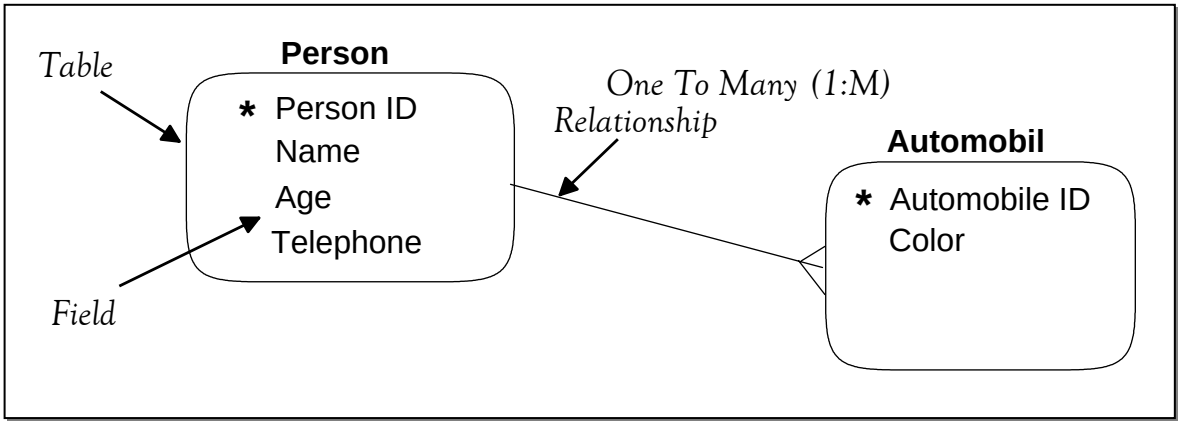
LESSON 6: CONCEPTUAL DESIGN - RELATIONSHIPS AND MODELING

♦ Relationship Modeling

OVERVIEW

Entity Relationship Modeling (E-R Modeling or E-R Diagramming) is the act of designing a relational database using our knowledge of data normalization. E-R Models can be drawn on paper or with the use of an automated drawing tool. There are many special software tools available for this task on the market today.

In the database industry, there are many methodologies for 'drawing' a relational database design, all which are valid and basically convey the same information. The following methodology is a simple one, which borrows heavily from the ORACLE Case Methodology.



Entity Relationship Model
**Primary Key*

Basic Concepts

- Each prospective table is an entity represented by a 'soft box'
- Each prospective database column is a field and is listed within its respective table.
- The asterisk (*) designates the primary key field.

Relationships are designated by lines with optional 'crow's feet' at either or both ends, depending on whether or not the side is a 'Many'.

♦ One to One (1:1) Relationship

A 1:1 relationship although valid, is rare. In a 1:1 relationship, each record in the first table is related to *one and only one* record in the second table. In many cases, you may wish to consolidate both tables into one, unless:

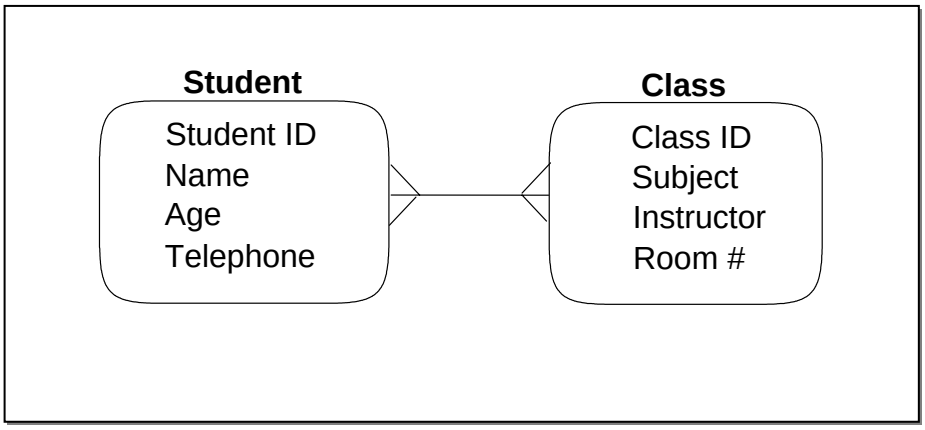
- Each table has a long list of fields.
- The subjects of the two tables are clearly distinguishable.
- Your forms will access the tables separately, more often than not.

♦ One to Many (1:M or M:1) Relationship

The most common of all relationships, the (1:M) or many to one relationship implies that each record on the "1" side may be related to one or more records on the "M" side.

♦ Many to Many (M:M) Relationship

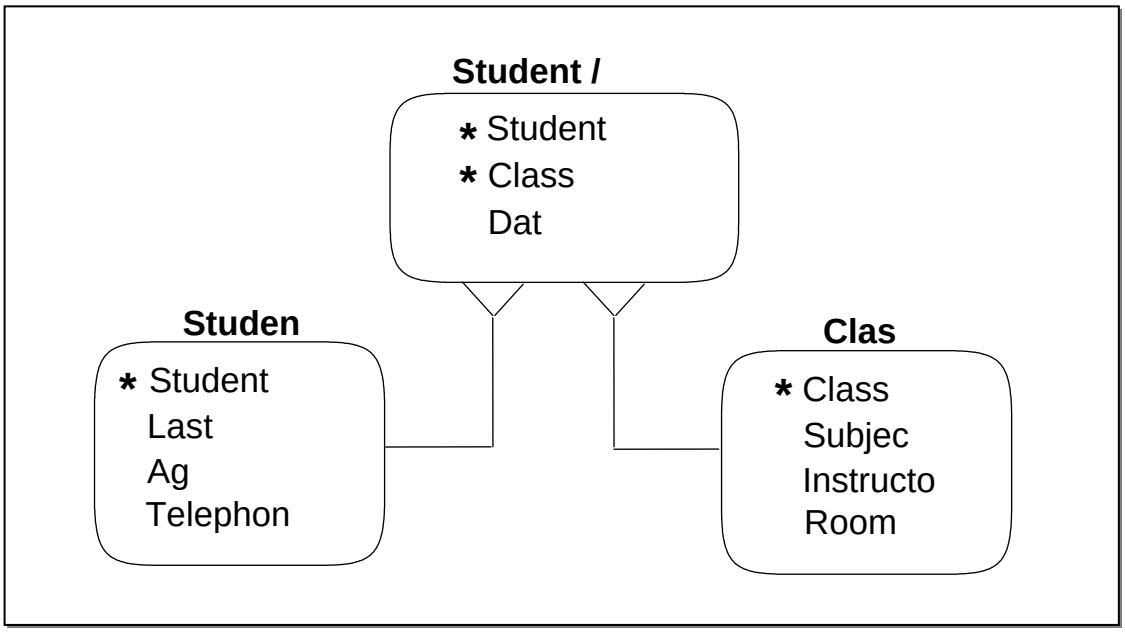
The complex many to many (M:M) relationship is quite common, and before implementation (the actual building of your application using a database tool) requires further resolution in the design phase. Let's examine the following diagram:



Many to Many

In the above illustration, as in real life, each **Student** can attend more than one **Class**; likewise, each **Class** can have more than one **Student**. Unfortunately, most database tools cannot accept this relationship as is. So, we will create another table (basically, a cross-reference table) called **Student/Class**.

NOTES



By combining the names of each of the original M:M fields, we create a meaningful name for the new table. Forever more, this naming convention will remind us of the table's purpose. Also note: the *Date* field was added to further describe the **Student / Class** table.

♦ Foreign Key

Although not specifically noted in the previous diagram, foreign keys are implied whenever a join occurs. A foreign key in a table is actually the primary key of the table on the other side of the join. It serves as the 'pointer' that links two specific records (one in each of the related tables) to each other. In a 1:M relationship, the foreign key generally is created in the table on the many side of the join. In a 1:1 relationship, the foreign key can go with either of the joined tables.

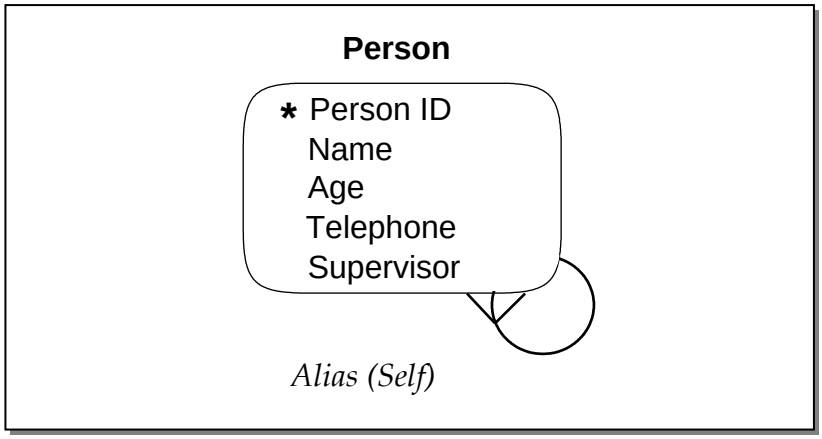
♦ Joins

A join is when data is retrieved from more than one table. This feature of joining two or more tables is what sets relational database systems apart from other types of databases. By having one field, the primary key, appear in different tables, the user has a way to join two tables together. So rather than having to look at tables separately, you can join them with the primary key and see the fields of both tables together. It is possible to join many tables together as long as there is a common field (primary key) among them.

A relationship in which a single table is related to itself is called an *Alias Join* or a *Self Join*. For example, in the following table, the *Supervisor* field contains the same type of information as the *Person ID* field, effectively 'pointing' to a different *Person ID*. This allows two persons to be 'linked' to

each other in a subordinate/supervisor relationship. Other times when a self join might be appropriate are the Parent/Child or Instructor/Student scenarios.

NOTE: A Self Join may be 1:1, 1:M or M:M.



STEPS:

1. Open Access.
2. Create a new database and name it.
3. Create all the tables needed and add the fields to each table as required.
4. Remember to define the primary keys.
5. Choose ***EDIT, RELATIONSHIPS*** from the menu.
6. Add all the tables needed to perform the join.
7. Click on a field name that is common in two or more tables. Select the field in one table and join from the one side of the relationship to the many side of the join by clicking and dragging to connect the tables. A line will connect the two field names between the tables.
8. Click on the ***SAVE*** icon.

NOTES

EXERCISE 11:

1) Open Access. Build the tables as outlined in previous exercises.

2) Using the tables of the Maine Widget database, model the join so that you can find the fields, **customer name, customer contact name, product name,** and **product description** on the **sales by customer** report (exercise 1).

LESSON 7: DATABASE QUALITIES

Regardless of the size and purpose of a new database application, there are certain qualities we should consider when designing it.

♦ Performance

Our database should respond well for the intended users. This means that when selected, either via a 'button' or menu selection, a form should appear as quickly as possible on the computer screen. Likewise, data retrieval time should be minimal and calculations expedient.

♦ Maintainability

Our database design should be as 'near normalized' as possible. This makes for easier modifications should 'bugs' occur or the need for enhancements arise. Likewise, when records are added, modified, or deleted, this process should be easy and fast.

♦ User Friendliness

A much overused (but not overrated) phrase, "user friendly" refers to the interface the user 'experiences', the 'glitz and glitter' if you will. To elaborate, the forms should be appealing, yet not too cluttered. Color and special effects should be used carefully, creating a professional, yet attractive visual effect. Fonts should be an acceptable size and type for ease of reading.

In addition to aesthetics, functionality is part of the user-friendly challenge. Key strokes and mouse clicks should be minimized as much as possible creating less work for the user. Movement among the forms and reports should be easily accomplished and somewhat intuitive, and, if possible, online context specific help should be available. Forms should fit on one screen to eliminate scrolling.

NOTES

♦ Models Real World Situation

Above all, your database should reflect the users' requirements. For the developer, this means involving the users in every step of the development process. At a minimum, you should:

- Conduct a personal interview to gather all data requirements.
- Explain and verify the E-R Model.
- Conduct a visual presentation of all form and report designs.
- Once the application is completed, provide users with a version of the application for the purpose of acceptance testing.

NOTES

LESSON 8: TESTING YOUR DESIGN

OVERVIEW

As you might expect, testing your design is extremely important. Make sure you dedicate some time in the development phase for testing purposes.

1. If your design consists of one table you will be better off to use a spreadsheet program for your data.

2. Test each Form:

   - Tab through each field.
   - Enter data where appropriate.
   - Ensure that several records are added to each table.

3. View each table to ensure the correct information was inserted into the records.

4. 'Run' each report.

Following testing, it's time to distribute your new database. And keep in mind:

- *There will always be enhancements.*
  Users have great imaginations (and many ideas too)! Welcome their ideas and continue to perfect your product. Remember, with each enhancement comes a new learning opportunity.

- *The true test comes during everyday usage.*
  Regardless of how thorough your testing scenario, no testing procedures are a true match against 'real world testing'. Expect *at least* a few 'bugs' to turn up.

EXERCISE 12:

1) Build forms, reports, and queries as needed for Maine Widget Company.

NOTES

LESSON 9: OTHER DATABASE TOPICS

♦ Databases on Mainframes

OVERVIEW

Downloading information from a database that resides on a mainframe is becoming a very common activity. If tasked to do so, be sure to involve all the right individuals: the 'caretaker' of the mainframe and programmers who understand how to interface with the mainframe database application. Also, read the help information provided in your database tool, specifically the "Importing" sections. Above all, talk with other developers who have accomplished downloads.

It may be very important to remember that unless the download is a live link, you will soon have outdated information in your smaller database. Figure out how you will deal with this. What should the "refresh" interval be?

♦ Object-Oriented Databases

OVERVIEW

As data structures become more complex, programmers are turning to a new database technology that can deliver more power without sacrificing ease of maintenance. Object-oriented databases are filling this need. Dubbed OODBMS (Object-oriented Database Management Systems), queries are performed on objects stored in the database. These objects are entities, which have certain attributes that can be cloned and used in other applications. These objects have subtypes and attributes, which set them apart from other entities. You can achieve more levels of abstraction with better performance. Expert or rule-based systems make great use of this technology.

GLOSSARY

- **Attribute / Column / Field**

A field is a single piece of information about a particular subject. Basically, these words are synonymous.

- **Database Application**

A database application is a collection of one or more of the following components: tables, queries, forms, reports (minimally, a database application has two tables), providing a series of data storage/display functions for end users.

- **Database / Table**

A table is a collection of individual pieces of information which should be related to the same general topic. *Note:* These two words may or may not be synonymous, depending upon the database tool used. For instance, Microsoft Access uses the word "database" to refer to the database application, which may contain one or more tables, queries ... By contrast, Lotus Approach refers to a single table as a "database".

- **Join**

When two databases are related, they are said to be "joined". Joins possess their own properties and can yield varying results when retrieving information from the joined tables.

- **Key Attribute / Primary Key / Key**

An attribute/field is what you designate as the unique identifier for a single row in a database. Typically, the Key field is used as the 'join' field when databases are related to each other. The Primary Key can be *complex* (compound), *artificial* (a counter field which consists of serialized numbers), or *natural* (a unique field that exits in your table like a birth date).

- **Normal Form: First**

First Normal Form is when each field contains the smallest meaningful value and the table does not contain repeating groups of fields.

- **Normal Form: Second**

Second Normal Form is important only when you're dealing with *Compound Keys*. In this situation, each field must depend on the **entire** Key including all fields which comprise the Compound Key.

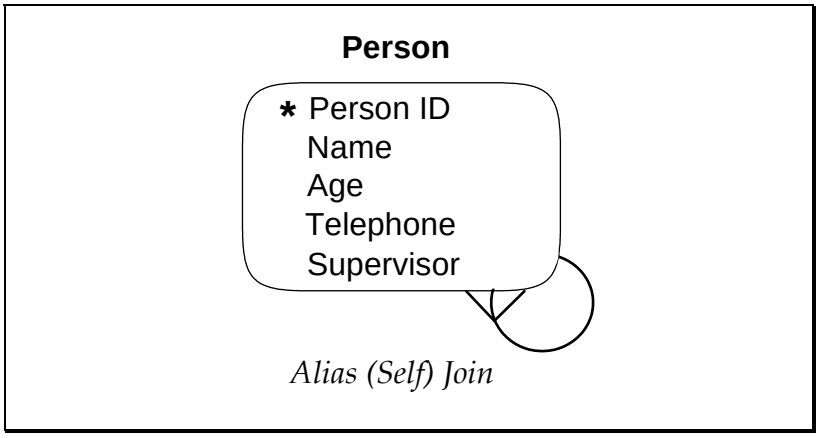
NOTES

- **Normal Form: Third**

Third Normal Form involves examining each field in your design and asking the question, *"Is this field dependent on the key?*"

- **Relationship: Alias (Self) Join**

A relationship in which a single table is related to itself. For example, in the following table, the *Supervisor* field contains the same type of information as the *Person ID* field, effectively 'pointing' to a different *Person ID*. This allows two persons to be 'linked' to each other in a subordinate/supervisor relationship. Other times when a self join might be appropriate are the Parent/Child or Instructor/Student scenarios. Note: a Self Join may be 1:1, 1:M or M:M.



Alias (Self) Join

- **Relationship: One-to-One (1:1)**

A One-to-One relationship is when a single record in each of the joined tables may be related to one and only one record in the other table.

- **Relationship: One-to-Many (1:M or M:1)**

A One-to-Many relationship is when a single record in the table on the *one side* of the join may be related to one or more records in the table on the *many side* of the join.

- **Relationship: Many-to-Many (M:M)**

A Many-to-Many relationship is when a single record in either table of the join may be related to one or more records in the other table of the join. M:M relationships must be resolved before the application can be created in a database tool.

- **Row / Record**

NOTES

A record is a collection of information (in a table) consisting of one occurrence of each field. For example, in a **Person** table comprised of the following fields: *Last Name, First Name, Age* and *SSN*, a typical row might contain the following information: Brown, Sam, 45, 007-55-1234.

ANSWERS TO EXERCISES

EXERCISE 1

1.

How many vendors do they have?
How many customers do they have?
What products are they selling the most of?
What efforts have been made to manage the inventory so far?
Who is involved in the inventory process?
Who is involved in tracking sales?
Where are most of the customers located?
What is the most popular product?
How much of it have they sold by customer?
Are they buying enough raw materials for our most popular product?

2.

```
**********************************************************************************************
**************
DATE: 99/99/99                    MAINE WIDGET COMPANY                  PAGE:  99
TIME:  99:99:99                      SALES BY CUSTOMER

CUSTOMER            CUSTOMER            PRODUCT             PRODUCT           QUANTITY
NAME                CONTACT NAME        NAME                DESCRIPTION           SOLD
!!!!!!!!!!!!!!!!!   !!!!!!!!!!!!!!!!!!!!!!!!  !!!!!!!!!!!!!!!!!!!!!!!!!! !!!!!!!!!!!!!!!!!!!!!    999.99

**********************************************************************************************
*************
DATE: 99/99/99                                  MAINE WIDGET COMPANY              PAGE  99
TIME:  99:99:99                                 CUSTOMER INFORMATION

NAME                ADDRESS             CITY          STATE        ZIP    PHONE        CONTACT
NAME
!!!!!!!!!!!!!!!!!!!  !!!!!!!!!!!!!!!!!!              !!!!!!!!!!!!!!   !!             !!!!!!   999-999-9999
       !!!!!!!!!!!!!!!!!!!!!!!!

**********************************************************************************************
*************
DATE: 99/99/99                    MAINE WIDGET COMPANY              PAGE  99
TIME:  99:99:99     PRODUCT INFORMATION - BEGINNING OF MONTH

                                                        BIN                IN
NAME                DESCRIPTION         COLOR         SIZE  LOCATION    STOCK
!!!!!!!!!!!!!!!!!!!  !!!!!!!!!!!!!!!!!!              !!!!!!!!!!!!!!   !!       !!!!!!        9999.99
**********************************************************************************************
**************
```

EXERCISE 2

Storyboard:

Maine Widget **has found** themselves unable **to track** sales on many of their *customers'* invoices. In order **to straighten** out this problem, Maine Widget **would like** to know *which* customers **are buying** *which* product. It **would** also **be helpful** for Maine Widget **to know** what *raw* material **to buy to supply** the demand.

Adjective	Noun	**Verb**
	Maine Widget	has found
	sales	to track
customer's	invoices	
	problem	to straighten
	Maine Widget	would like
which	customers	are buying
which	product	
	It	would be helpful to know
raw	material	to buy
	demand	to supply

EXERCISE 3

1.

customer name
customer address
customer city
customer state
customer zip
customer phone
customer contact name
product name
product description
product in stock beginning of month
product bin location
product size
product color
customer product quantity sold

2 and 3**.**

*customer Id
customer name
customer address
customer city
customer state
customer zip
customer phone
customer contact first name
customer contact last name
*product Id
product name
product description
product in stock beginning of month
product size
product color
*sales Id
customer Id
product Id
customer product quantity sold

* Primary key

EXERCISE 4:

There are no **compound keys.**

EXERCISE 5

*customer Id	*product Id	*Sales Id
customer name	product name	customer Id
customer address	product description	product Id
customer city	product in stock beginning of month	customer product quantity sold
customer state	product size	
customer zip	product color	
customer phone		
customer contact first name		
customer contact last name		

*Primary Key

EXERCISE 6

*customer Id	*product Id	*sales Id
customer name	product name	customer Id
customer address	product description	product Id
customer city	product in stock beginning of month	customer product quantity sold
customer state	product size	
customer zip	product color	
customer phone		
customer contact name		

*Primary Key

EXERCISE 7

Customer Id of Customer table is...
Customer Name of Customer table is...
Customer Address of Customer table is...
Customer City of Customer table is...
Product Id of Product table is...
Product Name of Product table is...

EXERCISE 8

Report names: Sales by Customer, Customer Information, and Product Information

Customer Table	Data Type	Size	Default Value
*customer Id	Counter		
customer name	Text	20	
customer address	Text	20	
customer city	Text	20	
customer state	Text	2	ME
customer zip	Text	9	
customer phone	Text	10	
customer contact name	Text	20	

Product Table	Data Type	Size	Default Value
*product Id	Counter		
product name	Text	20	
product description	Text	20	
product in stock beginning of month	Number	Double	
product color	Text	10	
product size	Text	5	
product bin location	Text	5	

Sales Table	Data Type	Size	Default Value
*sales Id	Counter		
customer Id	Number	Double	
product Id	Number	Double	
customer product quantity sold	Number	Double	

*Primary Key

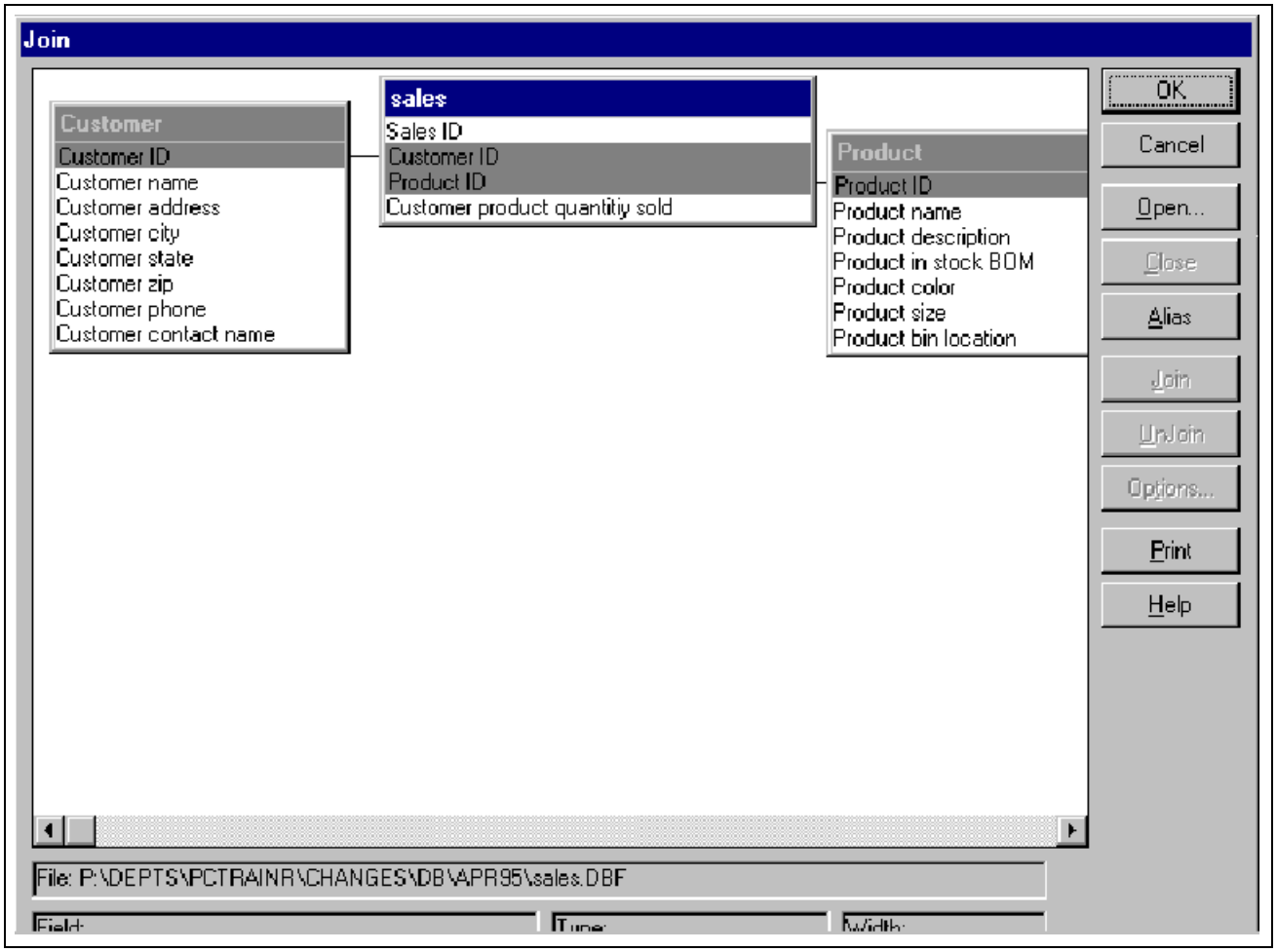
EXERCISE 9

There are no validation checks needed. However, there may be a special reason to include one.

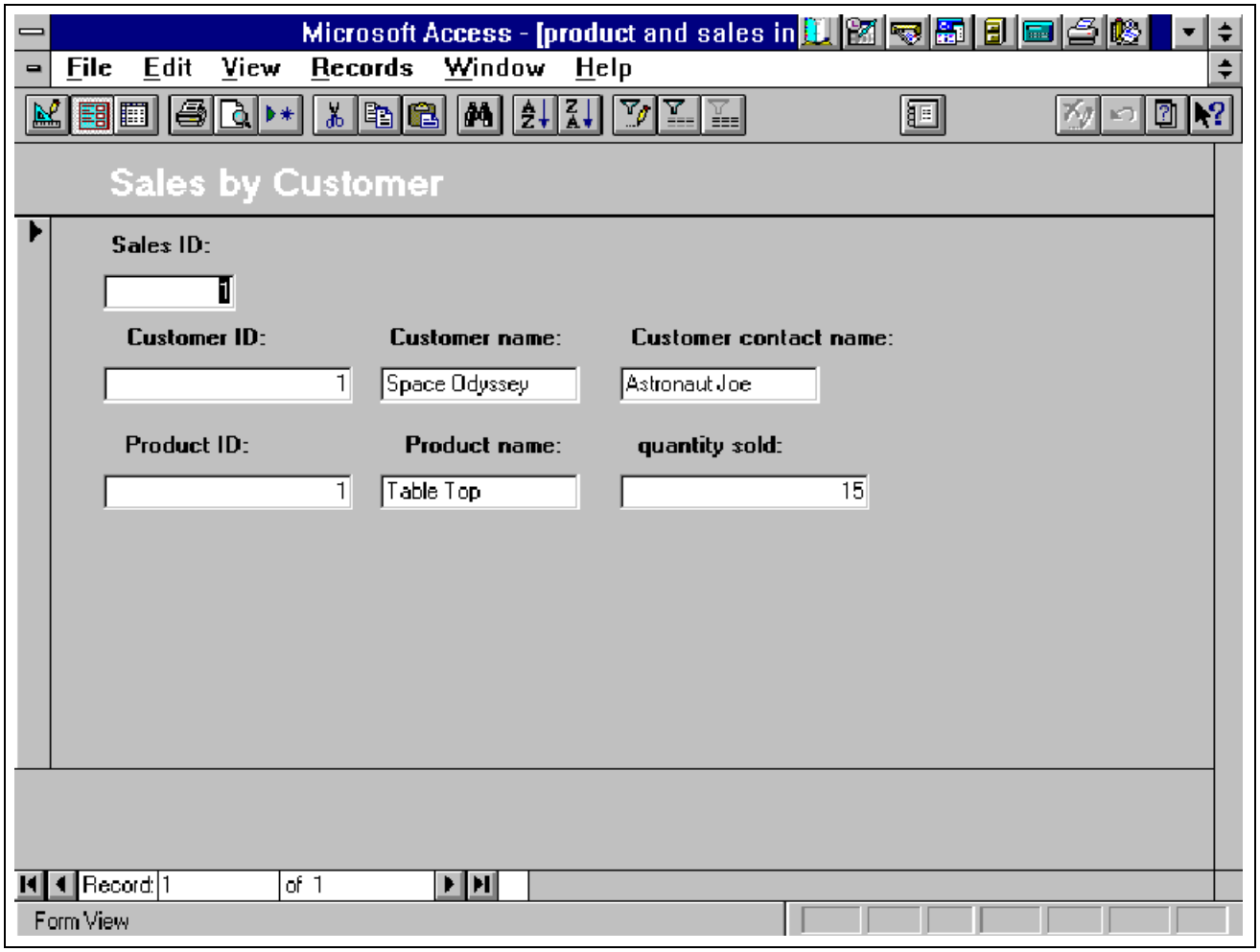
EXERCISE 10

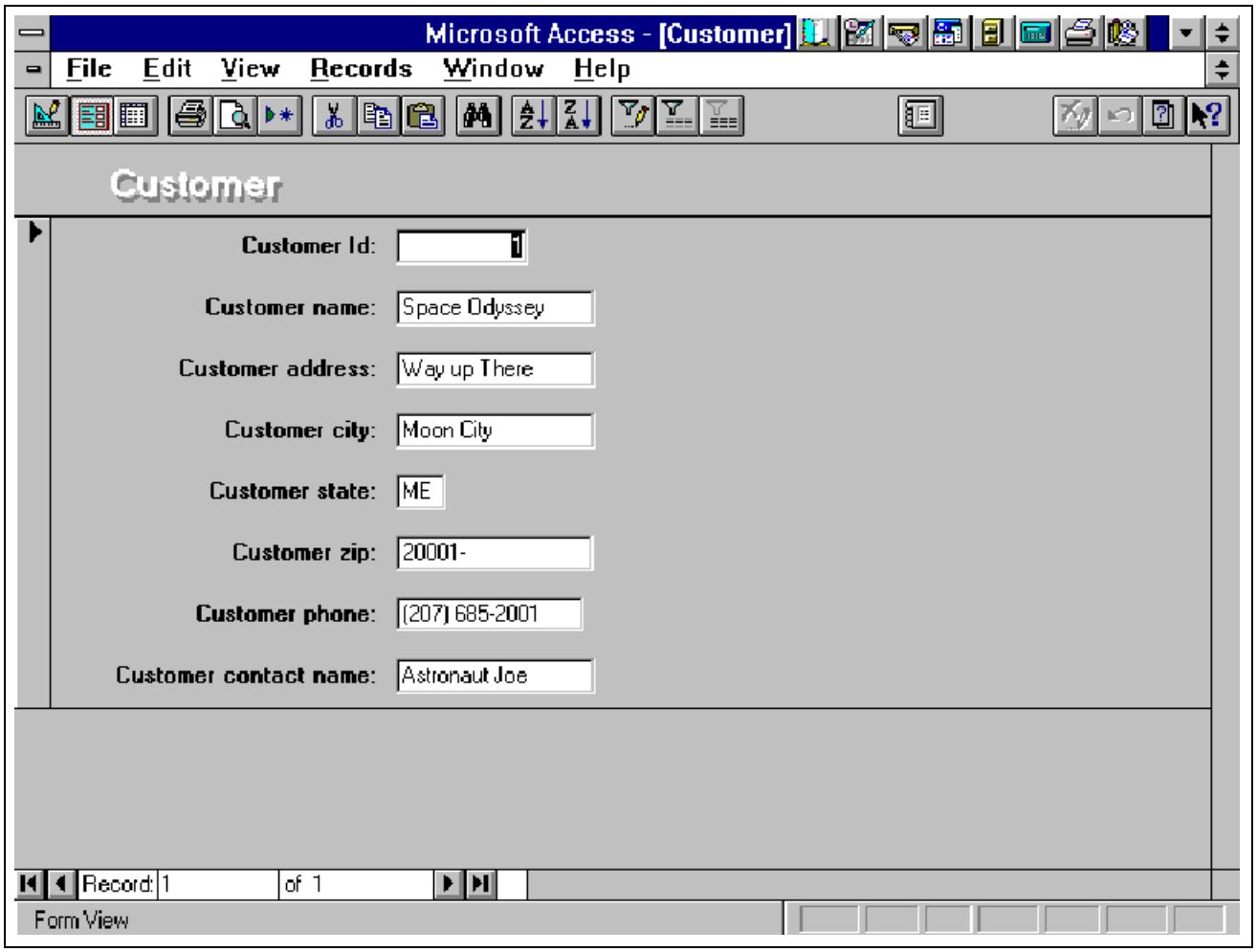
The entry, “In Stock,” on the report, “Product Information - Beginning of Month,” will be equal to the previous month’s number MINUS the total for each product of the entry, “Quantity Sold,” on the report, “Sales by Customer.” For this example, we will not add the number of items purchased during the month, but, of course, we would have to in the “real world.”

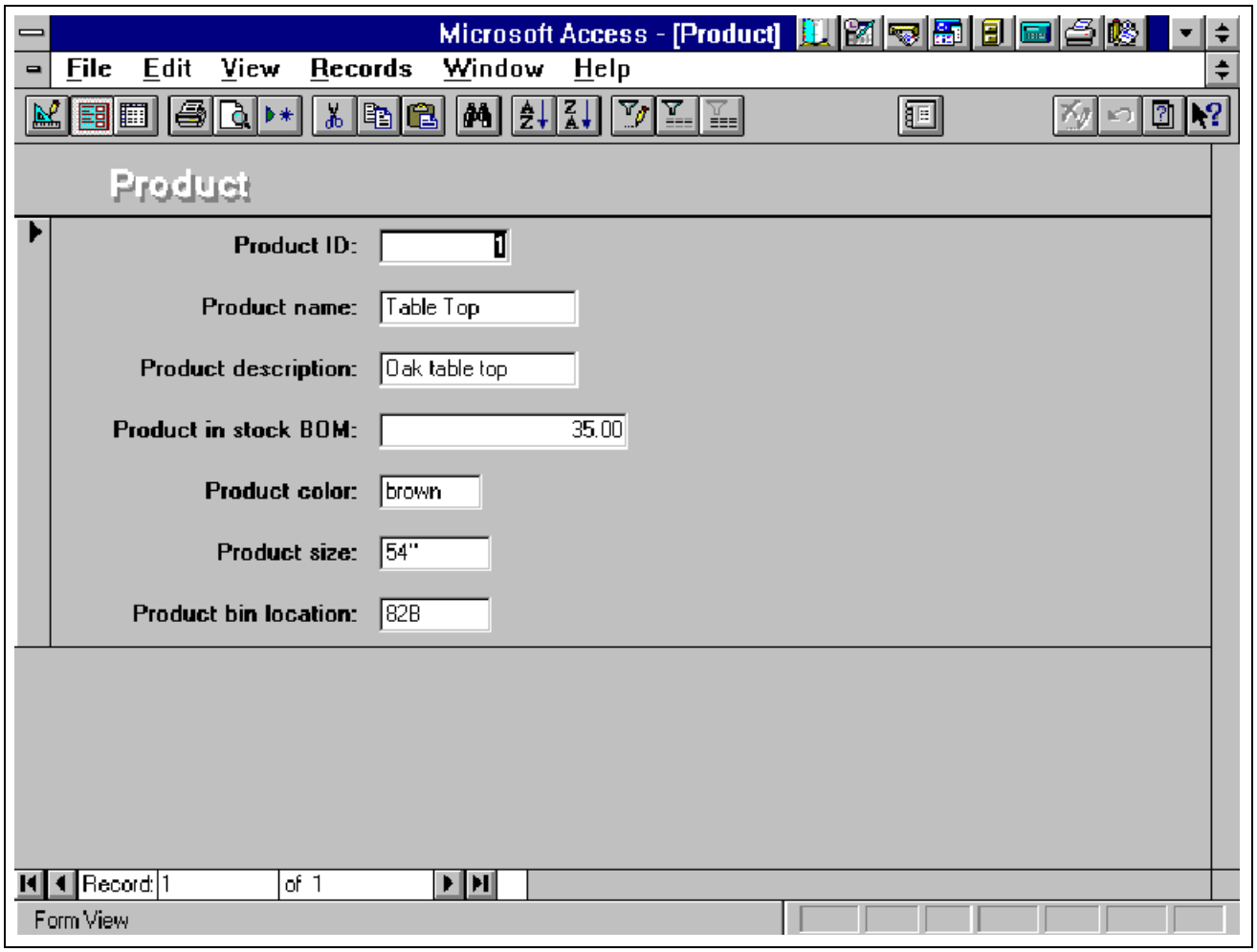
EXERCISE 11


Join
Customer
Customer ID
Customer name
Customer address
Customer city
Customer state
Customer zip
Customer phone
Customer contact name
sales
Sales ID
Customer ID
Product ID
Customer product quantitiy sold
Product
Product ID
Product name
Product description
Product in stock BOM
Product color
Product size
Product bin location
OK
Cancel
Open...
Close
Alias
Join
UnJoin
Options...
Print
Help
File: P:\DEPTS\PCTRAINR\CHANGES\DB\APR95\sales.DBF
Field:
Type:
Width:

EXERCISE 12


Microsoft Access - [product and sales in
File Edit View Records Window Help
Sales by Customer
Sales ID:
1
Customer ID:
Customer name:
Customer contact name:
1
Space Odyssey
Astronaut Joe
Product ID:
Product name:
quantity sold:
1
Table Top
15
Record: 1
of 1
Form View


Microsoft Access - [Customer]
File Edit View Records Window Help
Customer
Customer Id: 1
Customer name: Space Odyssey
Customer address: Way up There
Customer city: Moon City
Customer state: ME
Customer zip: 20001-
Customer phone: (207) 685-2001
Customer contact name: Astronaut Joe
Record: 1 of 1
Form View


Microsoft Access - [Product]
File Edit View Records Window Help
Product
Product ID: 1
Product name: Table Top
Product description: Oak table top
Product in stock BOM: 35.00
Product color: brown
Product size: 54"
Product bin location: 82B
Record: 1 of 1
Form View