EXTENDED LEARNING MODULE J

IMPLEMENTING A DATABASE WITH MICROSOFT ACCESS

Student Learning Outcomes

1. Identify the steps necessary to implement the structure of a relational database using the data definition language provided by Microsoft Access.
2. Demonstrate how to use the data manipulation subsystem in Access to enter and change information in a database and how to query that information.
3. Explain the use of the application generation subsystem in Access to create reports and data entry screens.

Introduction

A few short years ago, you could have performed a search of Monster.com and found literally hundreds of job postings requiring knowledge of Excel, spreadsheet software. Today, most companies *expect* you to know Excel—it's no longer a skill that will help you get a job; it's a necessity for evening getting an interview.

Today, the competitive advantage for many people seeking employment is a knowledge of database management system software, for which Microsoft Access is the most popular. We performed a search on Monster.com and found over 5,000 job postings requiring expertise in Microsoft Access. We performed two more searches, this time using *DBMS* and *database management system* as the search terms, and found an additional 5,500 + job postings. Some of the job titles included:

• Senior Financial Analyst	• Senior Accountant
• Material Control Specialist	• Managed Care Analyst
• Property Administrator	• Loan Servicing Auditor
• Regulatory Specialist	• Quality Assurance Inspector
• Payroll Coordinator	• Regional Sales Manager
• Claim Associate	• Product Requirements Analyst
• Compensation Manager	• Quantitative Market Researcher
• Policy Analyst	• Merchandising Specialist
• Service Team Leader	• Marketing Manager

If you look carefully at the above list, you'll see that not a single job title is IT-specific. Rather, they represent job openings in such areas as finance, logistics, retail sales, health care, and marketing. If you read *Extended Learning Module K, Careers in Business,* you'll find that every single business area covered—accounting, finance, hospitality and tourism management, management, marketing, production and operations management, and real estate and construction management—lists database management as an IT skill you should pursue to be a success in those areas.

That's why we wrote this learning module—because a knowledge of database management system software can help you in your professional career. While it might not in and of itself help you land that really great job (who knows, perhaps it will), your work responsibilities will most probably include some forms of information management. Database management system software can you help you manage that information.

Solomon Enterprises Database

In Chapter 3 we discussed the important role that databases play in an organization. We followed that with *Extended Learning Module C,* in which you learned how to design the correct structure of a relational database. That module includes four primary steps. They are:

1. Define entity classes and primary keys.
2. Define relationships among the entity classes.
3. Define information (fields) for each relation (the terms *relation* and *table* are often used to refer to a file in the context of a database).
4. Use a data definition language to create your database.

In *Extended Learning Module C,* you followed the process through the first three steps above. In this module, we'll take you through the fourth step—using a data definition language to create your database—by exploring the use of Microsoft Access, today's most popular personal database management system package (it comes as part of Microsoft's Office suite).

We'll also show you how to use Microsoft Access's data manipulation subsystem, including how to enter and change information and build queries; and how to use the application generation subsystem to create reports and input forms.

In Figure J.1 (on this page and the next) we've recreated the complete Solomon Enterprises database structure we defined in *Extended Learning Module C.* If it has been a while since you covered that module, we suggest that you review it before creating the database.

Figure J.1
The Database Structure We'll Be Implementing
(continued on the next page)

CONCRETE TYPE RELATION

Concrete Type	Type Description
1	Home foundation and walkways
2	Commercial foundation and walkways
3	Premier speckled (with smooth gravel aggregate)
4	Premier marble (with crushed marble aggregate)
5	Premier shell (with shell aggregate)

CUSTOMER RELATION

Customer Number	Customer Name	Customer Phone	Customer Primary Contact
1234	Smelding Homes	3333333333	Bill Johnson
2345	Home Builders Superior	3334444444	Marcus Connolly
3456	Mark Akey	3335555555	Mark Akey
4567	Triple A Homes	3336666666	Janielle Smith
5678	Sheryl Williamson	3337777777	Sheryl Williamson
6789	Home Makers	3338888888	John Yu

EMPLOYEE RELATION

Employee ID	Employee Last Name	Employee First Name	Date of Hire
123456789	Johnson	Emilio	2/1/1985
435296657	Evaraz	Antonio	3/3/1992
785934444	Robertson	John	6/1/1999
984568756	Smithson	Allison	4/1/1997

SUPPLIER RELATION

Supplier ID	Supplier Name
412	Wesley Enterprises
444	Juniper Sand & Gravel
499	A&J Brothers
999	N/A

TRUCK RELATION

Truck Number	Truck Type	Date of Purchase
111	Ford	6/17/1999
222	Ford	12/24/2001
333	Chevy	1/1/2002

ORDER RELATION

Order Number	Order Date	Customer Number	Delivery Address	Concrete Type	Amount	Truck Number	Driver ID
100000	9/1/2004	1234	55 Smith Lane	1	8	111	123456789
100001	9/1/2004	3456	2122 E. Biscayne	1	3	222	785934444
100002	9/2/2004	1234	55 Smith Lane	5	6	222	435296657
100003	9/3/2004	4567	1333 Burr Ridge	2	4	333	435296657
100004	9/4/2004	4567	1333 Burr Ridge	2	8	222	785934444
100005	9/4/2004	5678	1222 Westminster	1	4	222	785934444
100006	9/5/2004	1234	222 East Hampton	1	4	111	123456789
100007	9/6/2004	2345	9 W. Palm Beach	2	5	333	785934444
100008	9/6/2004	6789	4532 Lane Circle	1	8	222	785934444
100009	9/7/2004	1234	987 Furlong	3	8	111	123456789
100010	9/9/2004	6789	4532 Lane Circle	2	7	222	435296657
100011	9/9/2004	4567	3500 Tomahawk	5	6	222	785934444

RAW MATERIAL RELATION

Raw Material ID	Raw Material Name	QOH	Supplier ID
A	Water	9999	999
B	Cement paste	400	412
C	Sand	1200	444
D	Gravel	200	444
E	Marble	100	499
F	Shell	25	499

BILL OF MATERIAL RELATION

Concrete Type	Raw Material ID	Unit
1	B	1
1	C	2
1	A	1.5
2	B	1
2	C	2
2	A	1
3	B	1
3	C	2
3	A	1.5
3	D	3
4	B	1
4	C	2
4	A	1.5
4	E	2
5	B	1
5	C	2
5	A	1.5
5	F	2.5

Figure J.1
The Database Structure We'll Be Implementing *(continued)*

LEARNING OUTCOME 1

IMPLEMENTING THE STRUCTURE OF THE SOLOMON ENTERPRISES DATABASE

As we said previously, you can't simply start typing information into a database as you can when you create a document with Word or a workbook with Excel. You must first define the correct structure of the database (see *Extended Learning Module C*), and then create the structure of the database by creating its data dictionary before entering any information. The ***data dictionary*** contains the logical structure for the information in a database. It includes a description of each relation (also called a *table* or *file*) and each piece of information in each relation.

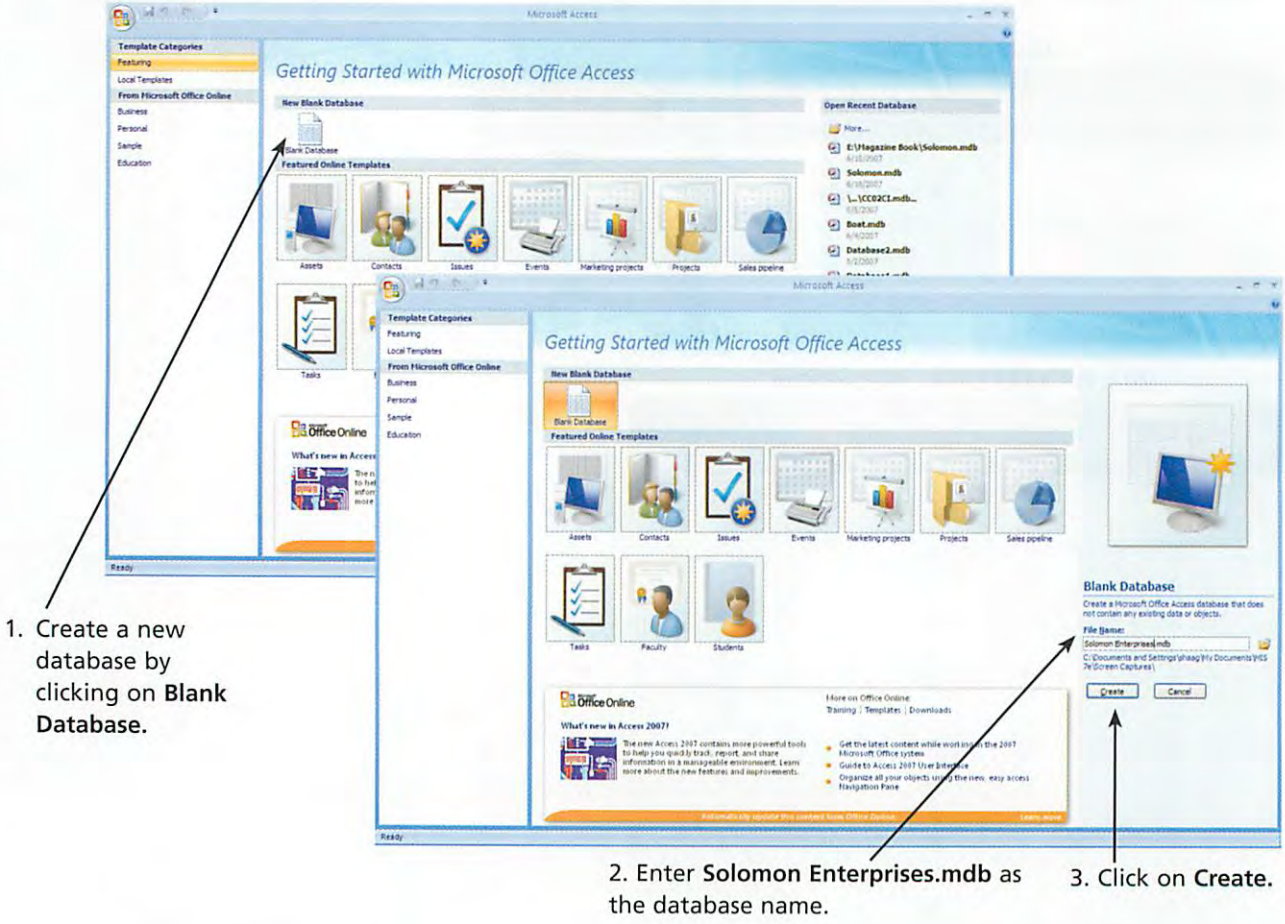
To create a database using Microsoft Access, we performed the following steps (see Figure J.2). We will assume you already have Microsoft Access open.

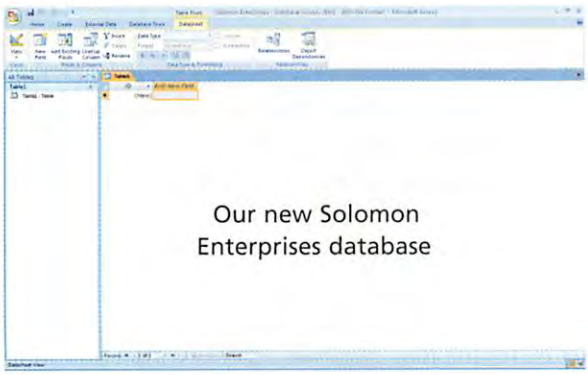
1. Click on **Blank Database** in the upper left corner of the screen.
2. Enter **Solomon Enterprises.mdb** as the database name.
3. Click on **Create.**

Throughout this module, we will include numerous steps and screen captures (labeled according to the step) for clarification.

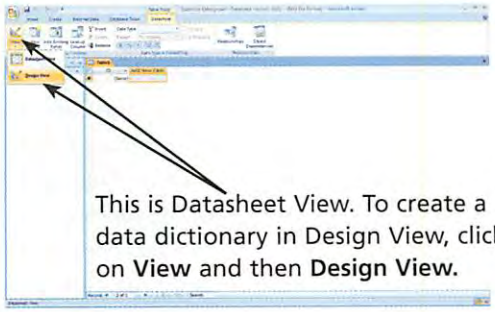
Figure J.2

The First Step in Creating a Database

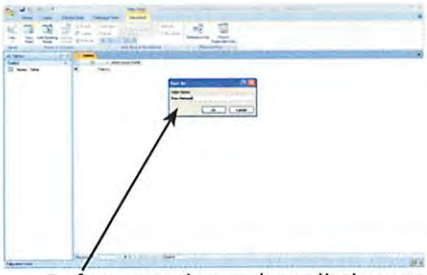




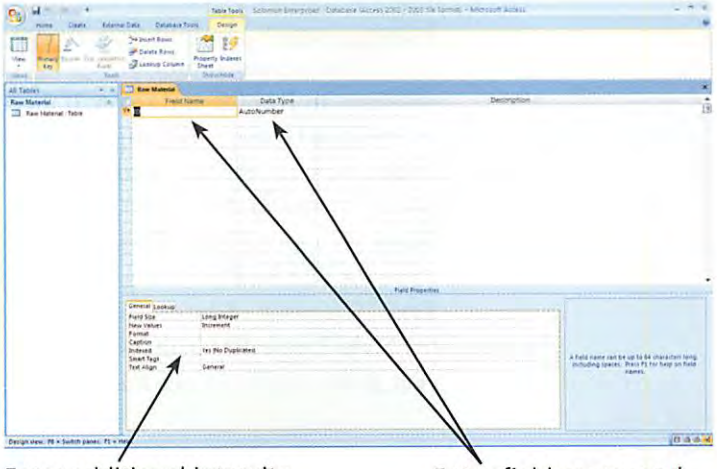
Our new Solomon Enterprises database



This is Datasheet View. To create a data dictionary in Design View, click on **View** and then **Design View.**



Before creating a data dictionary in Design View, you must first provide a table name. We did so with *Raw Material.*



Enter additional integrity constraints in **Field Properties.**

Enter field names and data types here.

Figure J.3

The First Step in Defining the Structure for Each Relation

After we followed these steps, we got the upper left screen in Figure J.3. Now that we've created a blank database with the name **Solomon Enterprises.mdb,** we're ready to define the structure of the database.

There are two main ways to create the data dictionary for e relation: in Datasheet View; and in Design View. They both achieve the same result. We'll do so using the Design View. The upper left screen in Figure J.3 allows you to create a relation in Datasheet View. To switch to Design View, click on **View** in the upper left corner of the screen and then click on **Design View** (the upper right Screen in Figure J.3). Access will ask you for a file name (the lower left screen in Figure J.3), and we entered *Raw Material* (and clicked on **OK**) because that is the first relation we're going to work with. Access then presented us with the lower right screen in Figure J.3, which is the Design View for the *Raw Material* relation.

IMPLEMENTING THE *RAW MATERIAL* RELATION STRUCTURE

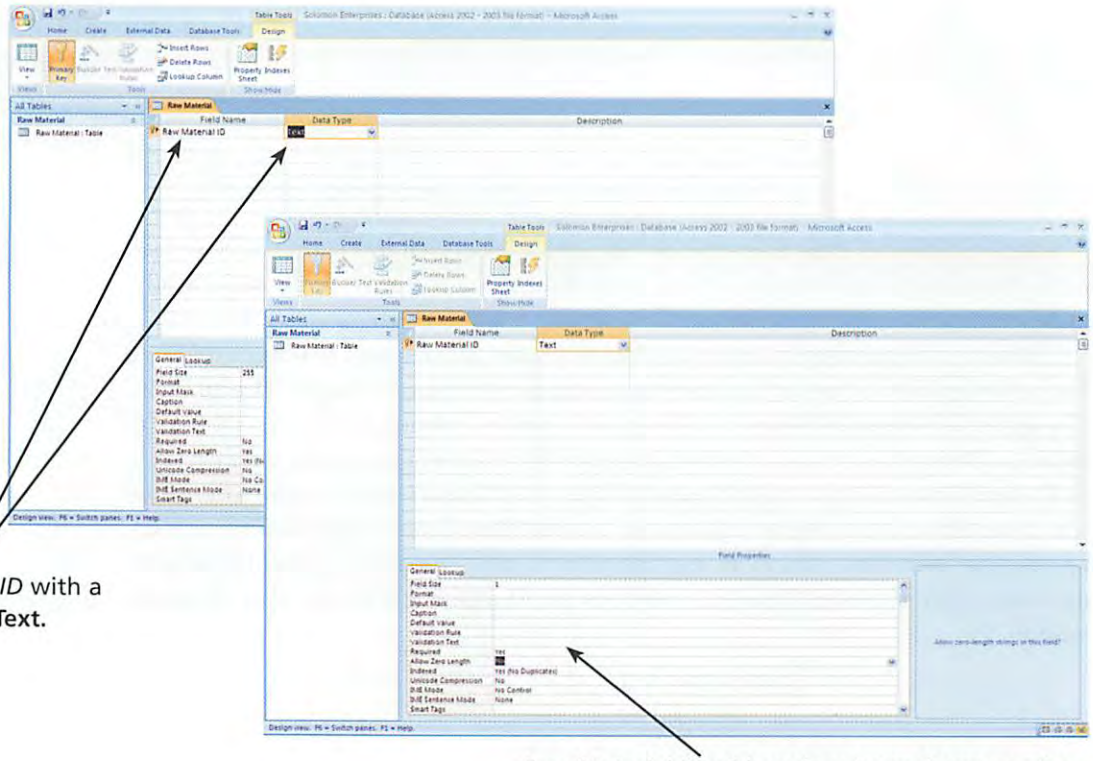
In the lower right screen in Figure J.3, Access will now allow us to enter field names, data types, and descriptions (the last is optional) for the *Raw Material* relation. We'll also be entering some information in the **Field Properties** box in the lower portion of the screen. That area shows the various field properties for whatever field we have selected in the top portion of the screen.

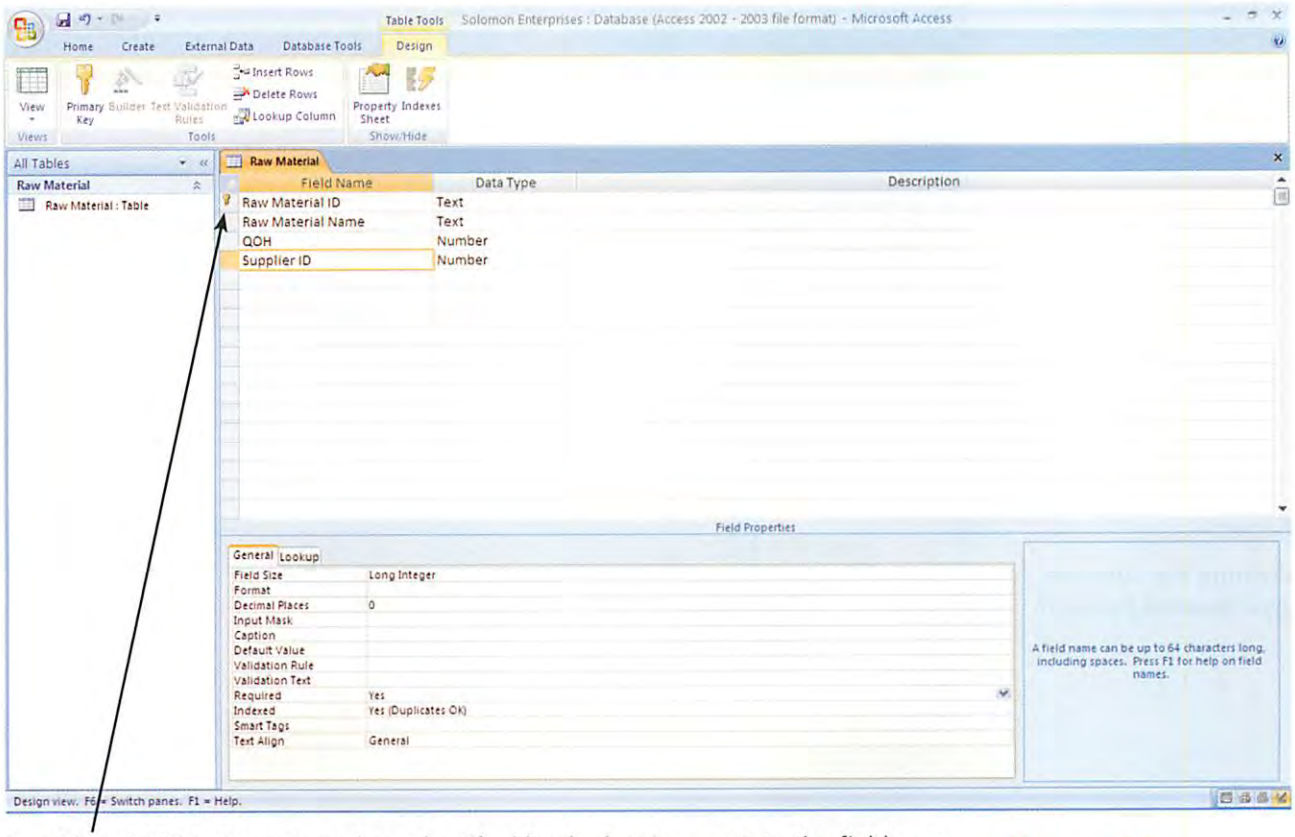
Once you have entered all that information for a given relation, you save that structure and repeat the process until you've created the structure for each relation in your database. We'll do that for three of the relations in Solomon's database.

To start, let's implement the structure for the *Raw Material* relation within **Solomon Enterprises.mdb.** The *Raw Material* relation has four fields: *Raw Material ID, Raw Material Name, QOH,* and *Supplier ID.* As you can see in the lower right screen in Figure J.3 on the previous page, Access assumes that the first field name is *ID* and that its Data Type is **AutoNumber,** meaning that the first record will have an ID of 1, the second will have an ID of 2, and so on. If this is the format you want for the first field, you can accept it and move to defining the second field.

But we want the first field name to be *Raw Material ID* and its data type to be text (all numbers and characters). To implement that, we clicked in the first row of the **Field Name** column, deleted *ID,* and entered *Raw Material ID.* We then moved to the **Data Type** for that field (by using the tab key), clicked on the pull down arrow, and selected **Text.** The result of those actions is shown in left screen in Figure J.4. Now, take a look at the **Field Properties** in that screen. The **Field Size** defaults to 255 characters, which we changed to 1 because our *Raw Material IDs* range from A to F, one single character.

Figure J.4
Creating the *Raw Material* Relation Structure





Raw Material ID is the primary key, identified by the **key** icon next to the field row.

Figure J.5
The Final Structure for the *Raw Material* Relation

We also changed **Required** to Yes because we want to require that each raw material have a *Raw Material ID.* That is an example of an integrity constraint. ***Integrity constraints*** are rules that help ensure the quality of information. And we also changed **Allow Zero Length** to *No,* which again requires that something is entered in the *Raw Material ID* when a new record is added. The result of our changes is shown in the right screen in Figure J.4.

To complete creating the structure of the *Raw Material* relation, we then clicked in the second row in the **Field Name** column and proceeded to enter the field names and data types for the three remaining fields of *Raw Material Name, QOH,* and *Supplier ID.* The screen showing the final structure of the Raw Material relation is shown in Figure J.5.

When creating the data dictionary entries for both *QOH* and *Supplier ID,* we chose **Number** as the data type and set **Decimal Places** to zero in the **Field Properties** box. For *Supplier ID,* we also changed **Required** to Yes because we want to know which supplier is providing a given *Raw Material.* These are more examples of integrity constraints that Access will now enforce for us.

Access defaults to identifying the first field as the primary key. A ***primary key*** is a field (or group of fields in some cases) that uniquely describes each record. In our case, the *Raw Material ID,* which we placed in the first field, is the primary key. If you need to change that, position the cursor in the line of the appropriate primary key field and click on the **Primary key** button in the menu area.

To save the structure (which you'll need to do if you make any changes), click on the **Save** button (the disk icon) in the upper left corner of the screen.

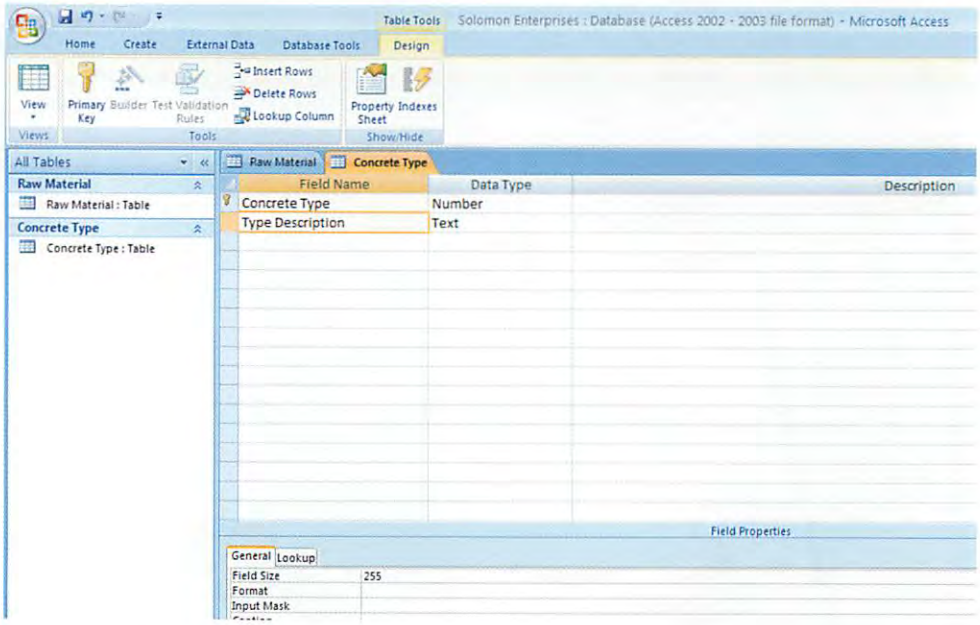


Figure J.6
Creating the *Concrete Type* Relation Structure

IMPLEMENTING THE *CONCRETE TYPE* RELATION STRUCTURE

To develop the structures of the remaining relations, you simply follow the process outlined in the previous section. Let's now create the structure for the *Concrete Type* relation. To create the *Concrete Type* relation using the Design View (you can have the Design View open for the *Raw Material* relation from the previous task), click on **Create** in the menu and then click on the **Table** button.

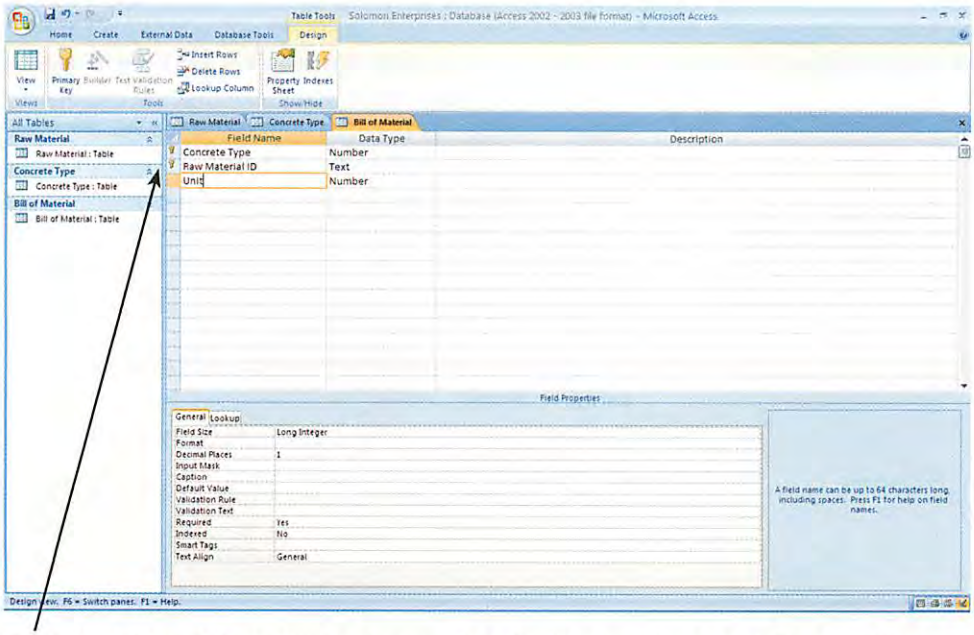
You will once again see a blank and unnamed table in Datasheet View. Click on the **View** button, select **Design View,** enter a new table name (*Concrete Type* in this instance), and click on **OK.** What you will then see is a screen identical to the lower right screen in Figure J.3 on page 435 that we used to create the *Raw Material* relation, only now we're creating the *Concrete Type* relation.

In Figure J.6, you can see that we entered the field names of *Concrete Type* and *Type Description. Concrete Type* is also the name of the table, and that's fine if it suits your purpose. *Concrete Type* is a numeric field (**Number**) and *Type Description* is a text field. *Concrete Type* is the primary key field as you can see by the Key icon to the left of the field name.

Specifically, for each we made some additional modifications to their field properties as follows:

- *Concrete Type*
  - **Decimal Places** = zero
  - **Required** = Yes
- *Type Description*
  - **Required** = Yes
  - **Allow Zero Length** = No

We did this to further enforce integrity constraints within the database. We then saved this table structure by clicking on **Save**—the disk icon—in the upper left portion of the screen.



The *Bill of Material* relation is a composite relation, meaning that it has two field (*Concrete Type and Raw Material ID*) making up the primary key.

Figure J.7
Creating the *Bill of Material* Relation Structure

IMPLEMENTING THE *BILL OF MATERIAL* RELATION STRUCTURE

The last relation that we'll cover here is that of the *Bill of Material* relation. As you can see in Figure J.7, we've entered all the field names and their types.

This relation is a little different from the rest because it has a composite primary key. A ***composite primary key*** consists of the primary key fields from the two intersecting relations. An ***intersection relation*** (sometimes called a ***composite relation***) is a relation you create to eliminate a many-to-many relationship. In *Extended Learning Module C,* we created the *Bill of Material* relation to eliminate the many-to-many relationship that existed between *Concrete Type* and *Raw Material.* So, the *Bill of Material* relation has a primary key composed of two fields: the primary key *Concrete Type* from the *Concrete Type* relation and the primary key *Raw Material ID* from the *Raw Material* relation. The primary key of the *Bill of Material* relation is a composite primary key.

To identify two fields that aggregately create a primary key, we followed these steps:

1. Define the basic structure of the relation by entering the field names and their properties.
2. Move the pointer to the column immediately to the left of the first field name of the composite primary key (the pointer will turn into an arrow pointing to the right).
3. Click on that row and don't unclick.
4. Drag the pointer so that the second field in the composite primary key is also highlighted.
5. Unclick.
6. Click on the **Primary Key** button.

The Key icon will appear next to each of the two fields, identifying that together they make up a composite primary key.

Relation	Notes
Customer	*Customer Number:* Primary key, Number, no decimals, required entry *Customer Name:* Text, required entry *Customer Phone:* Text (so you can add parentheses around the area code and a dash if you wish), required entry (optional for our purposes) *Customer Primary Contact:* Text, required entry (optional for our purposes)
Employee	*Employee ID:* Primary key, Number, no decimals, required entry *Employee Last Name:* Text, required entry (optional for our purposes) *Employee First Name:* Text, required entry (optional for our purposes) *Date of Hire:* Date/Time, required entry (optional for our purposes)
Supplier	*Supplier ID:* Primary key, Number, no decimals, required entry *Supplier Name:* Text, required entry (optional for our purposes)
Truck	*Truck Number:* Primary key, Number, no decimals, required entry *Truck Type:* Text, required entry (optional for our purposes) *Date of Purchase:* Date/Time, required entry (optional for our purposes)
Order	*Order Number:* Primary key, Number, no decimals, required entry *Order Date:* Date/Time, required entry *Customer Number:* Number, no decimals, required entry *Delivery Address:* Text, required entry (optional for our purposes) *Concrete Type:* Number, no decimals, required entry *Amount:* Number, no decimals, required entry *Truck Number:* Number, no decimals, required entry *Driver ID:* Number, no decimals, required entry

Figure J.8
Field Information for the Remaining Relations

You now need to create the relation structures for *Employee, Order, Supplier, Truck,* and *Customer.* We'll leave that task to you. However, you can use Figure J.8 as a reference guide as it provides notes concerning data types, keys, and the like for each field in those relations.

Defining Relationships within the Solomon Enterprises Database

So far, we've created the structure of the relations for our database. In other words, we specified the names and types of the field in each of the tables. As you can see, the process is very different from creating a word processing document or a workbook. We haven't yet entered any information into our database. We have one last structural issue to take care of, and that is to define how all the relations or files relate to each other.

Recall from our discussions in Chapter 3 and *Extended Learning Module C* that you can create relationships among the various tables by identifying foreign keys. A ***foreign key*** is a primary key of one file (relation) that appears in another file (relation). See Figure J.9 for the logical ties between primary and foreign keys.

Note that all the foreign keys have the same names as the primary keys in the original tables, except for *Employee ID*, which becomes *Driver ID* in the *Order* relation.

It's vitally important that you establish the relationships between primary and foreign keys. That way, the DBMS can enforce integrity constraints and disallow inconsistent information being entered. For example, when we specify that *Supplier ID* is a foreign key (from the *Supplier* relation) in the *Raw Material* relation, the DBMS will not allow us to enter a *Supplier ID* in the *Raw Material* relation that does not appear as a primary key in the *Supplier* relation. This makes business sense: You can't get raw material from a supplier who doesn't exist.

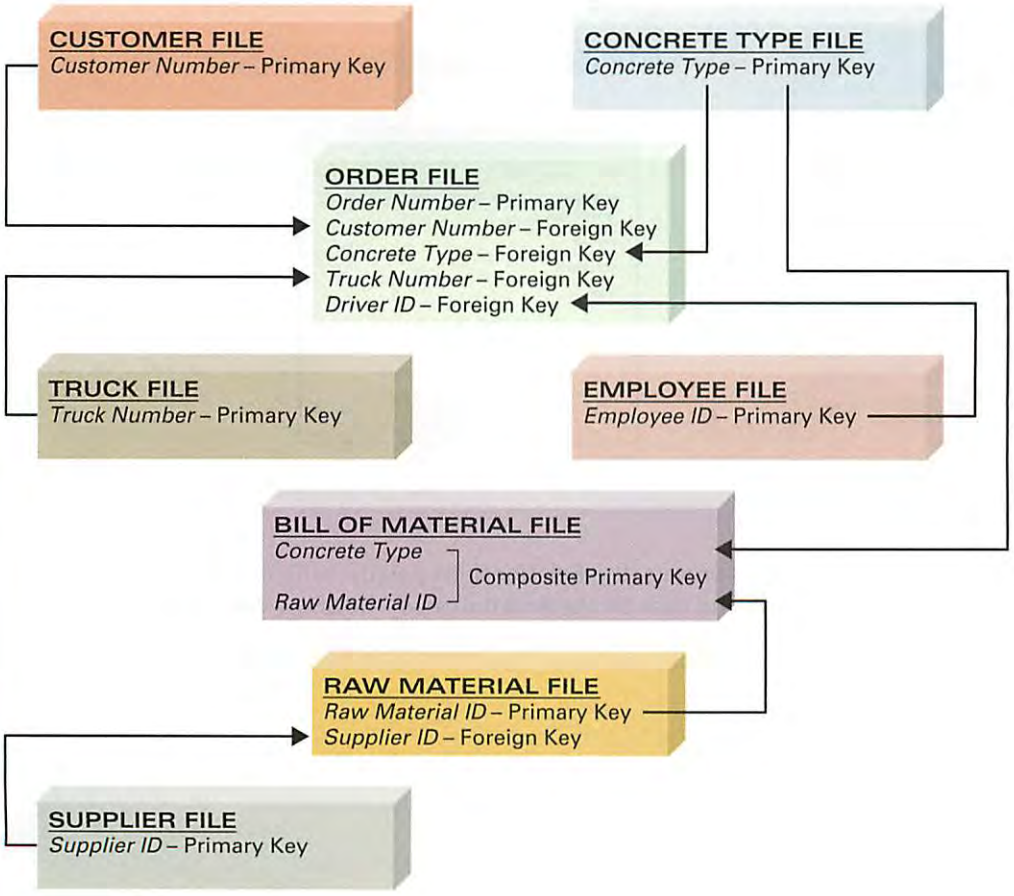


Figure J.9
Primary and Foreign Key Logical Ties

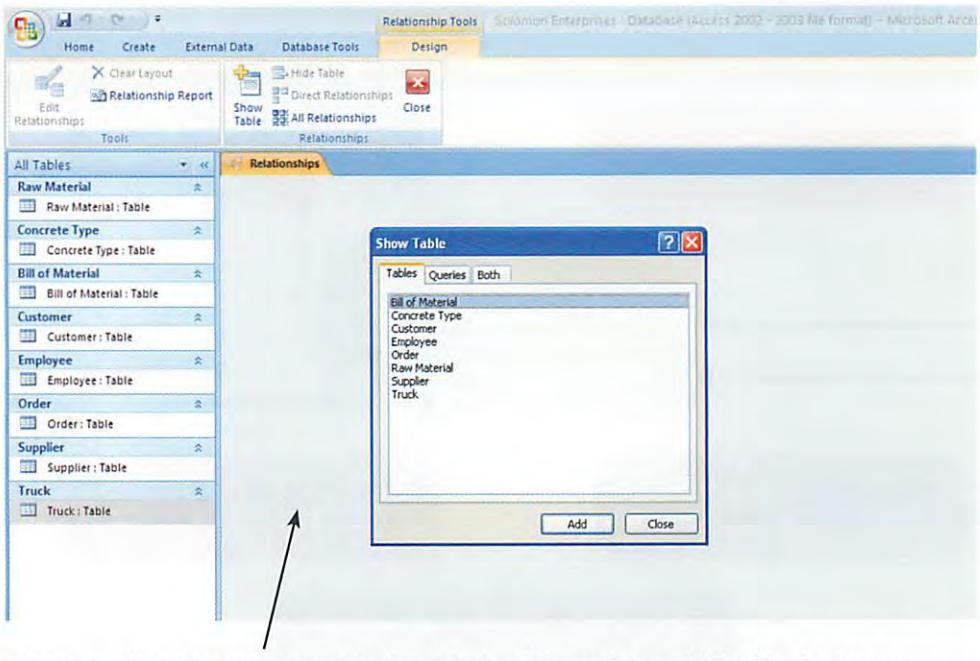
Prior to creating these relationships, you need to close all tables that you have open. You can easily right click on a tab and select **Close.** Then, to create these relationships, which means telling Access which fields are foreign keys, you click on the **Database Tools** in the menu area and then click on the **Relationships** button. You'll then see the screen in Figure J.10. Notice that it lists all of the relations in our database. When you first start this process, the palette in the background is blank. To identify the relationships, you must make each relation appear on the palette. To do this, simply highlight each relation name and click **Add.** When you have all the tables on the palette, click on the **Close** button to make the **Show Table** box disappear. When all the tables are on the palette, you're ready to identify how all the tables relate to each other.

To do this (and you must follow this process exactly), we clicked on and dragged each primary key to its respective foreign key counterpart and dropped it there. Once you drop the primary key onto its foreign key counterpart, you'll see the Edit Relationships box (see Figure J.11 on the next page). In that box, we clicked on **Enforce Referential Integrity** and then **Create.**

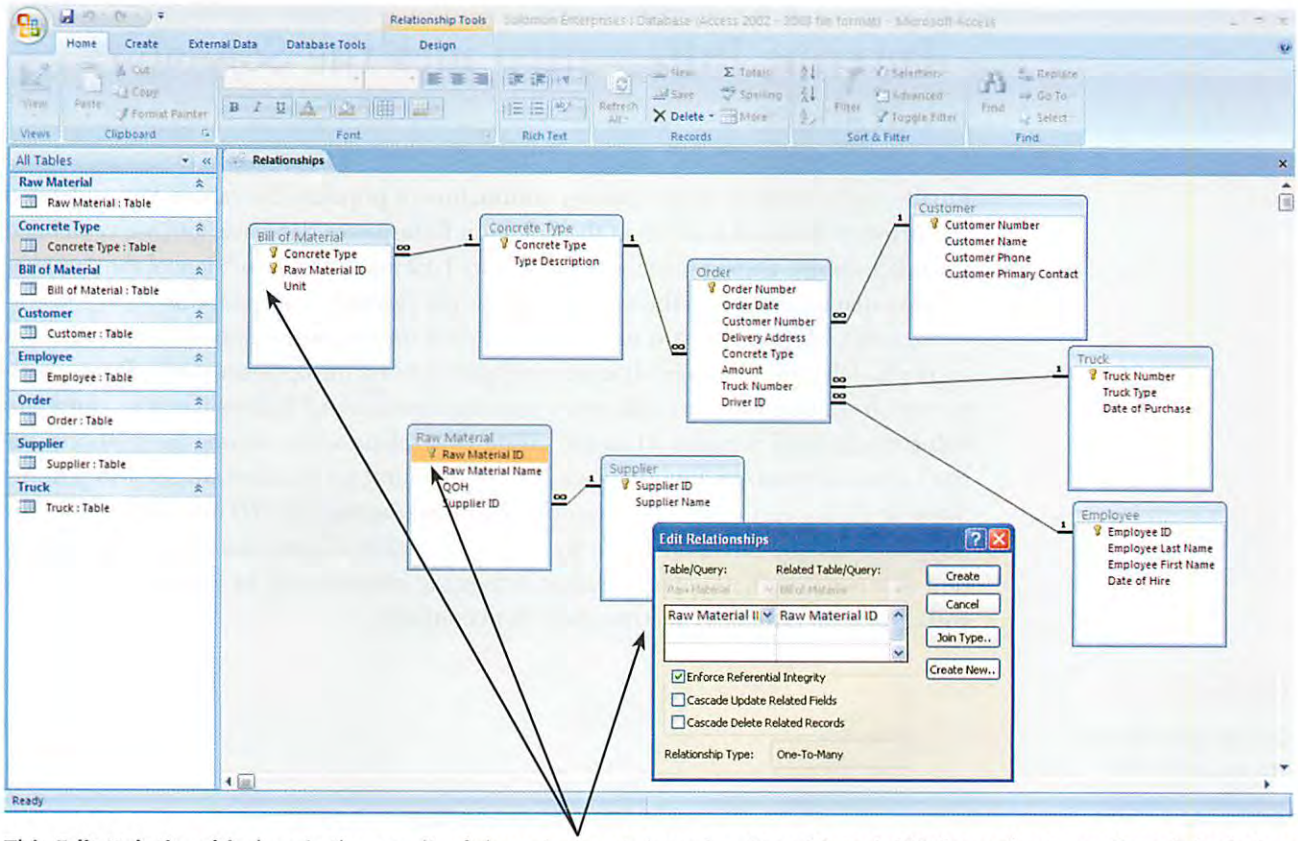
When you drag and drop a primary key onto a foreign key, Microsoft Access assumes that the relationship is one-to-many (1:M). That is, a primary key may appear many times as a foreign key, and a foreign key must appear once and only once as a primary key. If you perform the process in reverse (dragging and dropping the foreign key onto a primary key), the relationship will be reversed (M:1), which is not what you want.

Figure J.10

The First Step in Defining Relationships in a Database



When you first start this process, the **Relationships** palette will be blank. As you highlight each relation and click on the **Add** button in the Show **Table** box, the relation will appear on the palette.



This **Edit Relationship** box is the result of dragging and dropping *Raw Material ID* from the *Raw Material* relation onto the *Bill of Material* relation. As a general rule, you should turn on **Enforce Referential Integrity** to protect your database from inconsistent data. Then, click on the **Create** button and you get the connecting line.

Figure J.11

Defining Relationships for the Solomon Enterprises Database

You also need to turn on the enforcement of referential integrity. By doing so, your DBMS will make sure that when you enter a foreign key in a relation it matches one of the primary keys in the other relation. Figure J.11 shows the **Edit Relationships** box we got when we dragged and dropped *Raw Material ID* from the *Raw Material* relation onto *Raw Material ID* in the *Bill of Material* relation. We also clicked on **Enforce Referential Integrity** and **Create** and got the connecting lines you see between the other pairs of relations.

In Figure J.11, you can see many instances of a line connecting the primary key in one relation to the foreign key in another. Let's look first at the relationship between the *Supplier* and *Raw Material* relations where Access shows a connecting line between *Supplier* and *Raw Material* with a 1 near the *Supplier* relation and an infinity symbol (∞) near the *Raw Material* relation. Likewise, a *Supplier ID* that appears in the *Raw Material* relation can appear only one time in the *Supplier* relation. Also note that the foreign key can have a different name than the primary key, as, for example, *Employee ID* in the *Employee* relation becomes *Driver ID* in the *Order* relation. This is quite permissible, and how you name the field depends on your context.

Once we completed this process, we clicked on **Save** (the disk button) to save the relationships and then closed the **Relationships** box.

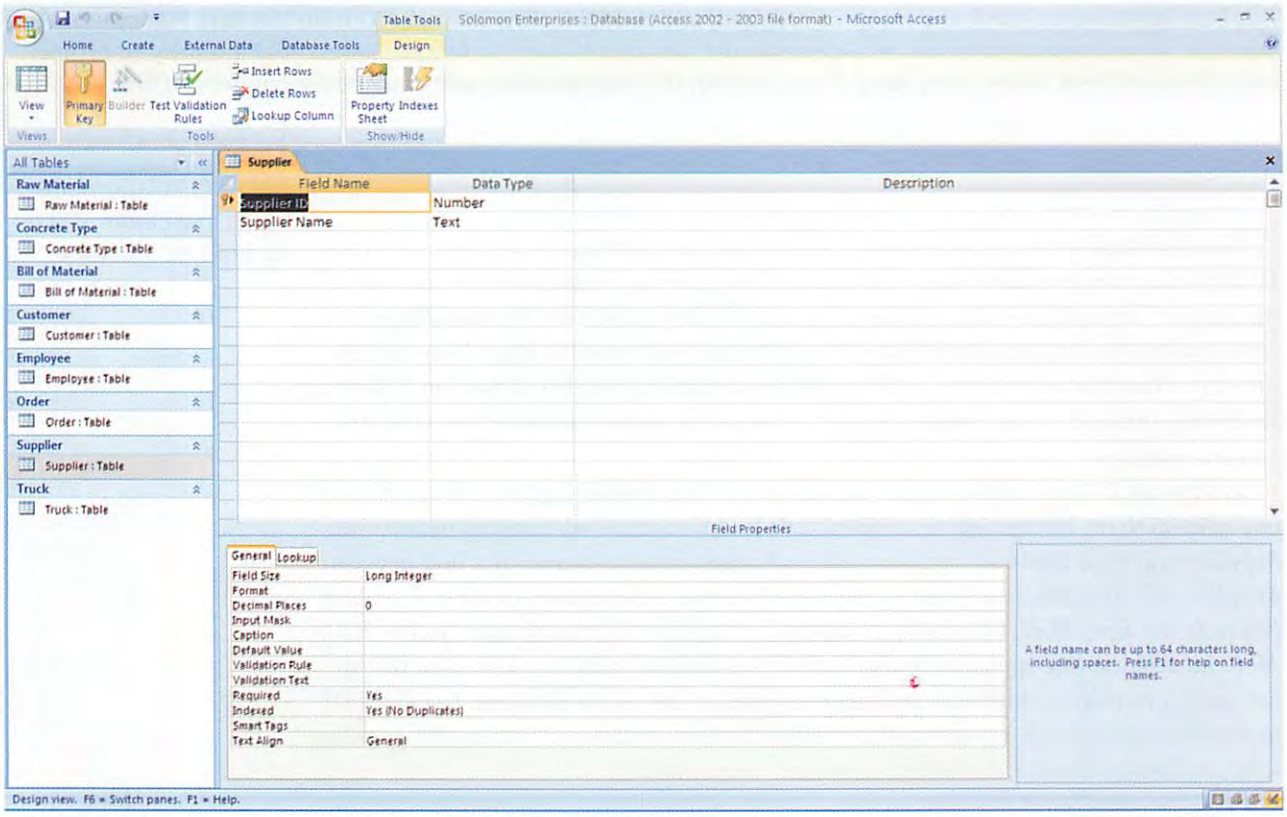
Entering Information into the Solomon Database

Finally, we're ready to begin entering information to populate the tables. We've defined the structure for each relation in the Solomon Enterprises database, and we've defined the relationships among the tables. See Figure J.12 for the Design View of the *Supplier* relation that we created in the same manner as the *Concrete Type* relation.

To enter information, you simply double-click on the table name in the list of tables on the far left (see the upper left screen in Figure J.13 on the opposite page). Does it matter which relation you start with when entering information? It does if you've chosen to enforce referential integrity when you created the relationships among the relations. We can't enter information into the *Raw Material* relation yet because we need to put in a *Supplier ID* for each row and we've not yet entered the *Supplier ID* information into the *Supplier* relation. Therefore, if we try to put a *Supplier ID* into the *Raw Material* relation that is not in the *Supplier* relation, referential integrity will be violated and Access will display an error alert and not allow us to continue.

Figure J.12

Design View of the Structure of the *Supplier* Relation



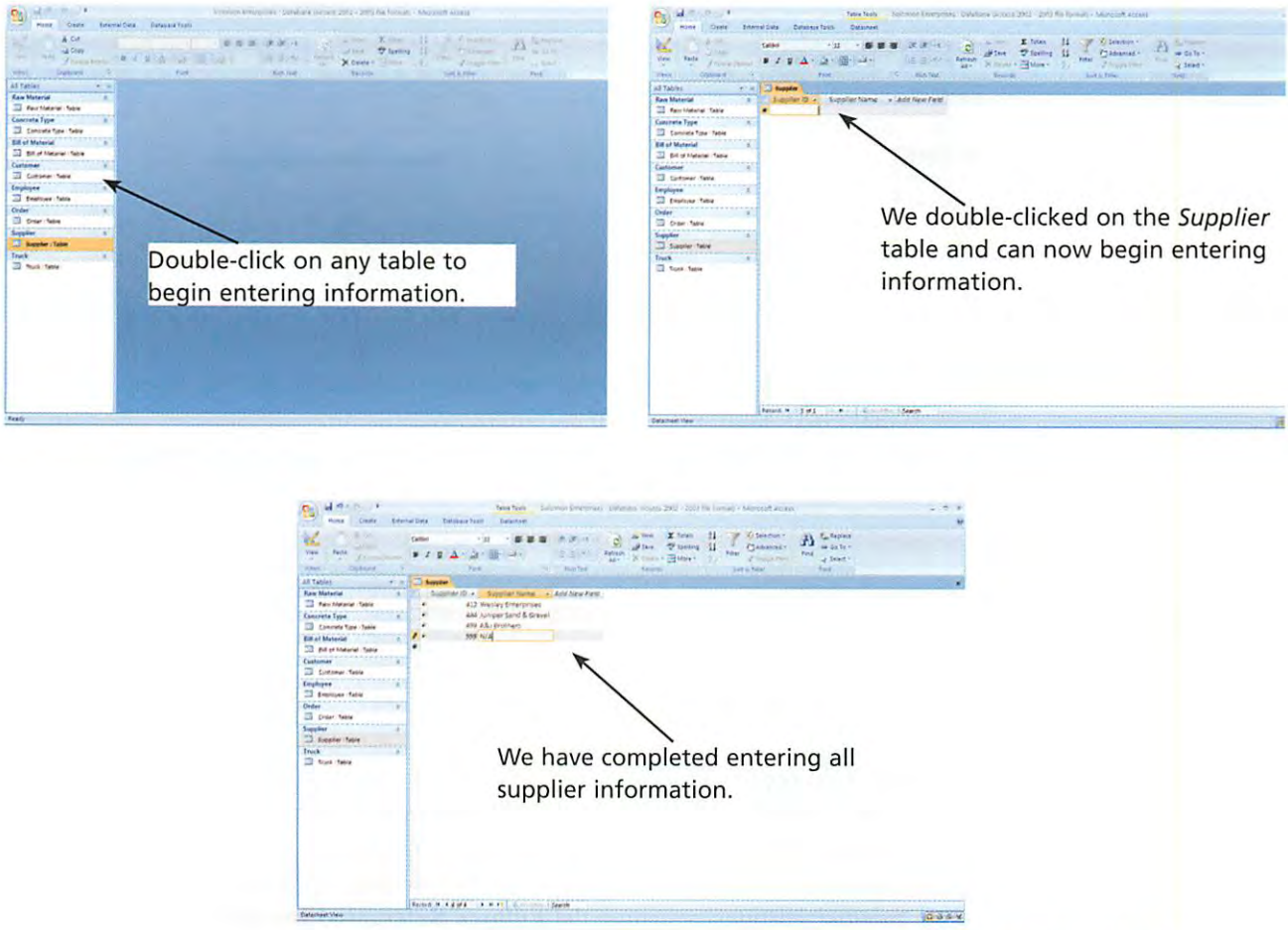
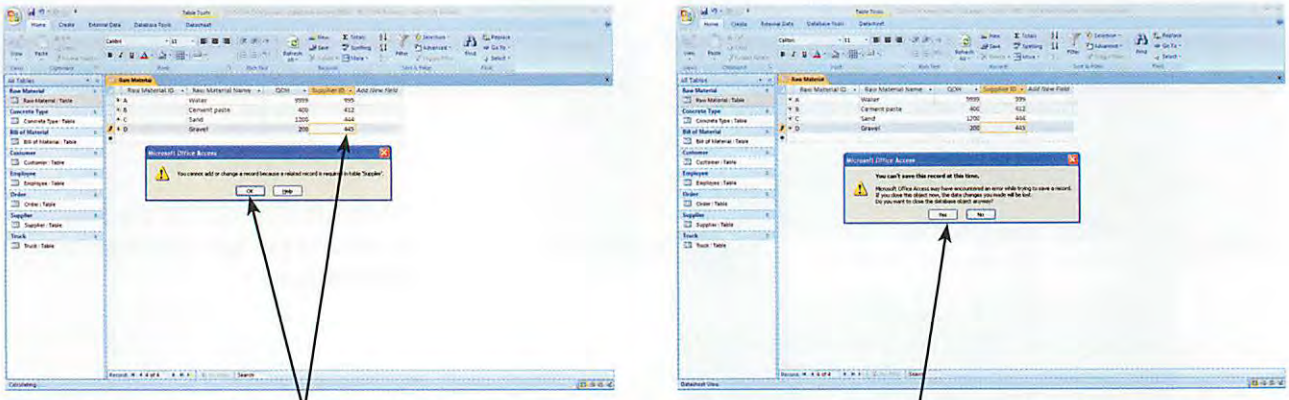


Figure J.13
Entering Information into the *Supplier* Relation

Before entering information into those relations that have foreign keys, you must first enter information into those tables with primary keys that show up as foreign keys in other tables. So, we must complete the information entry for the *Concrete Type, Customer, Employee, Supplier,* and *Truck* relations before populating the *Raw Material, Bill of Material,* and *Order* relations.

In the upper right screen in Figure J.13, you can see that we have opened the *Supplier* relation and are now ready to begin entering information. After typing in the appropriate information into each field, we used the **Tab** key to get to the next field or row. Once we entered all the information (see the bottom screen in Figure J.13), and closed the tab for the *Supplier* table, the information was automatically saved by Access. We can then move on to entering information into the other relations.



Because we entered a *Supplier* ID (445) that doesn't exist in the supplier relation, Access will not allow us to continue.

If you try to close the information entry window, Access will allow you to change the information or save the good information without the bad.

Figure J.14

Encountering an Integrity Error While Entering Information

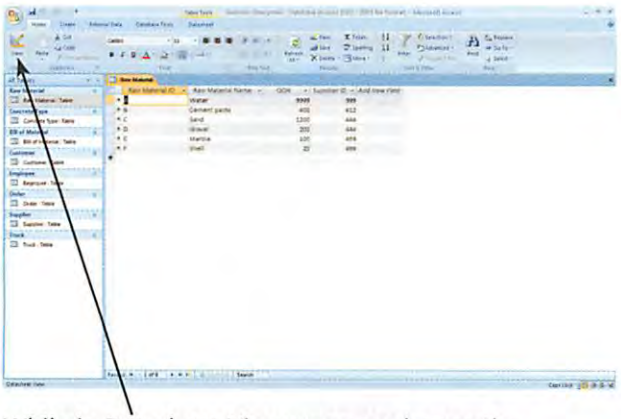
Next, we'll populate the *Raw Material* relation, which has *Supplier ID* as a foreign key. In the screen on the left of Figure J.14 you can see that we've already entered some of the information; however, as the error message shows, we put in an incorrect *Supplier ID* (445 instead of 444). When we established the relationship between the *Raw Material* and *Supplier* relations, we chose the **Enforce Referential Integrity** option, so Access won't allow us to enter a *Supplier ID* in the *Raw Material* relation that doesn't already exist in the *Supplier* relation. That's why we got the dialog box informing us of the problem. If we click on **OK,** Access will give us a chance to fix the problem. If we don't fix the problem and we try to exit this window (see the screen on the right in Figure J.14), Access will alert us with a new box telling us that the incorrect information will not be saved (although the good information will).

Up to this point, we've entered the information into the *Supplier* and *Raw Material* relations. The process is identical for entering information into the other relations. But you do need to keep in mind the order in which you enter information into the other relations. You will need to enter information into those relations first whose primary keys will appear as foreign keys in other relations.

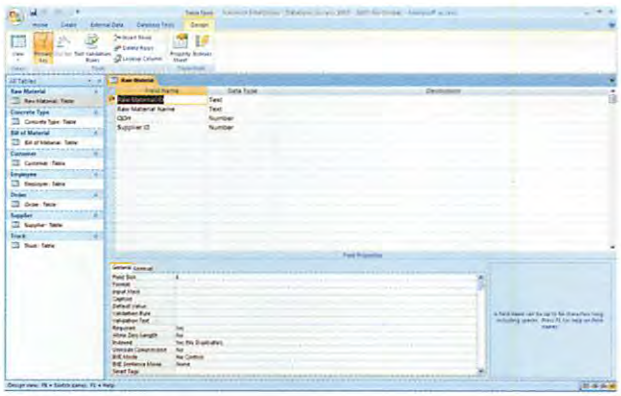
You should now enter all the information into the remaining relations just as we have done for the *Supplier* and *Raw Material* relations. We suggest you enter information into the other relations in the order below:

1. *Concrete Type*
2. *Customer*
3. *Employee*
4. *Truck*
5. *Bill of Material*
6. *Order*

The first four are interchangeable in terms of ordering as are the bottom two.



While in Datasheet View, you can change the structure of a relation only by clicking on **View** and then **Design View**.



In Datasheet View, you can change the structure of a table.

Figure J.15
Changing the Structure of a Relation

CHANGING THE STRUCTURE OF INFORMATION IN RELATIONS

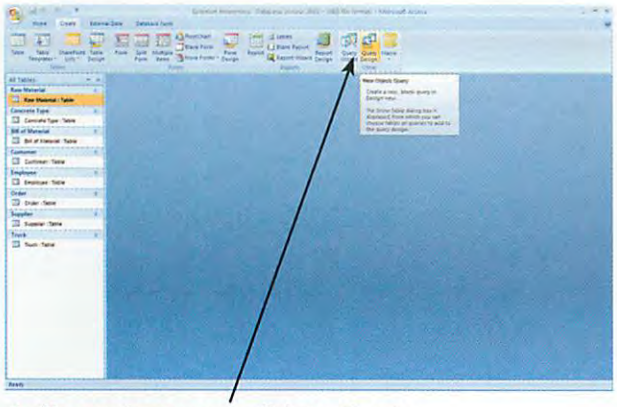
If you want to change the information in any of the relations or the structure of any of the relations, you can do so quite easily. However, you need to be careful about deleting fields that you used to establish relationships between pairs of tables.

The simpler task is to add, change, or delete records in a relation. To add a record, you simply open the relation and add the new information at the bottom. Recall that to open a relation you double-click on its name in the list of tables on the left of the Access screen. To change information in a field, click on the appropriate field and make the change. To delete a record, highlight the record (row) you want deleted and press the **Delete** key on the keyboard.

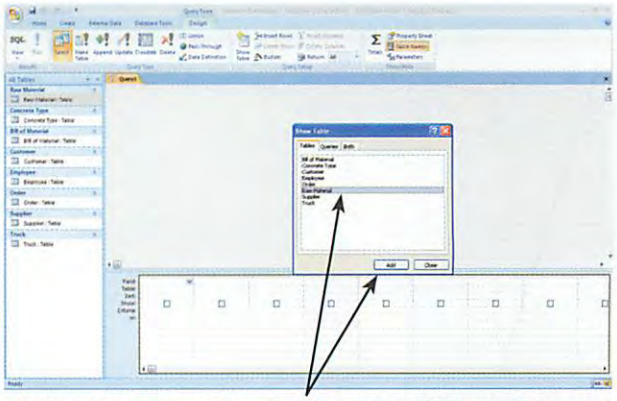
To change the structure of a relation, you'll need to open the Design View for the relation. To do this, first open the relation by double-clicking on its name. What Access will present to you is the table in Datasheet View (the left screen in Figure J.15). To switch to Design View, click on the **View** button and select **Design View.** Access will show you the Design View of the table and you can then make the necessary changes to the structure of that relation (see the right screen in Figure J.15).

Creating a Simple Query Using One Relation

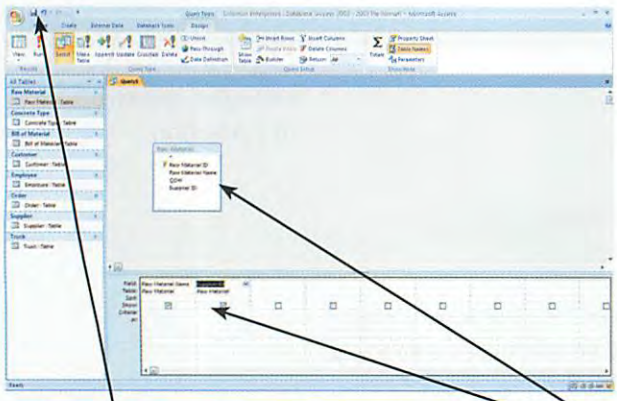
The easiest way to create a query is to use a query-by-example tool. A ***query-by-example (QBE) tool*** helps you graphically design the answer to a question. Suppose, for example, that we wanted to see a list of all the names of all the raw materials and the IDs of the associated suppliers for each raw material. All that information is located in the *Raw Material* relation, making it a relatively simple query requiring the use of only a single relation. Realize, of course, that our Solomon database is small and the query doesn't make much sense since you could easily open the *Raw Material* relation and see the information you want. However, the objective is to demonstrate the use of a QBE tool. In the next section, we'll create a more complicated query. But first let's look at a simple query.



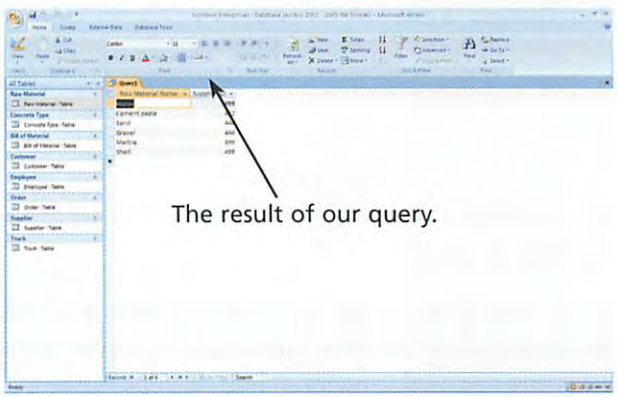
1. To create a query, click on **Create** and then **Query Design.**



2. Select the appropriate table name, click on **Add,** and then close the box.



4. Click on the exclamation point **(Run)** to see the result of query.



3. Now, drag and drop *Raw Material* Name and *Supplier ID* from the *Raw Material* table to the QBE grid

Figure J.16

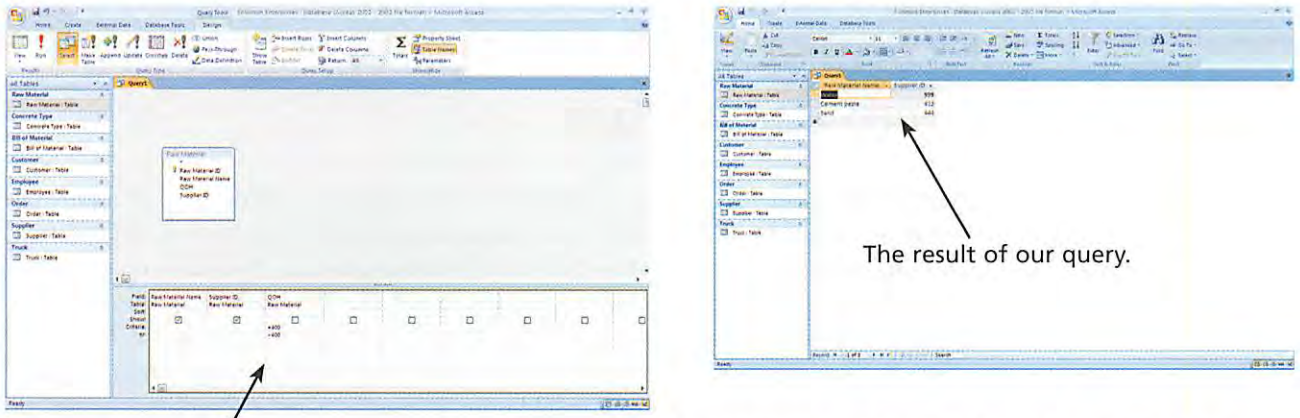
Creating a Simple Query Using One Relation

To create a simple query using only the *Raw Material* relation, perform the following steps (see Figure J.16):

1. Click on **Create** in the menu area and then **Query Design** in the button bar area.
2. In the **Show Table** dialog box, select the appropriate relation name, click on **Add,** and then close the **Show Table** dialog box.
3. Drag and drop the fields that you want in the query results into the QBE grid.
4. Click on the exclamation point (**Run**) in the button bar.

As you can see in Figure J.16, we followed that process by selecting the *Raw Material* relation, and dragging and dropping *Raw Material Name* and *Supplier ID* into the QBE grid. Once we clicked on the exclamation point button, Access returned a list of raw materials along with the ID of the supplier for each item.

If we wanted to be able to use this query at a later time, we could save the query, giving it a unique name such as *Raw Materials and Suppliers.* Then, the next time we need that information, we could simply open up that query in the Datasheet View (the view that shows the information) rather than having to start from scratch creating the query.



To create a conditional query, we add *QOH* from the *Raw Material* table and specify "= 400" or "> 400"in the **Criteria** fields.

Figure J.17
A Conditional Query Using One Relation

SIMPLE QUERY WITH A CONDITION (CONDITIONAL QUERY)

Creating a query with only one relation is pretty simple. So, let's add another requirement to our query. Let's say we wanted to see which raw materials from which suppliers we have in quantities of 400 or more units. This is a conditional query because it will return a result based on some condition. This conditional query requires an extra couple of steps in the process we just outlined.

In Figure J.17, you can see that we again selected only the *Raw Material* relation since it contains all the information we need. However, this time we also dragged and dropped *QOH* into the QBE grid. Within the QBE grid, we provided two important items of information to the query.

The first was to unselect the **Show** parameter. By doing so, we tell Access that we want to use *QOH* as part of the query, but that we don't want it to show in the query result. The second is to enter "**=400**" (without the quotation marks) in the **Criteria** parameter and then "**>400**" (again, without quotes) right below. In doing this, we're telling Access that we want to see only those raw materials that we have in quantities equal to 400 (first **Criteria** line) or greater than 400 (the **or** line). As you can see in the right screen of Figure J.17, Access provides information for only three raw materials, all of which Solomon has quantities on hand of 400 or more units. Also note that the *QOH* field does not show.

If you were looking for suppliers of Gravel, you would put "Gravel" (with or without quotation marks) into the **Criteria** box associated with *Raw Material Name*, but you might be sure to spell it correctly. The capitalization doesn't matter but the right letters (and digits and spaces, if applicable) in the right places do.