Lesson 10: Basic queries using QBE

10.1 Introduction: Using queries to get the information you need

At first glance, it appears that splitting information into multiple tables and relationships creates more of a headache than it is worth. People generally like to have all the information they require on one screen (like a spreadsheet, for instance); they do not want to have to know about multiple tables, foreign keys, relationships, and so on.

Saved queries address this problem. Queries allow the user to join data from one or more tables, order the data in different ways, calculate new fields, and specify criteria to filter out certain records. The important thing to keep in mind is that a query contains no data—it merely reorganizes the data from the table (or tables) on which it is built without changing the "underlying tables" in any way.

Once a query is defined and saved, it can be used in exactly the same way as a table. Because of this, it is useful to think of queries as "virtual tables". Indeed, in the majority of DBMSes, saved queries are called **views** because they allow different users and different applications to have different *views* of the same data.

10.2 Learning objectives

- create different types of queries
- understand how queries can be used to answer questions
- develop a naming convention for queries
- understand the difference between an "updatable" and "non-updatable" recordset

10.3 Exercises

10.3.1 Creating a query

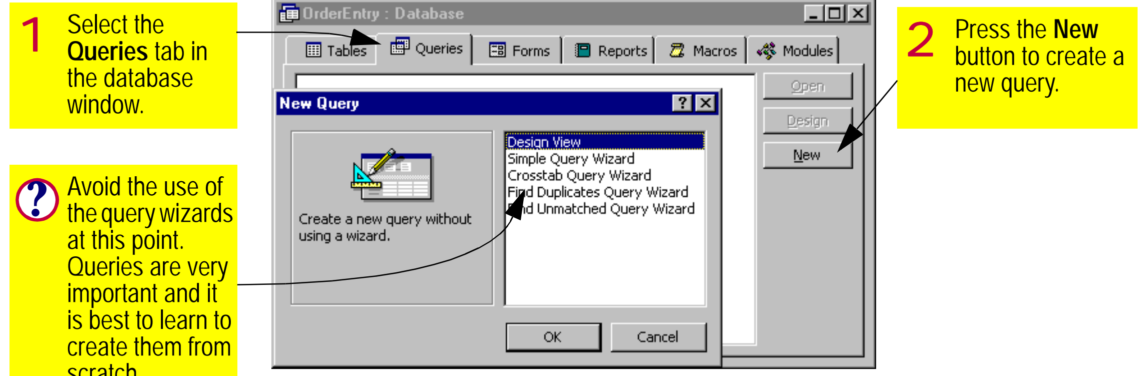
➡ Use the **New** button in the **Queries** pane of the database window to create a new query as shown in Figure 10.1.

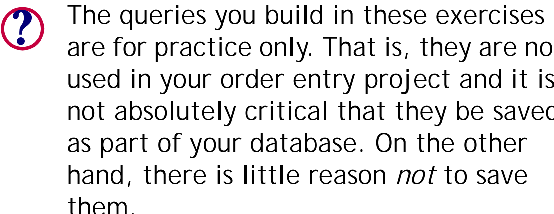
➡ Add the **`Products`** table to the query as shown in Figure 10.2.

➡ Examine the basic elements of the query design screen as shown in Figure 10.3.

➡ Save your query (**Ctrl-S**) using the name **`qryBasics`**.

FIGURE 10.1: Create a new query.





The queries you build in these exercises are for practice only. That is, they are not used in your order entry project and it is not absolutely critical that they be saved as part of your database. On the other hand, there is little reason *not* to save them.

10.3.2 Five fundamental query operations

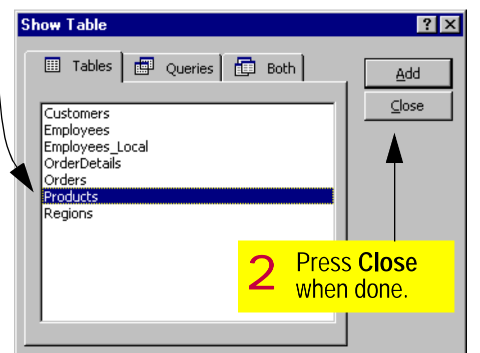
In the following sections, you are introduced to five fundamental query operations: projection, selection, sorting, joining, and calculated fields. The operations are *fundamental* in the sense that they are supported by all relational database systems.

10.3.2.1 Projection

Projecting a field into a query simply means including it in the query definition. The ability to base a query on a subset of the fields in an underlying table (or tables) is particularly useful when dealing with tables that contain some information that is confidential and some that is not confidential.

FIGURE 10.2: Add tables to your query using the “show table” dialog.

1 Add the **`Products`** table to the query by selecting it and pressing **Add** (alternatively, you can simply double-click on the table you want to add).



The “show table” dialog is always available from the **Query** → **Show Table** menu. Alternatively, you can press the “show table” button on the tool bar.

The “show table” window is “modal”—you can not do anything else in a WINDOWS application until a modal window is closed.

For instance, the **`PAY_EMPS`** table you used in Section 8.3.3 contains a field describing the employee’s pay level. If you created a saved query that projected all the employee fields *except* **`EMP_PAY_LE`** and gave users permission to view the saved query instead of the **`PAY_EMPS`** table, then unauthorized users would have no way of viewing sensitive pay information.

Recall that a view called **`SALES_REPS`** was used in Lesson 9 to show only the employees names and ID numbers from the **`PAY_EMPS`** table.

- Perform the steps shown in Figure 10.4 to project the **`ProductID`**, **`Description`**, and **`UnitPrice`** fields into the query definition.
- Select **View** → **Datasheet** from the menu to see the results of the query. Alternatively, press the datasheet icon ( ) on the tool bar.
- Select **View** → **Query Design** to return to design mode. Alternatively, press the design icon ( ) on the tool bar.

10.3.2.2 Selection

You select records by specifying conditions that each record must satisfy in order to be included in the results set. To achieve this in "query-by-example", you enter *examples* of the results you desire into the criteria row.

- Perform the steps shown in Figure 10.5 to answer the following question: *"Which products sell for more than $20?"*
- Select **View** → **Datasheet View** from the main menu to see the results of the query, as shown in Figure 10.6. Since this is a

FIGURE 10.4: Project a subset of the available fields into the query definition.

To project all the fields in the **`Products`** table (including any that might be added to the table after this query is created) drag the asterisk (*) into the query definition grid.

To save time when projecting fields, multi-select (by holding down the **Ctrl** key when selecting) and drag all the fields as a group.

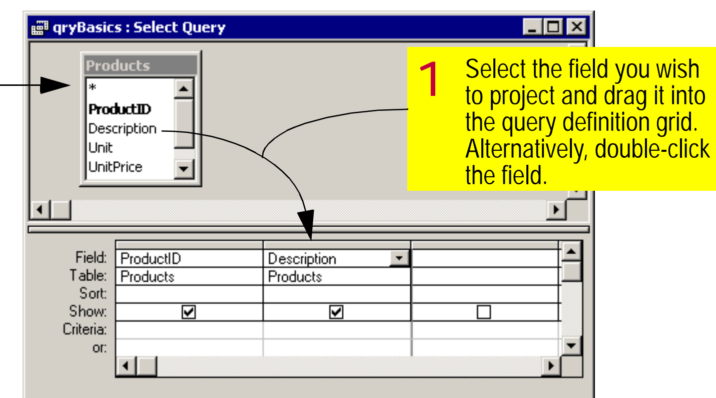
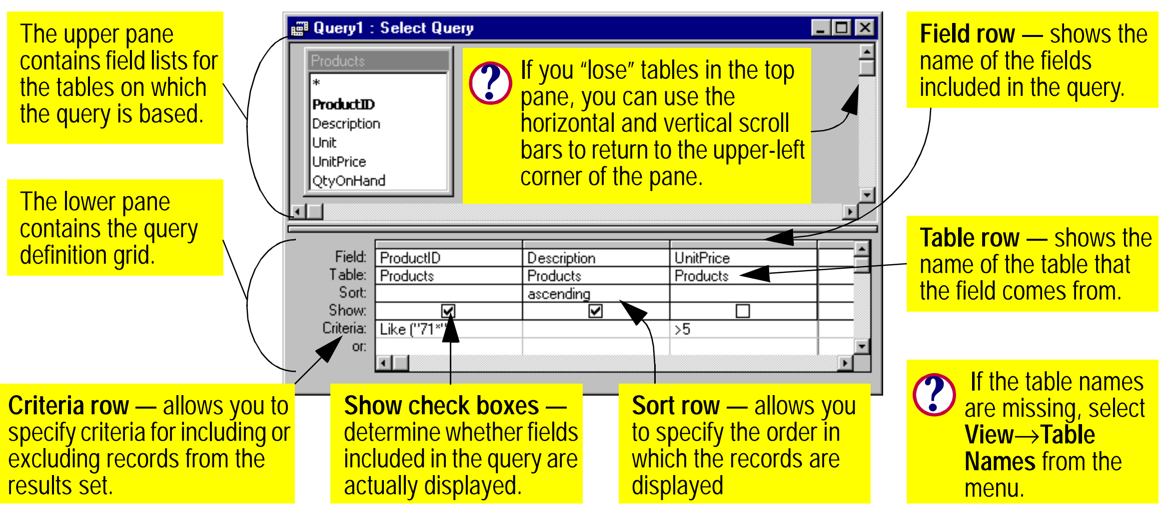


FIGURE 10.3: The basic elements of the query design screen.



select query, the resulting **recordset** contains only those records that satisfy the criteria.

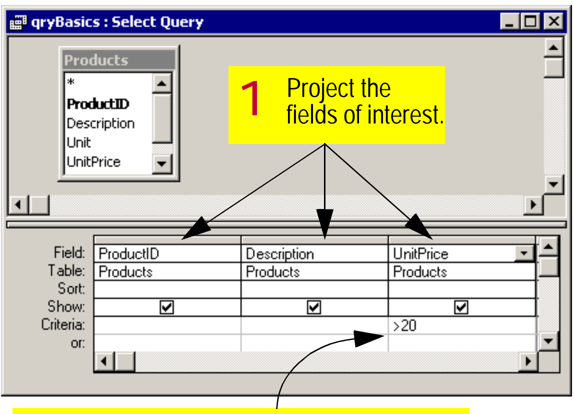
10.3.2.3 Complex selection criteria

It is also possible to create complex selection criteria using **Boolean** (logical) constructs such as AND, OR, and NOT.

➡ Use complex selection criteria to answer more complex questions:

- *"Which products cost less than $5 each?"* (see Figure 10.7)
- *" Which products cost less than $2 each or cost less than $5 for unit sizes greater than one?"* (see Figure 10.8)

FIGURE 10.5: Select records using a “greater than” criterion in the ***UnitPrice*** field.



10.3.2.4 Sorting

When you use a query to sort, you do not change the physical order of the records in the underlying table (that is, you do not sort the table). As a result, different queries based on the same table can display the records in a different sequence.

➡ Set the **Sort** row to sort the results of your query by **`Description`** in ascending (A→Z) order (see Figure 10.3).

? When multiple fields are used in a query to sort (e.g., sort first by last name, then by first name), fields on the left take precedence over fields on the right.

? Since a query is never used by itself to display data to a user, you can move the fields around within the query definition to get the desired sorting precedence. You then reorder the fields in the form or report for presentation to the user.

10.3.2.5 Joining

In Lesson 7, you were advised to break you information down into multiple tables with relationships between them. In order to put this information back together in a usable form, you use a join query.

➡ Save and close **`qryBasics`**.

➡ Open the relationships window and ensure you have a relationship declared between **`Customers`** and **`Regions`**. If not, declare one and ensure referential integrity is enforced (review Section 7.3 as required).

FIGURE 10.6: View the results of a select query.

The "datasheet" icon can also be used to switch between the different viewing modes of a query.

1 While in query design mode, select **View** → **Datasheet View** to see the results of the query.

2 Examine the resulting recordset to verify that the query returns the correct products.

qryBasics : Select Query

ProductID	Description	UnitPrice
51 5012	Water jug, s.s. w/ice guard, 2 litre	$23.50
74 6881	Lobster set	$22.00
82 25160B	Coffee grinder, 8" black	$25.00
82 25160W	Coffee mill, 8", white	$25.00
82 300128	Electric pepper mill, black	$25.00
82 300135	Electric pepper mill, w/light	$25.00
82 3052	Wine bottle pepper mill, 14 1/2"	$37.00
83 7505	Sharpener, Chef Choice, white	$78.00
91 354142	Pervenche oval roasting dish	$22.00
91 500304	Cobalt oval cassarole, 1.4 litre	$21.50
92 8DO2A	Cast iron dutch oven, 5 qt.	$30.50

- Create a new query called **qryJoin** based on the **Customers** and **Regions** tables.
- Project **Regions.RegionName**, **Customers.CustName**, and **Customers.City** as shown in Figure 10.9.

When a query is based on more than one table, it is often the case that certain field names are used in multiple tables (e.g., **RegionCode** is used in both **Customers** and **Regions**). To eliminate ambiguity, the ***<table name>.<field name>*** notation is used to refer to fields.

FIGURE 10.7: Complex queries using ANDed criteria.

1 To “AND” criteria, put them in the same row. This selects products for which **`UnitPrice`** < 5 AND **`Unit`**=“ea”.

If no comparison operator (such as “greater than”) is provided, “equals” is assumed.

Criteria for text fields must be in quotation marks. For non-text fields, do not use quotation marks.

Field:	ProductID	Description	UnitPrice	Unit
Table:	Products	Products	Products	Products
Sort:				
Show:	☑	☑	☑	☑
Criteria:			<5	"ea"
or:				

- Click on the column selector for **`Regions.RegionName`** and drag the field to the far right of the query definition grid (this is to show you how easy it is to re-order your fields once they have been added).

- Switch to datasheet view and notice that information from both tables is shown.

If you neglected to populate the **`Customers.RegionCode`** field in Section 8.5, the result of your join query will be empty (no records). A join matches each record on the “many” side with its corresponding record on the “one” side. If **`Customers.RegionCode`** is NULL for all customers, no records are

FIGURE 10.8: Complex queries using ORed criteria.

1 To “OR” criteria, put them on different rows. This example selects products for which **UnitPrice** < 2 OR (**UnitPrice** < 5 AND **Unit** is not equal to “ea”).

qryBasics : Select Query

Product ID	Description	Unit price	Unit
57 551	S.S. salad serve	$3.15	2PC
74 6102	Deluxe measuri	$3.50	4PC
74 6814	Rubber scraper	$0.80	EA
78 6932	Fondue fuel, 50(	$1.75	EA

qryBasics : Select Query

Products: *, ProductID, Description, Unit, UnitPrice

Field:	ProductID	Description	UnitPrice	Unit
Table:	Products	Products	Products	Products
Sort:				
Show:	☑	☑	☑	☑
Criteria:			<2	
or:			<5	Not "ea"

Care must be taken with complex selection criteria to ensure the ANDs and ORs are associated the way you think they are associated. The expression X OR (Y AND Z) is very different from (X OR Y) AND Z.

returned since there is no record in the **Regions** table with a **RegionCode** = NULL.

10.3.3 Using queries to edit records

Once ACCESS knows the **RegionCode** of a customer, it can uniquely identify the region to which the customer has been assigned. This allows us to show the more user-friendly region name instead of the single-letter **RegionCode** field. The result is similar to the “monolithic” table design discussed in Section 7.1.1. However, there are some important differences, as you will see in the following exercises.

10.3.3.1 Editing a record on the "many" side

- Add **Customers.RegionCode** to your query. Ensure you add **Customer.RegionCode**, not **Regions.RegionCode**.
- Switch to datasheet view, as shown in Figure 10.10.
- Change the **RegionCode** value for ROSCH DRY GOODS from "E" to "K" (assume the firm is being transferred to the special "key accounts" region). When you click on the record selector to save the record, you will notice that the value of the **RegionName** field changes automatically.

10.3.3.2 Edit a record on the "one" side

- Click on one of the "Key" values in the **RegionName** field and change it to a more descriptive value "Key Accounts", as shown in Figure 10.11.
- Save the record and notice how the change propagates to all key account regions.

FIGURE 10.10: Use a join query to make a change to a record on the "many" side of the relationship.

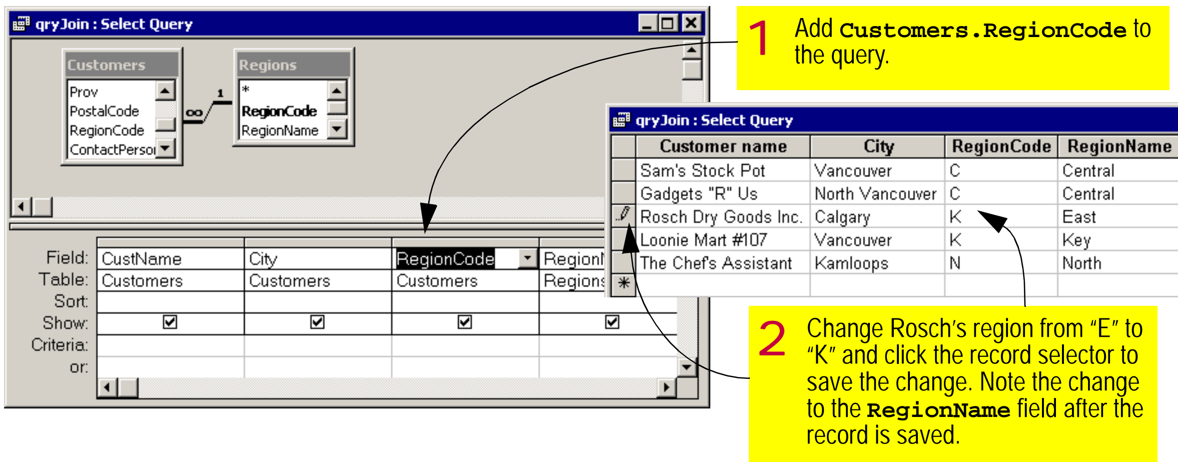
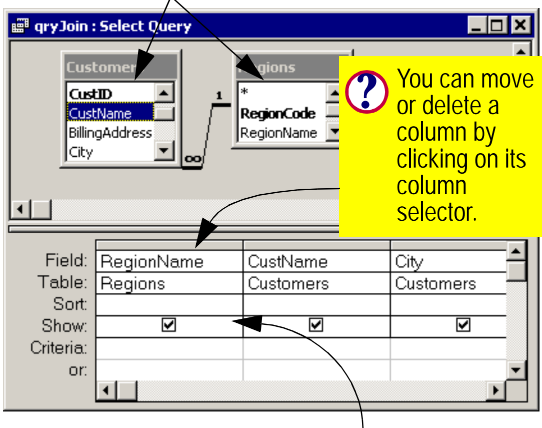


FIGURE 10.9: Create a query that joins **`Customers`** and **`Regions`**.

1 Bring **`Customers`** and **`Regions`** into the query. Note that the relationship between the tables is inherited from the relationship window.



2 Project fields from both tables into the query definition.

Normalized tables and join queries provide a very efficient means of administering data. In this exercise, it is important to realize that the **`RegionName`** field in the query is a direct "window" into the **`Regions`** table. Since the name of the region is only stored in the database once, it only needs to be changed once.

FIGURE 10.11: Use a join query to make a change to a record on the "one" side of the relationship.

qryJoin : Select Query

Customer name	City	RegionCode	RegionName
Sam's Stock Pot	Vancouver	C	Central
Gadgets "R" Us	North Vancouver	C	Central
Beach Dry Goods Inc.	Calgary	K	Key Accounts
Loonie Mart #107	Vancouver	K	Key
The Chef's Assistant	Kamloops	N	North

1 Change the name of the "Key" regions to "Key Accounts". Observe the values in the **`RegionName`** field for other customers in the region when the record is saved.

10.3.4 Using queries to add records

When adding records to a table that is on the "many" side of a relationship, it is often helpful to use a join query to provide *feedback* to the user during data entry.

➨ Create a new query for adding customers called **`qryCustomerAdd`**. Project the following fields into the query definition: **`Customers.*`**, **`Regions.RegionName`**.

Since the purpose of this query is to add records to the `Customers` table, it is clear that you need to project *all* of the customer fields. The asterisk (*) provides a convenient means of doing this, even if the fields in the table change over time.

➡ Switch to datasheet view and add a new customer record (make one up).

➡ Note that when the `RegionCode` is specified, the name of the region is “looked-up” and filled in automatically. This is what is meant by feedback—the additional region information helps you determine whether you have entered the correct `RegionCode`.

10.4 Discussion

10.4.1 Naming conventions for database objects

As discussed in the section on field names in Section 5.4.5.1, there are relatively few naming restrictions for database objects in ACCESS. However, a clear, consistent method for choosing names can save time and avoid confusion later on.

Although there is no hard and fast naming convention required for the project, the following guidelines should be appended to those introduced in Section 5.4.5.1:

- **Use meaningful names** — An object named `Table1` does not tell you much about the contents of the table. Furthermore, since there is no practical limit to the length of the names, you should not use short, cryptic names such as `s96w_b`. As the number of objects in your database grows, the time spent carefully naming your objects will pay itself back many times.
- **Give each type of object a distinctive prefix (or suffix)** — This is especially important in the context of queries since tables and queries cannot have the same name. For example, you cannot have a table named `Orders` and a query named `Orders`. However, if all your query names are of the form `qryOrders`, then distinguishing between tables and queries is straightforward.

Table 10.1 shows a suggested naming convention for ACCESS database objects (you will discover what these objects are in the course of doing the tutorials).

10.4.2 Using queries to populate tables on the “many” side of a relationship

In Section 10.3.4, you added a record to the `Customers` table to demonstrate the automatic

TABLE 10.1: A suggested naming convention for ACCESS database objects.

Object type	Prefix	Example
table	(none)	OrderDetails
query	qry	qryBackOrders
parameter query	pqry	pqryItemsInOrder
form	frm	frmOrders
sub form	sfrm	sfrmOrderDetails
report	rpt	rptInvoice
sub report	srpt	srptInvoiceDetails
macro	mcr	mcrOrders
Visual Basic module	bas	basUtilities

lookup feature of ACCESS. A common mistake when creating queries for adding or modifying data on the “many” side of a relationship is to forget to project the foreign key of the table you intend to populate.

For example, faced with two tables containing the **RegionCode** field, you might project the one from wrong table (the “one” side) into your query definition. To illustrate the problem, do the following:

- Create a new join query called **qryCustomerAdd2** that projects **Regions.RegionCode** instead of **Customers.RegionCode**, as shown in Figure 10.12.
- Attempt to add a new record. You will be unable add anything and will get an error message in the status bar at the bottom of the screen.

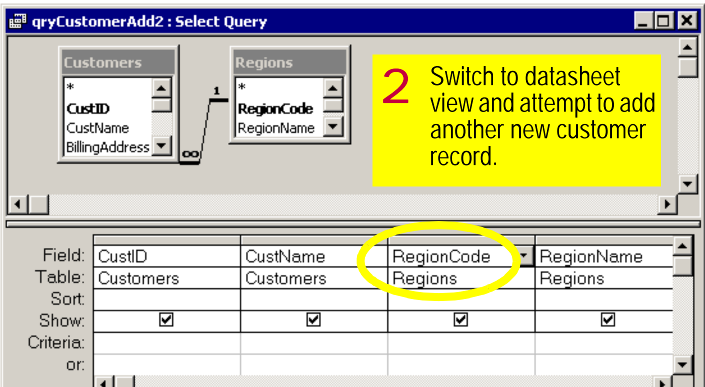
Even if you were allowed to add a record using this query, the results would be dangerous. In **qryCustomerAdd2**, the **RegionCode** field is bound to the **Regions** table instead of the **Customers** table. Entering a value of “N” into the field simply overwrites the current value of **Regions.RegionCode**.

10.4.3 Non-updatable recordsets

Another problem that sometimes occurs when creating join queries is that the query is not quite right in some way. In such cases, ACCESS will allow you to view the results of the query, but it will not allow you to change the data in any way.

In this section, will look at a nonsensical query that results from an incompletely specified

FIGURE 10.12: Create a data-entry query without a foreign key.



relationship. As you will probably discover, however, there are many different ways to generate nonsensical queries.

Show me (lesson10-1.avi)

- Create a new query called **qryNonUpdate** based on the **Customers** and **Regions** tables.
- Delete the **RegionCode** relationship and project a couple of fields from both tables, as shown in Figure 10.13.
- View the results of the query.

The result of this query is known as a **Cartesian join** (or cross product)—every customer is combined with every region regardless of the value of the **RegionCode** field. ACCESS recognizes that this is not a standard join query and designates the recordset as **non-updatable**.

FIGURE 10.13: Create a non-updatable recordset.

qryNonUpdate : Select Query

Customers: BillingAddress, City, Prov, PostalCode

Regions: *, RegionCode

Join Properties
Delete

Field:	CustID	CustName	Regi
Table:	Customers	Customers	Custo
Sort:			
Show:	☑	☑	
Criteria:			
or:			

1 Project fields from both tables into the query definition.

2 Right-click on the relationship and select **Delete** from the context menu. Note that this deletes the relationship from this query only.

3 Attempt to change a value in the recordset.

qryNonUpdate : Select Query

Customer ID	Customer name	Customers.Re	Regions.Regio	RegionName
1	Sam's Stock Pot	C	C	Central
2	Loonie Mart #107	K	C	Central
3	Rosch Dry Goods Inc.	K	C	Central
4	Gadgets "R" Us	C	C	Central
5	The Chef's Assistant	N	C	Central
1	Sam's Stock Pot	C	E	East
2	Loonie Mart #107	K	E	East
3	Rosch Dry Goods Inc.	K		
4	Gadgets "R" Us	C		
5	The Chef's Assistant	N		
1	Sam's Stock Pot	C		
2	Loonie Mart #107	K		
3	Rosch Dry Goods Inc.	K		

Record: 1 of 30

Note the absence of the asterisk and the "new record" row. These are sure signs that the recordset is non-updatable.

Later, when building forms, you may accidentally base a form on a non-updatable recordset. You then may spend a great deal of time trying to get the form to work when the real problem is the underlying query. A quick check for a "new record" row in all your new queries can save time and frustration later on.

10.5 Application to the assignment

- Create a query called **`qryOrderDetails`** that joins the **`OrderDetails`** and **`Products`** tables. When you enter a valid **`ProductID`**, the information about the product (such as name, quantity on hand, and so on) should appear automatically.

Due to the referential integrity constraints you specified in Lesson 7, you will not be able to save any records you use to test **qryOrderDetails** until you have a corresponding **OrderID** in the **Orders** table. At this point, you can simply hit the **Esc** key to discard the test data without saving it.

If the name of the product does not appear when you enter a valid **ProductID**, you have used the wrong **ProductID** field. Review Section 10.4.2 and ensure you understand (and fix) the error before continuing.

- Create a query to show additional information about the customer who placed the order when looking at data in the **Orders** table.

- Enter the first order (or at least a handful of order details from the first order) into your system by typing the information directly into tables or queries. Doing this correctly is going to require some thought.

Adding a new order involves creating a single **Orders** record and several **OrderDetails** records. You must also consult the **Products** table to determine the quantity of each item to ship (you cannot ship product that you do not have in inventory), and determine the default selling price of each good.

HINT: Much of the effort involved in switching between tables is eliminated if you use the **qryOrderDetails** query you created above for this exercise.

Entering orders into your system will be much less work once the input forms and event-driven programs are in place. The goal at this point is to get you thinking at a very specific level about the order entry process and what needs to be automated.