

17.1 Introduction: Queries that change data

All of the queries that you have created to this point have been variations of “select” queries. Select queries are used to display data, but do not actually change the data in any way. In this lesson, you are going to learn about action queries.

17.1.1 What is an action query?

Action queries are used to change the data in existing tables or make new tables based on the query's results set. The primary advantage of action queries is that they allow you to modify a large number of records without having to resort to writing VISUAL BASIC programs.

ACCESS provides four different types of action queries:

1. **Make table** — creates a new table based on the results set of the query;
2. **Append** — similar to a make-table query, except that the results set of the query is appended to an existing table;
3. **Update** — allows the values of one or more fields in the results set to be modified; and,
4. **Delete** — deletes all the records in the results set from the underlying table.

Since the operation of all four types of action queries is similar, we will focus on update and make-table queries in this tutorial.

17.1.2 Why use action queries?

The **Products** table includes a field called **QtyOnHand** that stores the inventory level for each product. Note, however, that this information already exists in the database. In principle, one could calculate the quantity on hand for each product by summing all the input transactions (shipments from suppliers) and subtracting all the output transactions (shipments to customers).

Given the emphasis to this point on minimizing the amount of redundancy and dependency in our databases, it may seem odd to store **QtyOnHand** when it can (in principle) be calculated whenever it is needed. The problem is that as the number of transactions grows large, it may become infeasible from a performance point of view to continuously recompute the inventory level for each product.¹

Instead, we use “master” fields such as **QtyOnHand** to store status information about transactions. The advantage of storing this type

of summary information is that it is available immediately. The disadvantage is that great care must be taken to insure that it is consistent with the information in the transactions that it summarizes.

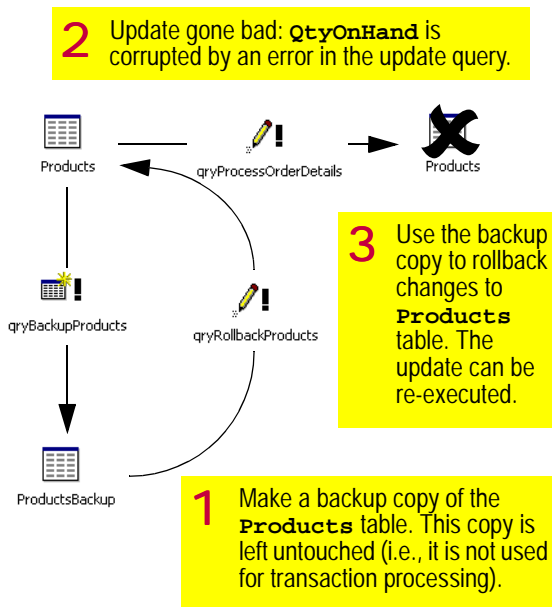
17.1.3 Rolling back updates

Since action queries permanently modify the data in tables, and since there is no undo feature for action queries, it is a good idea to create a mechanism to undo the effects of the query before executing it. Figure 17.1 shows the basic elements of the simple rollback feature that you are going to implement for your project.



The rollback in Figure 17.1 is “simple” because it only allows you to restore the **Products.QtyOnHand** field to its state when the **ProductsBackup** table was created. To implement a more realistic rollback feature, you would have to update the backup copy periodically.

FIGURE 17.1: Using action queries to enable a simple rollback feature.



This rollback feature will allow you to test your action queries without having to worry about corrupting the data in the **Products** table.



Rolling back transactions is so important that industrial strength databases (and even ACCESS) have a built-in infrastructure

¹ In practice, one must also account for changes to inventory levels (e.g., theft, breakage, spoilage) that do not appear as transactions. Hence the need to periodically count the inventory and reconcile any discrepancies.



for automatic commit and rollback (see the on-line help system). The exercises in this section are intended as an introduction to action queries, not database recovery.

17.2 Learning objectives

- understand the difference between action queries and select queries
- make a backup copy of a table using an action query
- undo (rollback) an action query once it has been executed
- update only certain records in a table
- create a button on a form that executes an action query when pressed

17.3 Exercises

17.3.1 Using a make-table query to create a backup

One way to make a backup copy of the **Products** table is to use a make-table query.

(lesson17-1.avi)

- Create a select query based on the **Products** table and save it as **qryBackupProducts**.

- Project the asterisk (*) into the query definition so that all the fields are included in the results set.
- While still in query design mode, select **Query** → **Make Table** from the main menu and provide a name for the target table (e.g., **ProductsBackup**) as shown in [Figure 17.2](#).
- Switch to datasheet mode to preview the action.

Switching to datasheet mode does not execute the action query. Instead, it shows you the records that will be used when the action is run. In the case of a make-table query, the datasheet shows you the records that will be used to make the new table.

- Select **Query** → **Run** from the main menu to execute the action query, as shown in [Figure 17.3](#). Respond to warning boxes as required.
- If you switch to the database window, you will notice that the new make-table query has a different icon than the select queries. If you click on the **Tables** tab, you will notice the new **ProductsBackup** table.



FIGURE 17.2: Use a make-table query to back up an existing table

The screenshot shows the Microsoft Access interface. The 'Query' menu is open, and 'Make-Table Query...' is selected. The 'Products' table is visible in the left pane. The 'Make Table' dialog box is open, showing 'ProductsBackup' as the table name and 'Current Database' as the target.

1 Project all fields (*) into the query definition.

2 Transform the **Select** query into a **Make Table** query

3 Provide a name for the new (target) table.

⚠ If the target table already exists, the make-table query will overwrite it.

17.3.2 Using an update query to rollback changes

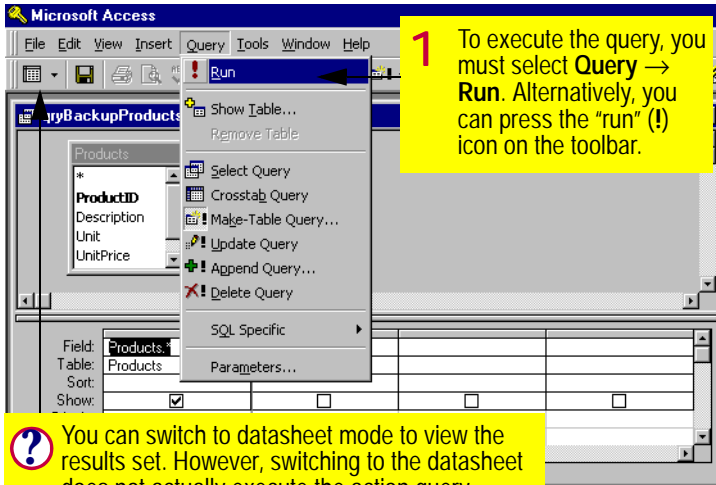
Having a backup table is not much use without a means of using it to restore the data in your original table. In this section, you will use an update query to replace the **QtyOnHand** values

in your **Products** table with pristine values from your **ProductsBackup** table.

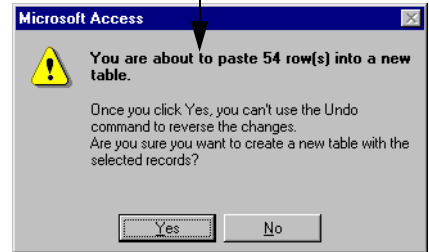
(lesson17-2.avi)

- ➔ Create a new query and add both the **Products** and the **ProductsBackup** tables. Save it as **qryRollbackProducts**.

FIGURE 17.3: Run the make-table query.



? The warning box reminds you that you are about to make permanent changes to the data in the database.



➔ Since no relationship exists between these tables, create an *ad hoc* relationship within the query as shown in Figure 17.4.

➔ Select **Query** → **Update** from the main menu. Note that this results in the addition of an **Update To** row in the query definition grid.

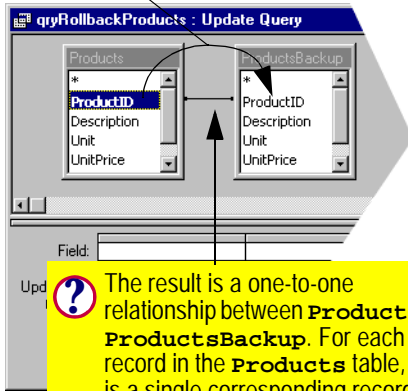
➔ Project **QtyOnHand** into the query definition and fill in the **Update To** row as shown in Figure 17.5.

Now is a good point to stop and interpret what you have done so far:

1. By creating a relationship between the **Products** table and its backup, you are joining together records from both tables that share the same **ProductID**.

FIGURE 17.4: Create an *ad hoc* relationship between the table and its backup copy.

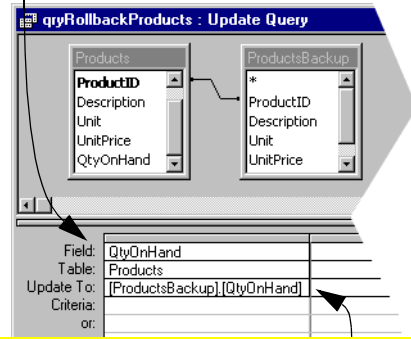
1 Drag the key on to its counterpart in the backup table.



The result is a one-to-one relationship between **Products** and **ProductsBackup**. For each record in the **Products** table, there is a single corresponding record in the **ProductsBackup** table.

FIGURE 17.5: Fill in the **Update To** field.

1 Indicate that you want to update **Products.QtyOnHand** to **ProductsBackup.QtyOnHand**.



Since there is more than one **QtyOnHand** field, you must use the <table name>.<field name> syntax to eliminate any ambiguity.

- By projecting **Products.QtyOnHand** into the query, you are making it the target for the update. No other field is modified by the action query.
- By setting the **Update To** field to **ProductsBackup.QtyOnHand**, you are telling ACCESS to replace the contents of **Products.QtyOnHand** with the contents of **ProductsBackup.QtyOnHand**.

This update query can be used at any time to restore the inventory levels in the **Products** table to the values they contained when the **ProductsBackup** table was created.

Although it would be a simple matter to use the backup to replace the contents of all the fields in the **Products** table (e.g., **Description**, **UnitPrice**, and so on),



QtyOnHand is the only field that can be corrupted by the update query you will create to process orders.

17.3.3 Using an update query to process the order

Now that you have an infrastructure for rolling back any errors, you can continue with the task of creating an action query to process orders.

(Lesson17-3.avi)

- Create an update query based on the **OrderDetails** and **Products** tables and save it as **qryProcessOrderDetails**.
- Set the **Update To** field to **[QtyOnHand]-[QtyShipped]**, as shown in [Figure 17.6](#).
- Execute the query.

Note that the query in [Figure 17.6](#) updates the **QtyOnHand** field once for every value in the **OrderDetails** table. Consequently, if you enter a new order and run the query, you will end up subtracting all the items shipped in the new

FIGURE 17.6: Create an action query to process *all* orders.

1 Indicate that you want to replace existing values of **QtyOnHand** with **QtyOnHand - QtyShipped**.

2 Use datasheet mode to preview the values that will be changed by the update query.

Note that if you do not include the square brackets, ACCESS will interpret "QtyOnHand" as a literal string rather than a field name.

Quantity on hand	
65	
20	
7	
0	
8	
11	
0	
*	

Record: 1 of 7

order plus all items shipped in previous orders. Clearly, this is incorrect.

A convenient means of avoiding this problem is to have the action query process items from one order only. To do this, add a parameter to your action query.

- Open your **frmOrders** form so that it appears in the expression builder under “loaded forms”.
- Switch back to the query and use the builder as shown in **Figure 17.7** to enter a parameterized criterion for the **OrderID** field. The criterion should pull its value from the order form.

FIGURE 17.7: Create an action parameter query to process a single order only.

1 Open the order form so that the query can be tested.


3 Use the expression builder to create a parameterized criterion for **OrderID**.

The screenshot shows the Microsoft Access interface. On the left, a query design grid for 'qryProcessOrderDetails : Update Query' is visible. It has two tables: 'Products' and 'OrderDetails'. The 'Criteria' row for the 'OrderID' field in the 'OrderDetails' table contains the expression 'Orders![OrderID]'. In the center, the 'Expression Builder' dialog box is open. The 'Forms!' dropdown is set to 'frmOrders'. The '<Field List>' pane shows 'OrderID' selected. The 'Criteria' row in the design grid is highlighted with a yellow box and labeled '2'. The 'Expression Builder' dialog is highlighted with a yellow box and labeled '3'. A yellow box labeled '1' points to the query design grid.

2 Project the **OrderID** field, but leave its **Update To** row blank.




- ➔ On the order form, navigate to an order that contains some order details.
- ➔ Run the query and verify that the update has been performed successfully. Since the action query is making changes to your data, you will see a number of warning dialogs, as shown in [Figure 17.8](#).

 Once an action query is created, it has more in common with programs written in VISUAL BASIC than standard select queries. Double-clicking an action query executes it and, apart from the warning messages, there is no visible indication that the action has been performed.

Since the action query is also a parameter query, you should rename it to be consistent with the parameter query naming convention used in [Lesson 16](#).

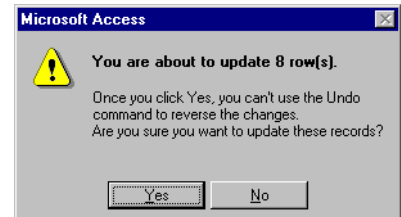
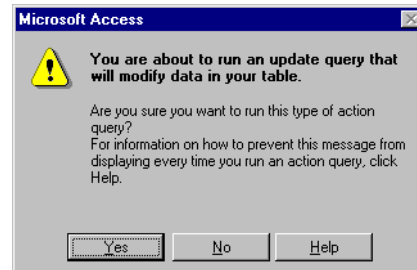
- ➔ In the database window, right-click on **qryProcessOrderDetails**, select **Rename**, and change the prefix to "pqry".

 You cannot rename a database object if it is open. If your query is open, close it before attempting to rename it.

17.3.4 Rolling back changes

While testing the **qryProcessOrderDetails** query, your exuberance may lead you to execute it more than once. To return the **Products** table to its state before any updates, all you need to do is run your rollback query.

FIGURE 17.8: Running an action query generates warning messages.




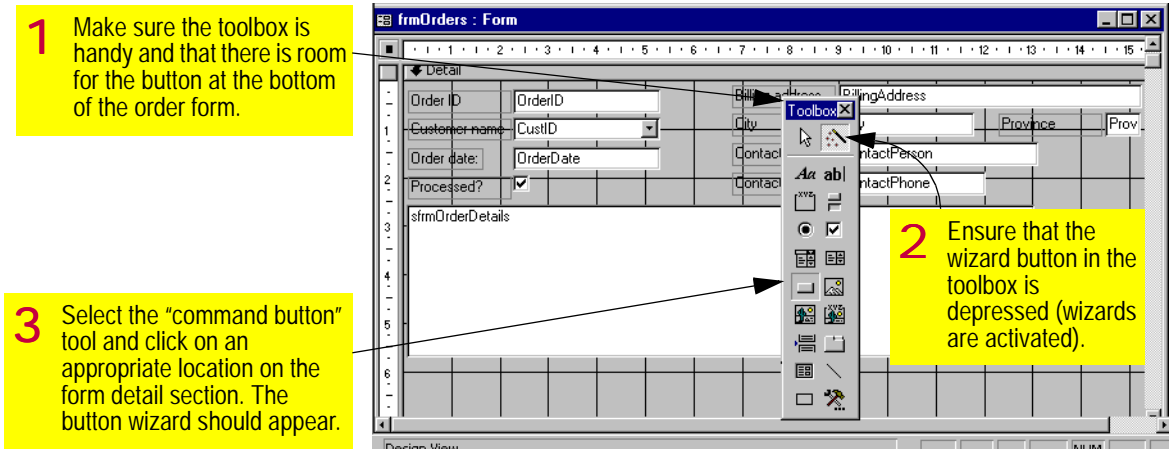
 The number of records to be processed should correspond to the number of order details in the currently visible order.



FIGURE 17.9: Add a button to the form using the button wizard (part 1).



- ➔ Run `qryRollbackProducts` by double-clicking its icon in the database window.

17.3.5 Attaching action queries to buttons

As a designer, you should not expect your users to understand your query naming convention, rummage through the queries listed in the database window, and execute the queries that need to be executed.

A better approach is to create buttons on forms and “attach” the action queries to the buttons. When the button is pressed, the query is

executed. Although we have not yet discussed buttons (or **events** in general), the button wizard makes the creation of this type of form object straightforward.

(lesson17-4.avi)

- ➔ Switch to the design view of `frmOrders` and add a button as shown in [Figure 17.9](#).
- ➔ Attach the `pqryProcessOrderDetails` query to the button as shown in [Figure 17.10](#).



FIGURE 17.10: Add a button to the form using the button wizard (part 2)

4 Buttons can be created to perform many different actions in ACCESS. The button wizard organizes these actions into categories. Select **Miscellaneous** and **Run Query**.

5 The wizard lists all the available queries (including non-action queries). Select the one that you want to execute when the button is pressed.

➔ Provide a caption and a name for the button as shown in [Figure 17.11](#).

➔ Switch to form view. Press the button to run the query (alternatively, use the shortcut key by pressing **Alt-P**) as shown in [Figure 17.12](#).

17.4 Application to the project

➔ Ensure you have implemented and tested the action queries described in this lesson.



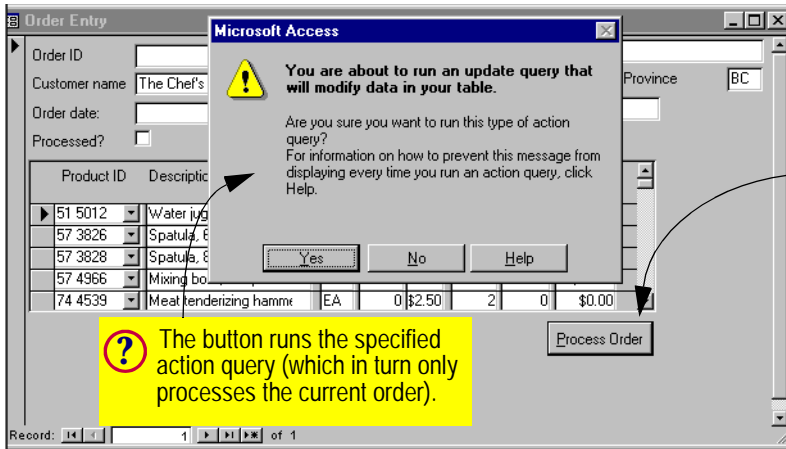
FIGURE 17.11: Add a button to the form using the button wizard (part 3)

6 You can either show an icon or text on the button. In this case, text is more appropriate.

7 Replace the generic button name provided by the wizard with a more meaningful name (e.g., `cmdProcess`).

? WINDOWS trick: If you enter an ampersand before a letter in the caption, that letter becomes the button's "shortcut key". In this example the button is activated when the user presses the **Alt-P** combination.

FIGURE 17.12: Execute the action query by pressing the button.



1 Press the button to execute the action query (or press **Alt-P** to use the shortcut).

? The button runs the specified action query (which in turn only processes the current order).