



16.1 Introduction: Dynamic queries using parameters

 The last few lessons have been primarily concerned with interface issues. In the next several lessons, the focus shifts to transaction processing—making changes to data in response to business events.

A **parameter query** is a query in which the criteria for selecting records are determined when the query is executed rather than when the query is designed. For example, recall the select query shown in [Figure 10.5](#). The criteria in the query were set so that the resulting record set consists exclusively of records that have a **UnitPrice** greater than \$20.

If you wanted a different set of results (say products that cost more than \$50), you would have to open the query in design view, change the criterion, and rerun the query. However, if a parameter is used for the criterion, ACCESS will determine the value of the parameter when the query is executed (for example, by prompting the user or pulling the value off an open form). The result is a flexible query.

 When the concepts from this tutorial are combined with action queries and event handlers, you will have all the tools required to create a basic transaction processing system without writing a single line of programming code.

16.2 Learning objectives

- understand the way in which parameters can be used to create flexible queries
- prompt the user to enter parameter values
- create a query whose results depend on a value on a form

16.3 Exercises

16.3.1 Simple parameter queries

- ➔ Create a query like **qryBasics** (recall [Figure 10.5](#)) and save it under the name **pqryExpensiveProducts**.
- ➔ Replace the criterion in the **UnitPrice** column with a variable criterion: > [**x**] as shown in [Figure 16.1](#).



(lesson16-1.avi)



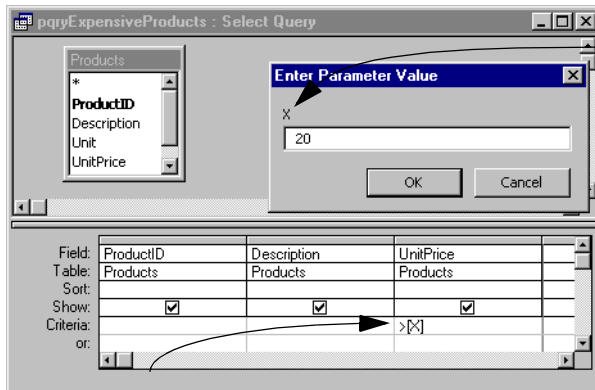
ACCESS expects criteria to be literal strings of text and automatically adds quotation marks to everything in the criteria row. Since the expression `UnitPrice > "x"` is not what we want, we have to tell ACCESS that `x` is the *name* of a parameter (not the letter "X") by enclosing the parameter name in square brackets.

Execute the query as shown in Figure 16.1.

When ACCESS encounters a variable (i.e., something that is not a literal string) during execution, it attempts to bind the variable to some value. To do this, the program does the following:

1. ACCESS checks whether the variable is the name of a field or a calculated field in the query. If it is, the variable is bound to the current value of the field. For example, if

FIGURE 16.1: Convert a select query into a parameter query.



- 1 Replace the literal criterion "20" with a parameter X. Remember to put the parameter name in square brackets so ACCESS does not treat it as text.

- 2 Run the query. When the "Enter parameter value" dialog appears, supply a value for the parameter X.

Product ID	Description	Unit Price
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	Shoemaker, Chef Choice, white	\$78.00
91 354142		\$22.00
91 500304		\$21.50
92 8D02A		\$30.50

- 3 Verify that the results set contains products for which `UnitPrice > X`.



the parameter is named `[ProductID]`, ACCESS replaces the parameter with the current value of the `ProductID` field (e.g., "51 5012"). Since `x` is not the name of a field or a calculated field in this particular query, ACCESS continues looking.

- ACCESS attempts to resolve the parameter as a reference to something within the current environment (e.g., the value on an open form). Since there is nothing called `x` in the current environment, ACCESS continues looking.
- As a last resort, ACCESS asks the user for the value of the parameter via the "Enter Parameter Value" dialog box.



Note that the spelling mistakes discussed in [Section 11.3.2](#) are processed by ACCESS as parameters.

- Press the requery button (**Shift-F9**) to re-execute the query. This time, enter a different value for `x` (e.g., 50).

The power of the parameter query in [Figure 16.1](#) is that the dollar value denoting "expensive" can be defined at query execution time rather than at query design time.

16.3.2 Using parameters to generate prompts

Since the name of the parameter can be anything (as long as it is enclosed in square brackets), you can exploit this feature to create quick and easy dialog boxes.

- Change the name of your `UnitPrice` parameter from `[x]` to `[Show products with a unit price greater than:]`.

- Run the query, as shown in [Figure 16.2](#).



To create a *real* dialog box, you would use an unbound form and provide additional options such as a **Cancel** button.

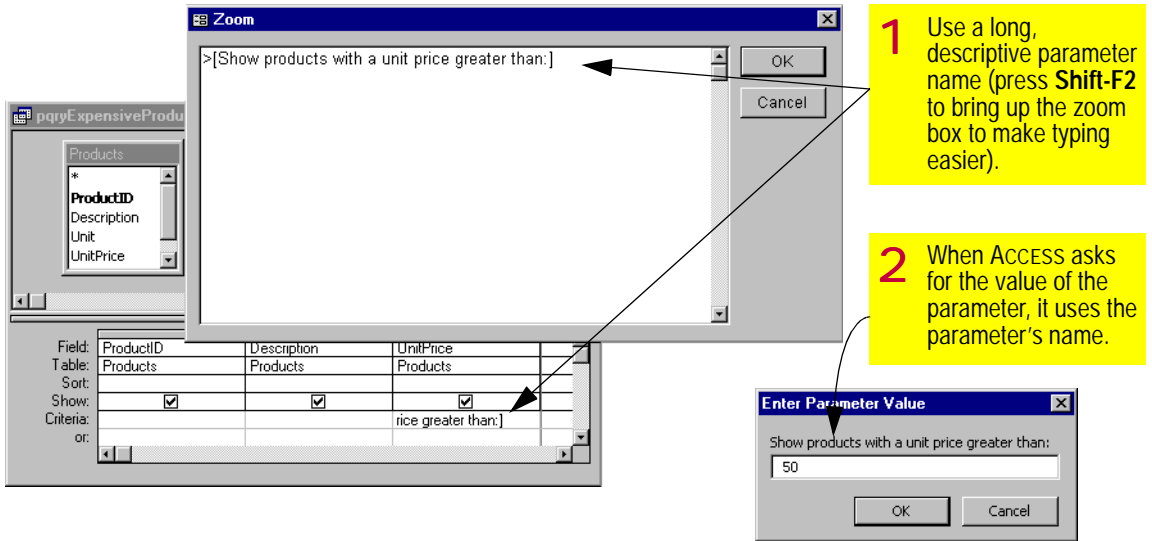
16.3.3 Using values on forms as parameters

A common requirement is to use the value on a form to influence the outcome of a query. For instance, if the user is viewing information about regions, it may be useful to be able to generate a list of customers in the region currently being viewed without the user having to enter any additional information.

Of course, if all you want to do is view the customers, then a synchronized form/subform works well. However, if you want to do more than view the customer data, you will need a parameter query that pulls the value of a



FIGURE 16.2: Select a parameter name that generates a quick-and-easy dialog box.



parameter directly from the open form. The basic idea is shown in [Figure 16.3](#).

- ➔ If you do not already have a **frmRegions** form, create a very simple one.
- ➔ Leave the form open (in form view or design view, it does not matter).

The key to making this parameter query work is to provide a parameter name that correctly references the form object containing the value of interest. In order to avoid having to remember ACCESS' convoluted syntax for naming objects on forms, you can invoke the **expression builder** to select the correct name from the hierarchy of database objects.

(lesson16-2.avi)



FIGURE 16.3: Using the value on an open form as a parameter in a query.

1 The parameter name tells ACCESS where to find the value used in the criterion.

2 The current value in the **RegionCode** field on the form is used as a parameter in the query.

3 The results match the criteria specified on the form.

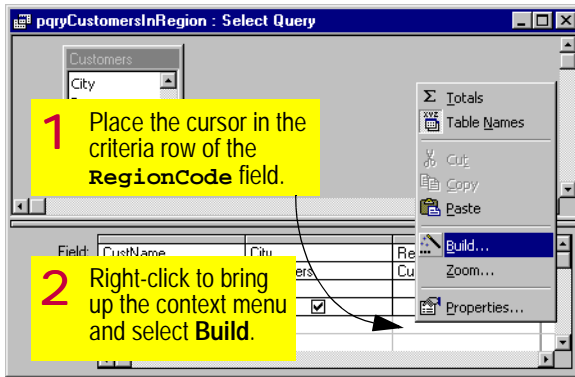
Field:	CustID	CustName	City	RegionCode
Table:	Customers	Customers	Customers	Customers
Sort:				
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:				[Forms][frmRegions][RegionCode]
or:				

Customer ID	Customer name	City	RegionCode
1	Sam's Stock Pot	Vancouver	C
4	Gadgets "R" Us	North Vancouver	C
* (AutoNumber)			

- Create a new query called **pqryCustomersInRegion**. Project a few customer fields including **RegionCode**.
- Move to the criteria row for **RegionCode** and invoke the expression builder, as shown in [Figure 16.4](#).
- Perform the steps shown in [Figure 16.5](#) to create a parameter that references the **RegionCode** field on the **frmRegions** form.
- View the query in datasheet mode. The results should correspond to the region showing on the regions form.
- Move to a new record on the form. Notice that you have to requery the form (**Shift-**



FIGURE 16.4: Invoke the builder to build a parameter.



F9) in order for the new parameter value to be used (see [Figure 16.6](#)).

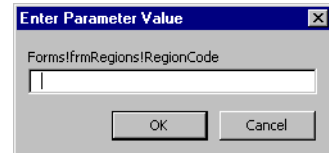


ACCESS does not continuously monitor the parameter for changes. When the query is opened, ACCESS evaluates the parameter and selects according to that value. If you want to force the query to re-evaluate, you have two choices: close and reopen the query or execute the requery action.

If the form is closed, ACCESS is unable to resolve the name of the criterion and has to prompt the user.

➔ Close **frmRegions** and requery the parameter query. You should get the “enter parameter” dialog box shown in [Figure 16.7](#).

FIGURE 16.7: When the form is closed, ACCESS cannot resolve the parameter’s value.



16.4 Application to the project

16.4.1 Selecting the current order

Parameter queries provide a convenient way to control what appears on a form or a report based on what the user is currently viewing on screen. For example, once an order is entered into the order entry system, a user may want to print or fax an invoice to the customer of that order. To implement this feature, an invoice report is created and bound to a parameter query. The parameter query uses the techniques described in [Section 16.3.3](#) to pull the value of the **OrderID** parameter off the currently visible order.



FIGURE 16.5: Use the builder to select the name of the object you want to use as a parameter.

1 Select **Forms** to get a list of all the forms in your database.

2 Since the **frmRegions** form is open, click on **Loaded Forms** and select the form.

3 Move to the middle pane and select **Field List** to get a list of the fields on the form in the pane on the far right.

4 Double-click **RegionCode** to move it to the text area. If you make a mistake, move to the text area, delete the text, and try again.

5 Press **OK** when done. The text will be copied into the criteria row of the query.

? The expression builder saves you from learning the complex naming syntax for objects in the ACCESS environment.

➔ Create a parameter query called **pqryInvoice** that pulls its **OrderID** criteria from the order form. In addition to order information, your invoice should show information about the customer to whom the invoice is being sent.

HINT: Remember when testing the query that the order form must be open in order for ACCESS to find the parameter value.

16.4.2 Using the report writer

Since invoices are meant to be printed, they should be created with the report writer. Creating reports is mostly a mechanical task and contributes little to your understanding of the relational database model. As a consequence, report creation is not covered in these lessons. However, you are not entirely on your own. ACCESS provides a graphical report writer and a report wizard to simplify the process of creating reports.



FIGURE 16.6: Requery the results set to reflect changes on the form.

1 Move to a new record on the form. Notice that the query is *not* automatically updated.

2 Press **Shift-F9** to requery. The new parameter value ("K" in this case) is used to select records.

The structure of an invoice is similar to the structure of the order form you created in [Lesson 14](#): information about the order is shown at the top and information about the order details are shown in the invoice body. The main difference between an order form and an invoice report is that you only need to generate the invoice for the currently visible order. You do not want to generate invoices for all the orders in the **Orders** table because chances are that these invoices have already been created.

As was the case with the order form, you should create a report for the order, a second report for the order details, and combine them into a report/subreport structure linked on **OrderID**.

- Use the report wizard to create a columnar main report based on **pqryInvoice**. Test it to make sure it gives the correct results and save it as **rptInvoice**.
- Use the report wizard to create a tabular subreport that shows order details. You should be able to use your



`qryOrderDetails` query as the record source for the subreport.



Since the invoice is not used for decision making, it can show less information than the order subform. For example, there is no requirement for the invoice to show confidential information such as quantity on hand.



Use the drag-and-drop process described in [Section 14.3.3](#) to create a subreport control on the `rptInvoice` form.



Report/subreport synchronization is very important. Make sure you verify the link fields for the subreport control in the same way that you verified the link fields for your subform controls (review [Section 14.3.4](#) as required).