

15.1 Introduction: What is a combo box?

So far, the only kind of user interface **control** you have used on your forms has been the text box. However, ACCESS provides other controls (such as combo boxes, list boxes, check boxes, radio buttons, and so on) that can be used to improve the attractiveness and functionality of your forms. These interface elements are called **bound controls** because they are bound to fields in the underlying table. When you change data in a bound control, you are changing the data in the underlying table.

In this lesson, we are going to focus on a particularly useful bound control: the combo box. A combo box is list of values from which users can select a single value. Not only does selecting from a list save typing, it can be used as a means of enforcing referential integrity since the user's choices are (typically) limited to the values in the list.



The term “combo box” is a MICROSOFT-ism that is slowly working its way into standard parlance. Synonyms include drop-down list, select list, pick list, and so on.

Figure 15.1 shows a combo box used to assign sales reps to regions. Since users are limited to the choices in the combo box, there is no danger of entering a non-existent employee ID or the employee ID of someone who is not in sales (assuming that the list provided by the combo box is correct).

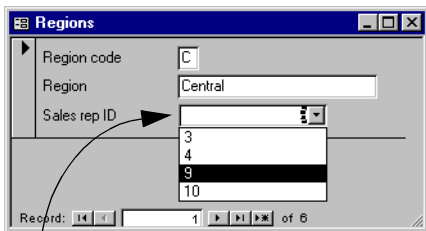
Although advanced controls such as combo boxes and list boxes look and behave very differently than simple text boxes, their function is ultimately the same. For example, the very basic combo box in Figure 15.1 is bound to the **Regions.RepID** field. When a value in the combo box is selected, it is copied into the underlying field exactly as if the user had typed the value (3, 9, 10, etc.) into a text box.



It is important to realize that combo boxes have no intrinsic *search* capability. Combo boxes change values; they do not automatically move to the record with the value you select. For example, selecting **RepID = 9** does not move to a region serviced by that employee (after all, there may be more than one region). Of course, it is possible to use combo boxes for search, but implementing this



FIGURE 15.1: A combo box for filling in the **RepID** field.



Instead of relying on the user to select a valid value for **RepID**, a combo box is used to show the **EMP_ID** values of all the employees in the sales group. When the user selects "9", the value is inserted into the **RepID** field of the **Regions** table.

functionality requires a small amount of programming.

15.2 Learning objectives

- create a bound combo box
- create a combo box that displays values from a different table
- show additional information in a combo box
- prevent certain information from showing in the combo box

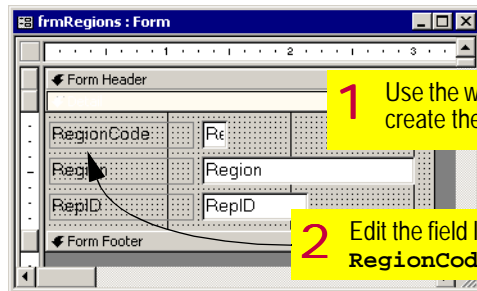
- change the order in which the items appear in a combo box
- understand and change tab order
- know whether to put a combo box on a key field

15.3 Exercises

In the following exercises, you will create a basic "regions" form using the wizard and replace the **RepID** textbox with combo boxes of increasing complexity.

- ➔ Use the form wizard to create a new columnar form called **frmRegions**, as shown in [Figure 15.2](#).

FIGURE 15.2: Create a simple columnar form for the **Regions** table.





Since you did not set a suitable caption for the **RegionCode** field when you created in [Section 5.3.5](#), the wizard uses the field name at the label for the textbox.

- Change the field label for the **RegionCode** field to “Region code” (or whatever you think is appropriate).
- Set the **Caption** property of the form to “Regions” (review [Section 14.3.6.1](#) as required).
- Ensure the toolbox and field list are visible (review [Figure 13.3](#) as required).


15.3.1 Creating a combo box manually


Although ACCESS has a wizard that simplifies the process of creating combo boxes, you will start by building a simple combo box (similar to the one shown in [Figure 15.1](#)) with the wizard turned off. This will give you a better appreciation for what the wizard does and provide you with the skills to make refinements to wizard-created controls.

15.3.1.1 Adding the control to the form

(lesson15-1.avi)

- Delete the existing **RepID** text box by selecting it and pressing the **Delete** key.

The wizard toggle button () in the toolbox allows you to turn wizard support on and off.

- Ensure the button is out (wizards are turned off).
- Click on the combo box tool (). The cursor turns into a small combo box.
- With the combo box tool selected, drag the **RepID** field from the field list to the desired location on the form’s detail section, as shown in [Figure 15.3](#).

The process of selecting a tool from the toolbox, and then using the tool to drag a field from the field list ensures that the control you create is bound to a field in the underlying table or query.



If you forget to drag the field in from the field list, you will create an unbound combo box, as shown in [Figure 15.4](#). If you accidentally create an unbound combo box, the easiest thing to do is to delete it and try again.

FIGURE 15.3: Create a bound combo box without using the wizard.

1 Ensure the wizard button is *not* depressed.

2 Click on the combo box button to activate the combo box tool.

3 Select the **RepID** field from the field list.

4 Drag the **RepID** field on to the detail area. If you have done this correctly, the name of the underlying field should show in the combo box and the label should take the value of the field's caption.

15.3.1.2 Filling in the combo box properties

➡ Switch to form view and test the combo box that you just created.

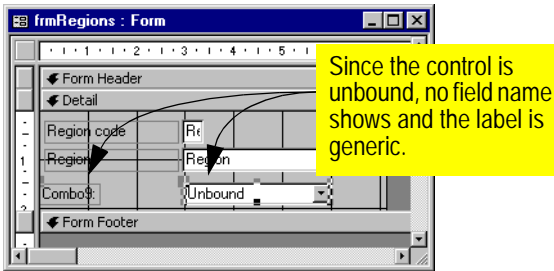
In this section, you will tell ACCESS what you want to appear in the rows of the new combo box.

At this point, the combo box does not show any list items because we have not specified what the list items should be. There are three

methods of specifying what shows up in the combo box list:

1. Enter a list of values into the combo box's **Row Source** property;
2. Tell ACCESS to get the value from an existing table or query;
3. Tell ACCESS to use the names of fields in an existing table (you will not use this approach).

FIGURE 15.4: An unbound combo box (not what you want).



Although the second method is the most powerful and flexible, we will start with the first.

- Switch to form view and test the combo box that you just created.

(lesson15-2.avi)

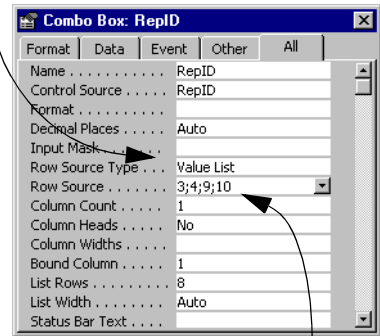
- Switch to form view and test the combo box that you just created.

- Bring up the property sheet for the **RepID** combo box.

- Change the **Row Source Type** property to Value List. This tells ACCESS to expect a list of values in its **Row Source** property.

FIGURE 15.5: Set the **Row Source Type** and **Row Source** properties for the combo box.


- 1 Set the **Row Source Type** property to "Value List".




- 2 Specify the values that you want to show in the combo box rows. Use a semicolon to separate items.

- Enter the following values into the **Row Source** property, as shown in **Figure 15.5**: **3;4;9;10**. These values correspond to the employee IDs of employees in the sales group.

➔ Set the **Limit To List** property to Yes.

 If the **Limit To List** property is set to No, the user can ignore the choices in the combo box and simply type in a value (e.g., "53"). In this particular situation, you want to limit the user to the four choices given.

➔ Switch to form view and experiment with the combo box.

 Notice that the combo box has some useful built-in features. For example, if you choose to type values rather than select them with a mouse, the combo box anticipates your choice based on the letters you type. Thus, to select "10", you need only type "1".

15.3.2 A combo box based on another table or query


An obvious limitation of the value-list method of creating combo boxes is that it is impossible to change or update the items that appear in the list without finding and modifying the **Row Source** property. If you have many forms that use combo boxes based on **EMP_ID**, the result is a maintenance nightmare.

A more elegant and flexible method of populating the rows of a combo box is to have

ACCESS create the list of items in the combo box dynamically using an existing table or query.


15.3.2.1 Preparing the source data

Since you only want sales employees to show up in this combo box, you need to create a query that provides this information before continuing with the combo box.

 If you saved the **qryEmployees** query you created in [Section 11.3.1](#), you can use it as the basis for the **qrySalespeople** query. The procedure below assumes that you are creating the query from scratch.

➔ Switch to the database window and create a new query called **qrySalespeople**.

➔ Project the asterisk and the **EMP_JOB_CL** field. Uncheck the **Show** box for **EMP_JOB_CL**, as shown in [Figure 15.6](#).

 You project the **EMP_JOB_CL** field into the query so that you can filter out non-sales employees. However, since you have also projected the asterisk, you must ensure that the **Show** box under the **EMP_JOB_CL** column is *unchecked*. Otherwise, **EMP_JOB_CL** will appear in the query's results set twice.



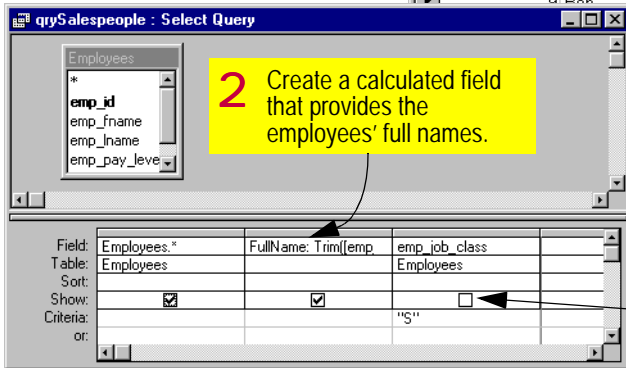
- ➔ As a criteria in the **EMP_JOB_CL** field, enter "S" ("S" is the job class for salespeople).
- ➔ Create a calculated field called **FullName** that concatenates the employees' first and

last names (review [Section 11.3.1](#) as required).

The resulting query and results are shown in [Figure 15.6](#).

FIGURE 15.6: Create a query that provides information about employees in the sales group.

1 Create a query that lists all employees with a job class equal to "S" (sales).



qrySalespeople : Select Query

| emp_id | emp_fname | emp_lname | emp_pay_level | emp_job_class | FullName |
|--------|-----------|------------|---------------|---------------|-------------------|
| 3 | Jocelyn | Scorer | 2 | S | Jocelyn Scorer |
| 4 | Bill | Williams | 2 | S | Bill Williams |
| 9 | Ben | Sidhu | 3 | S | Ben Sidhu |
| | | Villeneuve | 2 | S | Sabine Villeneuve |

4 Verify the results. These are the values of **EmployeeID** that can be used in the **RepID** field.

3 To avoid having **EMP_JOB_CL** in the results twice, ensure the **Show** box is unchecked.

15.3.2.2 Using the combo box wizard

Although the basic process of setting the combo box properties remains the same regardless of whether its properties are set manually or using the wizard, the wizard is far more efficient

when building a combo box based on a table or a query.

(lesson15-3.avi)

- ➔ Delete the existing **RepID** combo box.

- ➔ Ensure the wizard button (🔮) in the toolbox is depressed (wizards are activated).
- ➔ Repeat the steps for creating a bound combo box (i.e., select the combo box tool and drag the **RepID** field from the field list on to the detail section). Since the control

wizard is now activated, you should get the dialog shown in [Figure 15.7](#).

The wizard asks you to specify a number of things about the list of values that appears in the combo box:

1. the table (or query) from which values are taken;

FIGURE 15.7: Create a combo box using the combo box wizard.

1 Specify the source of the rows for the combo box and press **Next**.

2 Use the radio button to indicate that you only want to view queries as the possible source of records for your combo box.

3 Select **qrySalespeople** from the list and press **Next**.

2. the field (or fields) that you would like to show up as columns in the list;
3. the width of the field(s) in the list;
4. the column from the list (if more than one column is visible) that is bound to the underlying field; and,
5. the label that accompanies the combo box.

To complete your combo box, follow the instructions given by the wizard:

- ➔ Indicate that the rows in the combo box should come from a table or query, and select the correct query, as shown in [Figure 15.7](#).
- ➔ Since **RepID** has to be a valid value of **Employees.EMP_ID**, select this field for the combo box as shown in [Figure 15.8](#).

In the second last step of the wizard, you are asked whether to store the value picked from the combo box in a field. Since the drag-and-drop procedure you used to create the combo box binds it to the **RepID** field, you should not have to make any changes in the last two steps of the wizard.


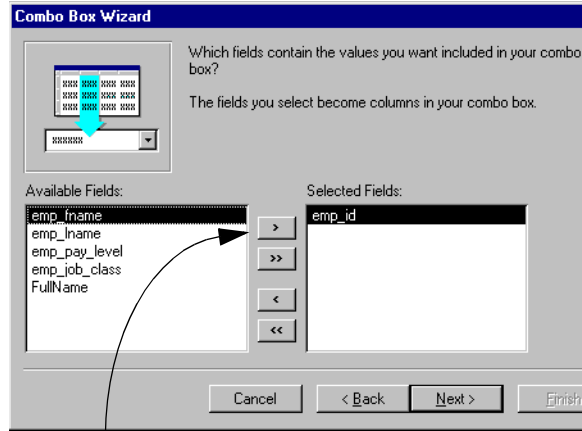
 Normally, in the last step of the wizard, you would enter a descriptive label to be shown on the left of the combo box. However, since you will shortly delete this

FIGURE 15.8: Select the field to show in the combo box.



1 Select **EMP_ID** as the field to show. No other choices are appropriate since **RepID** is a foreign key that references **EMP_ID**.

combo box and replace it with a better one, you should not waste any time formatting or tidying up these early efforts.

- ➔ Switch to form view and test your combo box. It should look similar to that shown in [Figure 15.1](#).

Clearly, updating or changing the values in the combo box is much easier when the combo box is based on a table. Since you have used the linked **Employees** table as the basis for **qrySalespeople**, then your combo box will always show the current members of the sales group (according to the human resources information system). This could be very helpful in a company with significant employee turnover.¹

15.3.2.3 Showing more than one column in the combo box

One problem with the combo boxes created to this point is that they are not of much use to a user who is not familiar with the employee IDs of the sales group. In this section, you will use the **FullName** field of the **qrySalespeople** query to make the combo box easier to use.

(lesson15-4.avi)

- ➔ Delete the existing combo box and start again.
- ➔ Fill in the wizard dialog sheets as in [Section 15.3.2](#), but create a multi-column

¹ The downside of using shared data is that an employee cannot be assigned to a region until he or she is added to the HR system. If the HR system is updated slowly or in batches, the use of shared data becomes a liability.

combo box, as shown in [Figure 15.9](#) and [Figure 15.10](#).

- ➔ Verify that your combo box resembles the one shown in [Figure 15.11](#).

15.3.2.4 Hiding the bound column

Assume that the **EMP_ID** values do not have any business meaning for users of this system. In such a case, you might be tempted to include only the **FullName** field in the combo box. However, this would not work because the target **RepID** field expects a long integer corresponding to a valid value of **EMP_ID**. If you try to stuff the text “Ben Sidhu” into a long integer field, you will get an error.

In this section, you will create a combo box identical to that shown in [Figure 15.11](#) except that the key column (**EMP_ID**) will be hidden from view. Despite its invisibility, however, the **EMP_ID** column will still be bound to the **RepID** field of the underlying table and thus the combo box will work as it should.

(lesson15-5.avi)

- ➔ Delete the existing combo box and start again using the combo box wizard.
- ➔ Include both the **EMP_ID** and **FullName** fields in the combo box.

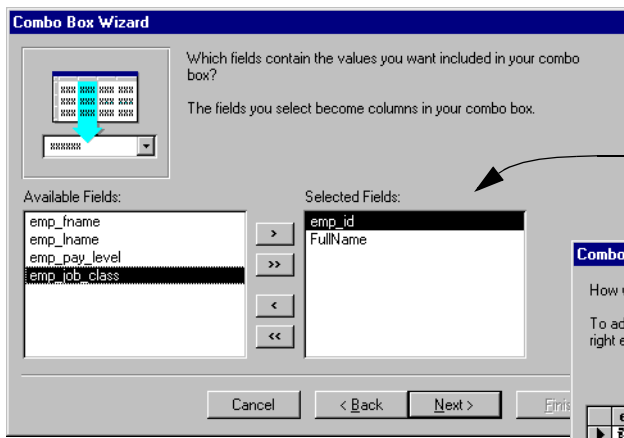
➔ Resize the **EMP_ID** column to a width of zero, as shown in [Figure 15.12](#).

❓ If you base your combo box on a table with a non-concatenated primary key, then ACCESS (version 7.0 and greater) provides a check box to hide the key, as shown in [Figure 15.13](#). However, since the **RepID** combo box in the current example

is based on a query, this feature is not available.

➔ Complete the combo box as in [Section 15.3.2.3](#).

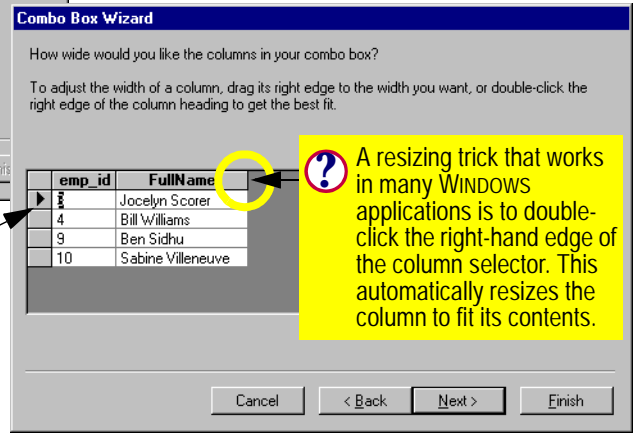
➔ Ensure that the **Input Mask** property for the combo box (which is inherited from the **RepID** field's **Input Mask** property) is blank.



1 Include both **EMP_ID** and **FullName** in the combo box.

FIGURE 15.9: Use the wizard to add more than one field to the combo box (part 1).

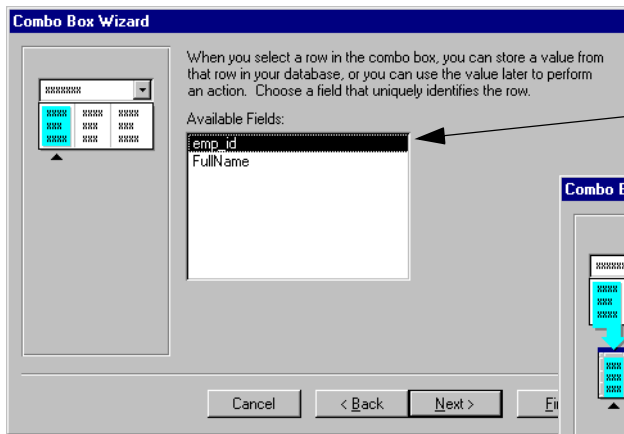
2 Resize the two columns to suit your tastes.



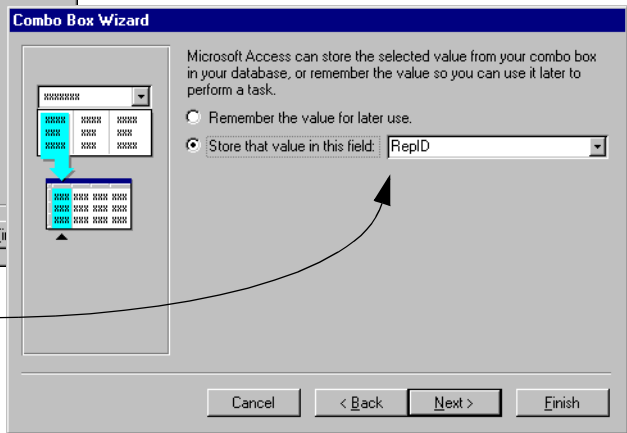
❓ A resizing trick that works in many WINDOWS applications is to double-click the right-hand edge of the column selector. This automatically resizes the column to fit its contents.



FIGURE 15.10: Use the wizard to add more than one field to the combo box (part 2).



3 Although more than one column can be shown in the combo box, only one can be used to supply the value for the underlying field. Since the purpose of this combo box is to select valid values of **EMP_ID**, it should be selected in this step.



4 As before, the value of **EMP_ID** supplied by the combo box should be stored in the **Regions.RepID** field.



If you create an input mask for a field and then use a multi-column combo box to display an entirely different data type in the control, you must remove the input mask for the control.

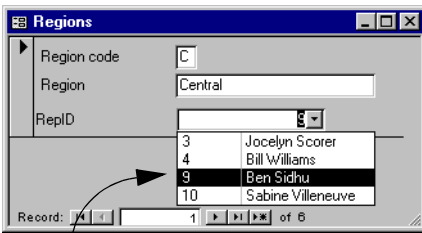


Although the combo box really contains a **RepID**, it looks to the user like the combo box is bound to the employee's name. You should change the label to make this illusion complete.

➡ Change the label for the **RepID** field from "RepID" to "Sales rep".

➡ Verify that the resulting combo box resembles that shown in [Figure 15.14](#).

FIGURE 15.11: A combo box showing multiple columns from the source query.



The **FullName** field is shown help the user choose between different values of **EMP_ID**.

15.3.2.5 Changing the order of rows in the combo box

In some cases, you may want to make minor modifications to the appearance of the combo box without altering the source table or query. For example, in the combo box in [Figure 15.14](#), you may wish to show the members of the sales group in alphabetical order. You can do this by changing the embedded SQL statement in the **Row Source** property of the combo box.

(lesson15-6.avi)

- Bring up the property sheet for the **ReplID** combo box.

- Put the cursor in the **Row Source** property and press the builder button (...). This shows the embedded SQL statement in the QBE editor.

- Modify the query to sort by **FullName**, as shown in [Figure 15.15](#).

- Instead of saving the query in the normal way, simply close the QBE box using the close button (X).

15.3.3 Changing a form's tab order

A form's **tab order** determines the order in which the objects on a form are visited when the **Tab** or **Enter** (or **Return**) keys are pressed. ACCESS sets the tab order according to the sequence in which controls are added to the form. As a result, when you delete a text box and replace it with a combo box or some other control, the new control becomes the last item in the tab order regardless of its position on the form.

To illustrate the problem, you are going to create a **CustID** combo box on the order form that you created in [Lesson 14](#).

(lesson15-7.avi)

- Open **frmOrders** in design mode and delete the **CustID** textbox.

FIGURE 15.12: Resize the columns to hide the key.

1 Click on the right side of the column selector and drag the edge of the **EMP_ID** column to the far left (i.e., make its width zero).

Although the **EMP_ID** field is still the first column in the combo box, it is hidden from view.

➔ Create a combo box that uses values from the **Customers** table.

no business meaning and should be hidden from users whenever possible.

➔ Hide **CustID** in the combo box.

➔ Use “Customer name” for the combo box’s label.

? Anytime you use an AutoNumber to automatically generate unique keys for a table, it probably means that the key has

➔ Switch to form view and use the **Tab** key to move from field to field.

FIGURE 15.13: The “hide key” option is available when using a table for a row source.

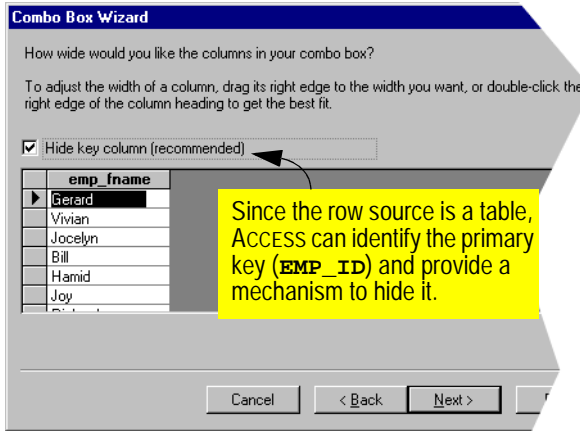
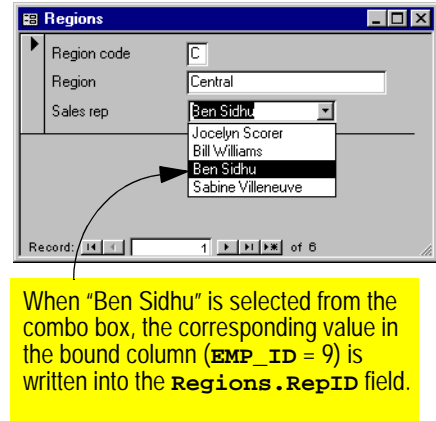


FIGURE 15.14: An multi-column combo box with the bound column hidden.



Notice that the focus seems to skip the combo box when tabbing from field to field. If you put the cursor on the combo box and press the tab key, you will move to the next record because **CustID** is now the last control on the form.

➔ To fix the problem, return to form design mode and select **View** → **Tab Order** from the main menu.

2 In ACCESS version 2.0, the menu structure is slightly different: Select **Edit** → **Tab Order** instead.

➔ Move the **CustID** field from the end of the list to an appropriate location, as shown in [Figure 15.16](#).

➔ Since the **CustID** combo box now shows the customers name, delete the **CustName** textbox from the form.



FIGURE 15.15: An multi-column combo box with the bound column hidden.

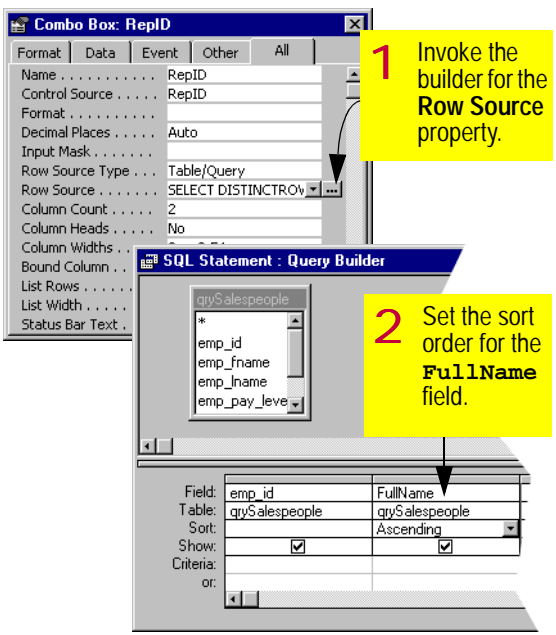
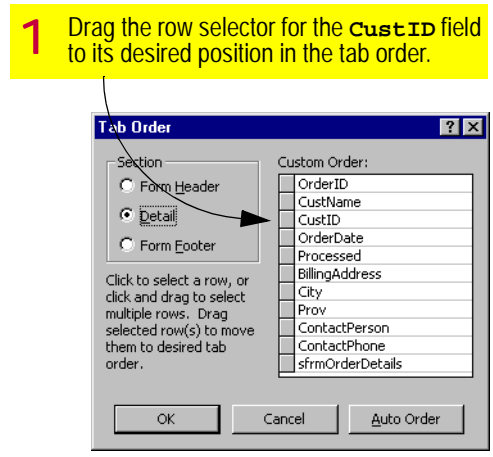


FIGURE 15.16: Set the tab order for the new combo box.



15.4 Discussion

15.4.1 Why you should never bind a combo box to a primary key.

Once new users learn how to create combo boxes, a mistake they often make is to put a combo box on everything. There are certain situations, however, in which the use of a combo box is simply incorrect.

For example, it never makes sense to put a combo box on a non-concatenated primary key. To illustrate, consider the **Employees** form shown in [Figure 15.17](#). On this form, the **EMP_ID** text box has been replaced with a combo box that draws its values from the same **Employees** table to which the form is bound.

This combo box appears to work. However, if you think about it, it makes no sense: The form in [Figure 15.17](#) is a window on the **Employees** table. As such, when the **EMP_ID** combo box is used, one of two things can occur depending on whether a new record is being created or an existing record is being edited:

1. **A new record is being created** — If a new employee is being added to the database, a unique value of **EMP_ID** must be created to distinguish the new employee from existing employees. However, the combo box only shows **EMP_ID** values of *existing* employees. If the **Limit To List** property is set to Yes, then the combo box actually prevents the user from entering a unique **EMP_ID** value.
2. **An existing record is being edited** — It is important to remember that a combo box has no intrinsic search capability. As such, selecting “Bill Williams” in the combo box does not take you to the employee record belonging to Bill Williams. Rather, selecting Bill Williams from the combo box is identical to typing “4” over whatever is

currently in the **EMP_ID** field. For example, in [Figure 15.17](#), Gerard Huff’s employee ID is overwritten by Bill William’s employee ID. Obviously, this generates an error.



A combo box may make sense when the key is concatenated. For example, in your **OrderDetails** subform, a combo box should be used to select values for **ProductID** even through **ProductID** and **OrderID** form a concatenated key.

15.4.2 Controls and widgets

Predefined controls are becoming increasingly popular in software development. Although MICROSOFT includes several predefined controls with ACCESS (such as combo boxes, check boxes, radio buttons, etc.), a large number of more complex or specialized controls are available from MICROSOFT and other vendors such as WWW.COMPONENTSOURCE.COM. In addition, you can write your own custom controls using a language like VISUAL C++ or VISUAL BASIC and incorporate them into many different forms and applications.

An example of a more complex control is the calendar control shown in [Figure 15.18](#). A calendar control can be added to a form to simplify the entry of dates. MICROSOFT calls such components ACTIVE X controls (formerly known as OLE controls). Non-WINDOWS environments also

support components, but tend to favor the more generic terms “interface components” or “widgets”. In JAVA, SUN’s SWING library provides a large collection of interface components that conform to the JAVA BEANS specification.

There are two main advantages of using pre-packaged controls. First, they cut down on the time it takes to develop and test an application. Second, they are standardized so that users encounter the same basic behavior in all applications.

❓ Non-interface (invisible) controls for doing chores like credit card processing, communicating over the internet, and encryption are also widely available.

15.5 Application to the project

There are a number of forms in your assignment that can be greatly enhanced by combo boxes.

- ➔ Ensure that you have created a combo box on your order form to allow users to select customers by name rather than **CustID**.

FIGURE 15.18: A calendar control on a form.

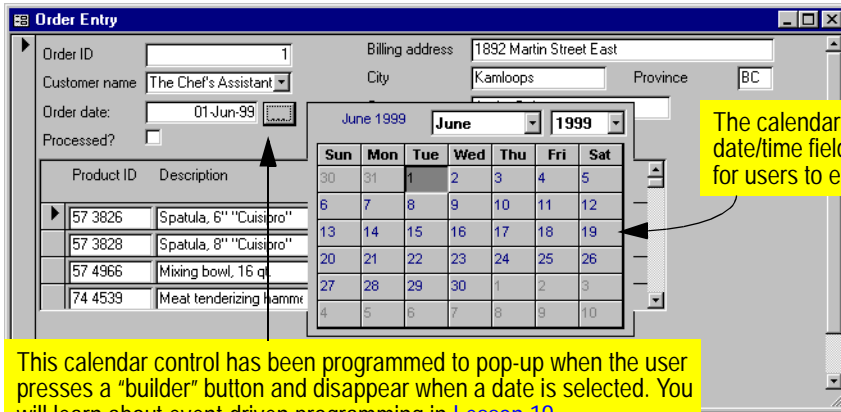


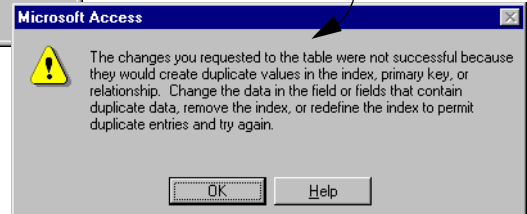


FIGURE 15.17: A combo box on a primary key field never makes sense.

| Employee ID | Last name | First name | Pay level | Job class |
|-------------|----------------|------------|-----------|-----------|
| 1 | Gerard Huff | Gerard | 3 | M |
| 2 | Vivian Peng | | | |
| 3 | Jocelyn Scorer | | | |
| 4 | Bill Williams | | | |
| 5 | Hamid Hassan | | | |
| 6 | Joy Kakuchi | | | |
| 7 | Richard Mason | | | |
| 8 | Russell Plevy | | | |

Since Bill Williams' record already uses **EMP_ID** = 4, and **EMP_ID** is the primary key, it cannot be used again for this record. Hence the error.

A different employee ID can be selected from the combo box. However, the result is that Gerard Huff's employee ID is overwritten by Bill William's employee ID. A bound combo box *does not* magically transport you to Bill William's record.



➔ Disable all the fields on the order form that you do not want users to change during order entry.

➔ Create a combo box in your order details subform to allow the user to select products.

HINT: The **ProductID** values are used by both you and your customers to identify products. In other words, the field has meaning outside of the information system and should not be hidden in combo

boxes bound to the **ProductID** field. In addition, the items in the product list should be sorted by **ProductID** to make it easier to select a product by typing the first few numbers.

HINT: It is very easy for users to confuse two similar **ProductIDs**. To minimize the possibility of entering the wrong product number, you should show the product description in the combo box. In this way, users can confirm that the **ProductID** is correct before they make a selection.