

11.1 Introduction: Virtual fields

A **calculated field** is a “virtual field” in a query. The field is *virtual* in the sense that it is not stored anywhere in the database. Instead, it is calculated dynamically when the query is used.

The value of the calculated field is typically a function of one or more fields in the underlying table. For example, in the **Employees** table, you have fields for the employees’ first and last names (**EMP_FNAME** and **EMP_LNAME** respectively). However, for some printed reports and mailing labels, you may want to combine the first and last names into a single value such as “Vivian Peng”. Although you could add a new field to the table called **emp_fullname** and populate it, this would be an unsatisfactory solution for at least three reasons:

1. An **emp_fullname** field replicates information that already exists in the database (albeit in a different format) and is therefore wasteful of disk space.
2. If Vivian changes her last name, the change has been made in both the **EMP_LNAME** and **emp_fullname** fields.



Whenever a single change in the business environment requires multiple changes in the information system, you can be sure that the information system will eventually be full of inconsistent data.

3. The employee data you are using is from the payroll application. Since you have not been granted write-access to the payroll database, you are not permitted to change the structure of its tables or modify its data.

To get around these problems, you can define a calculated field called **FullName** (or whatever name you like) that is not stored in the **PAY_EMPS** database.

11.2 Learning objectives

- create a calculated field
- understand why Access add square brackets around field names
- understand the use of the ampersand operator (&)



11.3 Exercises

11.3.1 Creating calculated fields

To create the **FullName** calculated field in an ACCESS query, you use the following syntax:

```
FullName: EMP_FNAME & " " & EMP_LNAME
```

Generally, the syntax of a calculated field is of the form: **<field name>: <expression>**, where **<field name>** is a name you provide for the calculated field.



The name of a calculated field can be just about anything, as long as it is unique *within the query* (i.e., it cannot be the same as the name of a field in one of the query's underlying tables). Generally, it is best to follow the same conventions you use when creating regular fields.

The **<expression>** part is any combination of fields, operators, and functions that evaluates to the desired value. In the **FullName** example used above, the expression is simply the concatenation of two text fields with a space inserted (see [Section 11.4.1](#) for more information about using the ampersand operator to concatenate text).

- ➔ Create a new query called **qryEmployees** based on the **Employees** table.

- ➔ Create the **FullName** field, as shown in [Figure 11.1](#).
- ➔ View the query to verify the results, as shown in [Figure 11.2](#).
- ➔ Switch back to the design view of the query. You will notice that ACCESS has added square brackets to the names of your fields.



In ACCESS, square brackets are used to indicate the name of a field (or some other object in the ACCESS environment). If your field name contains one or more blank spaces (e.g., **[Last Name]**), the use of square brackets is *mandatory*: the brackets tell ACCESS that **Last** and **Name** are not two separate expressions. If you use single-word field names (strongly recommended), the use of square brackets is entirely optional.

11.3.2 Errors in queries

It may be that after defining a calculated field, you get the "Enter Parameter Value" dialog box shown in [Figure 11.3](#) when you run the query.

The "Enter Parameter Value" box pops up whenever you spell something in your calculated field definition incorrectly. ACCESS cannot resolve the name of the misspelled field and thus asks the user for the unknown value.



FIGURE 11.1: Create a calculated field based on two other fields.

1 Put the cursor in the field row of an empty column and invoke the zoom window by either pressing **Shift-F2** or right-clicking and selecting **Zoom** from the context menu.

? The “zoom” window provides more room to type than the tiny space in the query definition grid.

? ACCESS ignores case in expressions. Thus, the names **emp_fname** and **EMP_FNAME** are equivalent.

Order Entry System

File Edit View Insert Query Tools Window Help

Zoom

FullName: EMP_FNAME & " & EMP_LNAME

OK

Cancel

Employees

Field:	EMP_FNAME	EMP_LNAME		
Table:	Employees	Employees		
Sort:				
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Criteria:				
or:				

Form View

2 Type in the name and the definition of the calculated field. Remember, a colon is used to separate the field name and its definition.

3 Press **OK** when you have finished typing the expression.



The term “spelling mistake” is used in its broadest sense to cover all sorts of goofs and slips. For example, one way to get an unwelcome “Enter Parameter Value” box is to use a non-existent field name such as **ProductNo** instead of **ProductID** in a calculated expression. You can stare at **ProductNo** for hours and insist that it is

spelled correctly. The lesson: check the names of your fields very carefully when you create calculated fields.

To eliminate the unknown parameter problem, simply return to design mode and correct the spelling mistake.



FIGURE 11.2: View the results of the calculated field.

qryEmployees : Select Query			
	EMP_FNAME	EMP_LNAME	FullName
▶	Gerard	Huff	Gerard Huff
	Vivian	Peng	Vivian Peng
	Jocelyn	Scorer	Jocelyn Scorer
	Bill	Williams	Bill Williams
	Hamid	Hassan	Hamid Hassan
	Joy	Kakuchi	Joy Kakuchi
	Richard	Mason	Richard Mason
	Russell	Plevy	Russell Plevy
	Ben	Sidhu	Ben Sidhu
	Sabine	Villeneuve	Sabine Villeneuve
*			

The **FullName** field is calculated for each record by concatenating the employee's first name, a space, and last name.

FIGURE 11.3: The user is prompted for the value of a misspelled field.

A field name in the calculated field has been misspelled. Since ACCESS does not know the value of "emp_fmame", it asks the user.



As the tutorials progress, you may be surprised to see the "Enter Parameter Value" box in different situations that are seemingly unrelated to queries. For example, the box may pop up when opening a form. Do not panic when this happens—instead, simply look at the name of the expression name that ACCESS cannot evaluate and find the mistake in the underlying query. Then, fix the mistake.

11.3.3 Creating mathematical expressions

In [Section 11.3.1](#), you used a calculated field to transform two text fields into a more useful format. However, as the name implies, calculated fields can also be used to evaluate mathematical expressions.

Assume, for example, that you are considering creating a price list of your products for customers in another country. It makes little sense to have a separate field stored in the table for each currency because there is a simple functional relationship between any two currencies. Not only would separate fields waste disk space, all price changes would have to be repeated for each currency. This would be bad news if you carried thousands of products.

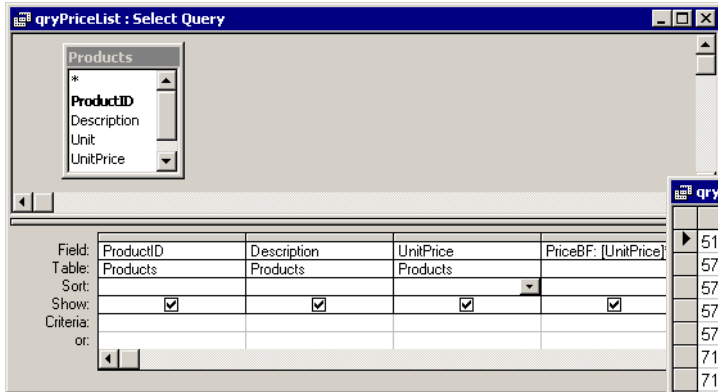


In this section, you are going to create a list that shows the price of your products in Canadian Dollars and Belgian Francs.

- Create a new query based on the **Products** table and save it as **qryPriceList**.
- Project the **ProductID**, **Description**, and **UnitPrice** fields.

- Create a calculated field to convert Canadian Dollars (the currency in which **UnitPrice** is stated) into Belgian Francs (assume that $\text{CDN}\$1 = \text{BF}30.40296$):
PriceBF: UnitPrice * 30.40296
- Switch to datasheet mode and inspect the results, as shown in [Figure 11.4](#).

FIGURE 11.4: Use a calculated field to show product prices in multiple currencies.



1 Create a calculated field to convert **UnitPrice** into Belgian Francs.

Product ID	Description	Unit price	PriceBF
51 5012	Water jug, s.s.	\$23.50	714.46956
57 3826	Spatula, 6" "Cui	\$4.00	121.61184
57 3828	Spatula, 8" "Cui	\$4.25	129.21258
57 4966	Mixing bowl, 16	\$12.50	380.037
57 551	S.S. salad serv	\$3.15	95.769324
71 12101	S.S. soup ladle	\$5.25	159.61554
71 12110	S.S. skimmer	\$5.00	152.0148
71 12111	S.S. sauce ladl	\$5.25	159.61554
71 12114	S.S. grave ladle	\$4.75	144.41406
74 4842	Snail pl		
74 4321	Pastry f		
74 4539	Meat te		
74 6083	Spring f		

2 Change the price of the first record (water jug) and press the **Tab** key. Notice the calculated field changes automatically.

3 Press **Esc** to discard the change before it is saved to the database.



The approach described above works well enough if you are going to print a price list every couple of months. However, in an environment in which exchange rates must be current (e.g., an online store), you would have to edit the **PriceBF** field constantly to update the “hard coded” exchange rate. Naturally, there are ways to do this automatically, but they are a bit beyond us at this early stage.

➡ Change the value of **UnitPrice** for the first item in the datasheet. When you press the **Tab** key, you will see that the value of **PriceBF** changes instantly and automatically.

➡ Press the **Esc** (escape) key to undo the price change without saving it to the database.



Recall from [Figure 5.6](#) that changes to the record buffer are not actually saved to the disk until you move to a different record or explicitly save the record. The escape key (or **Ctrl-Z** or **Edit** → **Undo**) discards changes made to the record buffer before they are written to the disk.

➡ Attempt to change the value of the **PriceBF** field. Notice the error message displayed in the status bar in the bottom-left corner of the Access window.

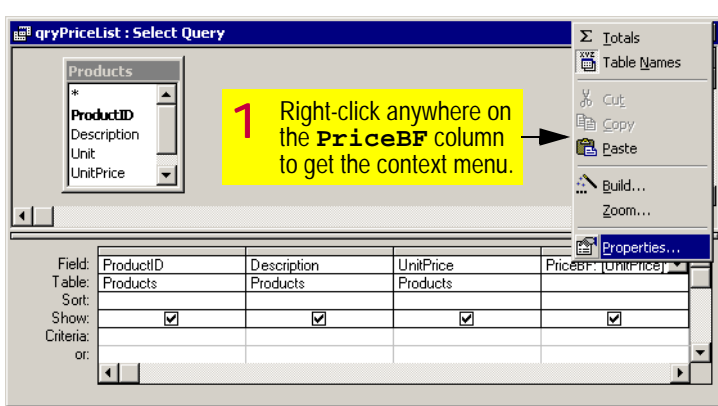
11.3.4 Formatting a calculated field

Users should never see queries or tables—all on-screen interaction with the application should occur through forms and printed output should be generated using the report writer. As such, it really does not matter what the output of a query output looks like because all data will be formatted in a form or report for end-user consumption anyway. This fact notwithstanding, it is often useful to specify the format of calculated fields so that the format is inherited when the forms and reports are created.

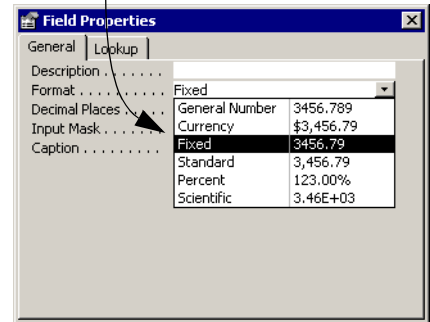
To illustrate, recall [Figure 11.4](#). It is clear that **UnitPrice** has taken the currency format of the underlying **UnitPrice** field in the **Products** table. Since my computer is set up for use in Canada, all currency values are shown with the (Canadian) dollar sign. In contrast, **PriceBF** is interpreted by ACCESS simply as floating point number. Although there is no option (on a computer configured for use in North America) to format a field as Belgian Francs, you can designate the field as a fixed point number (with two decimal places to show the number of *centimes*).

➡ Switch to the design view of **qryPriceList**.

➡ Right-click on the field selector for the **PriceBF** field, as shown on the left-hand side of [Figure 11.5](#).

FIGURE 11.5: Format the *PriceBF* calculated field.

2 Set the **Format** and **Decimal Places** properties to show two decimal places.



- ➔ Select **Properties** from the context menu to get the properties dialog shown on the right-hand side of [Figure 11.5](#).
- ➔ Set the value of the **Format** property to “Fixed”.
- ➔ Set the value of the **Decimal Places** property to “2”.
- ➔ Verify that the **PriceBF** field now has only two digits to the right of the decimal.

Whenever the `qryPriceList.PriceBF` field is used in a form or a report from this point forward, it will be displayed with the correct number of decimal places.



To display the prices “in french”, a number of trickier formatting changes are required. For example, the decimal has to be replaced with a comma. The following calculated field converts the **PriceBF** field into an appropriately-formatted text field (figuring out the expression is left as an exercise):



```
PrixFB: CStr(CInt(PriceBF)) & "\", " &
Right(PriceBF,2) & " FB"
```

11.3.5 Complex calculated fields

If you attempted to create an input mask for the **Products.ProductID** field, you will have already noticed that the field possess some structure. For example, the first two digits appear to indicate a product category.

A better table design would have the category information stored in a separate field in the **Products** table. However, you have very little freedom to make changes to the structure of the data since the current **ProductIDs** are used throughout your company and by your customers. When you encounter such **legacy data**, the best you can do is work around it.

In this section, we are going to assume the following scenario: Certain customers are interested only in small ceramic items and stainless steel utensils. You need to create a query to do the following:

1. Limit the product list to ceramics and stainless steel utensils only.
2. Map the product code to category names that are familiar to your customers (such as "ceramics" and "utensils")
3. Show the **ProductID** and **Description** fields and a calculated **Category** field.

Assume that **ProductIDs** that start with "71" are stainless steel utensils and **ProductIDs** that start with "88" are ceramic items.

➔ Create a new query based on the **Products** table. Project **ProductID** and **Description**. Call it **qryProductListCategories** or something appropriate.

➔ Enter the following criterion for the **ProductID** field:
Like "71*" OR Like "88*"



The **Like** operator allows you to use wildcards in your text-based criteria. In ACCESS, "*" is used to represent any sequence of characters and "?" is used to represent any single character.

To map the first two digits of **ProductID** to categories, you are going to use the "immediate if" function, **iif()**. The **iif()** function uses the following syntax:

```
NL iif(<expression>, <output if true>,
      <output if false>)
```

If you have done any programming, you will recognize **iif()** as a shorthand version of the following code:

```
NL If <expression> = TRUE Then
NL   Return <output if true>
NL Else
NL   Return <output if false>
```



NL End If



The syntax of the `iiif()` statement appears to depend on your locale. For example, some versions of ACCESS used in Europe require that the arguments in the `iiif()` statement be separated by semicolons rather than commas. The best way to resolve an international issue (if you encounter one) is to verify the syntax

of the function using your localized on-line help facility.

- ➔ Create a new calculated field called **Category** as shown in Figure 11.6. Use the following field definition:
 - NL **Category: iiif(Left(ProductID,2)=88, "Ceramics", "Utensils")**
- ➔ Verify the results, as shown in Figure 11.7.

FIGURE 11.6: Create a calculated field using the "immediate if" function

1 Create a calculated field to assign a textual category to products based on the left-most two characters in their **ProductID** fields.

2 Use the **Like** operator to limit the results to products with **ProductIDs** of the form "71xxx..." and "88xxx...".

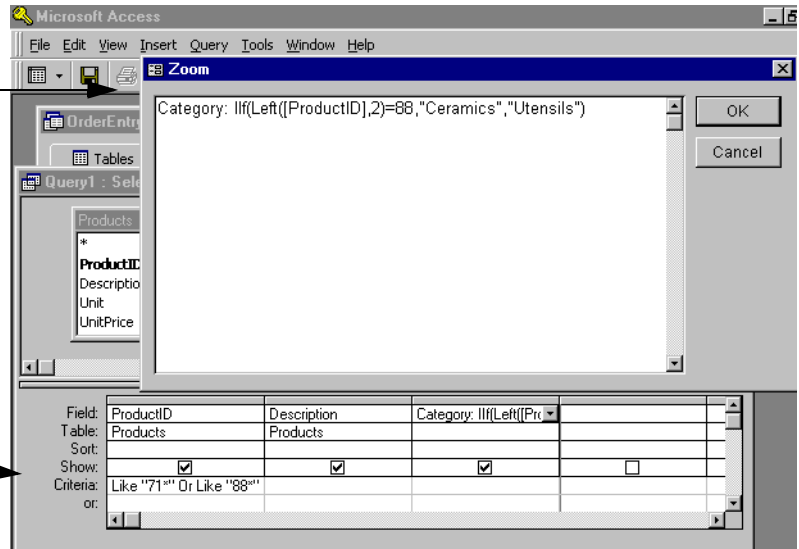




FIGURE 11.7: Results showing the calculated *Category* field.

Product ID	Description	Category
88 3113196015	Salad plate, ocre	Ceramics
88 3122216004	Pasta dish, cobalt	Ceramics
88 3115106014	Sugar, hunter green	Ceramics
88 3115106004	Sugar, cobalt	Ceramics
88 3114106014	Creamer, hunter green	Ceramics
88 3114106057	Creamer, red	Ceramics
88 4017	Mug, "Fat Cat"	Ceramics
88 4077	Mug, white hearts	Ceramics
88 4491	Mug, window cats	Ceramics
88 4742	Mug, polar bear	Ceramics
71 12101	S.S. soup ladle	Utensils
71 12110	S.S. skimmer	Utensils
71 12111	S.S. sauce ladle	Utensils
71 12114	S.S. grave ladle with spout	Utensils



Predefined functions such as `iff()` and `Trim()` functions are discussed in [Section 11.4.2](#).

11.4 Discussion

11.4.1 The concatenation operator

The ampersand operator (&) is like any other operator (e.g., +, -, ×, ÷) except that it is intended for use on strings of characters. In

ACCESS, the ampersand simply adds one string on to the end of another string (hence its other name: the "concatenation" operator). For example, the expression

```
NL "First string" & "Second string"
```

yields the result

```
NL First stringSecond string
```

If a space is include within the quotation marks of the second string (" **Second string**"), the result is:

```
NL First string Second string
```



Use of the ampersand to concatenate text is not widespread outside of MICROSOFT ACCESS and MICROSOFT VISUAL BASIC. In many computer languages, the plus sign (+) is used instead.

11.4.2 Predefined functions

In computer programming, a function is a small program that takes zero or more **arguments** (or **parameters**) as input, does some processing, and returns a value as output. A *predefined* (or *built-in*) function is a function that is provided as part of the programming environment.

For example, `cos(x)` is a predefined function in many computer languages—it takes some number **x** as an argument, does some processing to find its cosine, and returns the answer. Note that since this function is predefined, you do



not have to know anything about the algorithm used to find the cosine, you just have to know the following:

1. what to supply as inputs (e.g., a valid numeric expression representing an angle in radians),
2. what to expect as output (e.g., a real number between -1.0 and 1.0).



The on-line help system provides these two pieces of information (plus a usage example and some additional remarks) for all predefined functions in ACCESS.

these *business* questions before creating the field.

11.5 Application to the assignment

- ➔ Add a calculated field called **ExtendedPrice** to the **qryOrderDetails** query you created in [Section 10.5](#).
Extended price is defined as the number of items of a particular product multiplied by the price of each object.



You have several choices to make when defining this field: Do you use quantity ordered or quantity shipped? Do you multiply by the default price of the product (**UnitPrice**) or the price at which the product is sold to the customer (**ActualPrice**)? Make sure you can answer