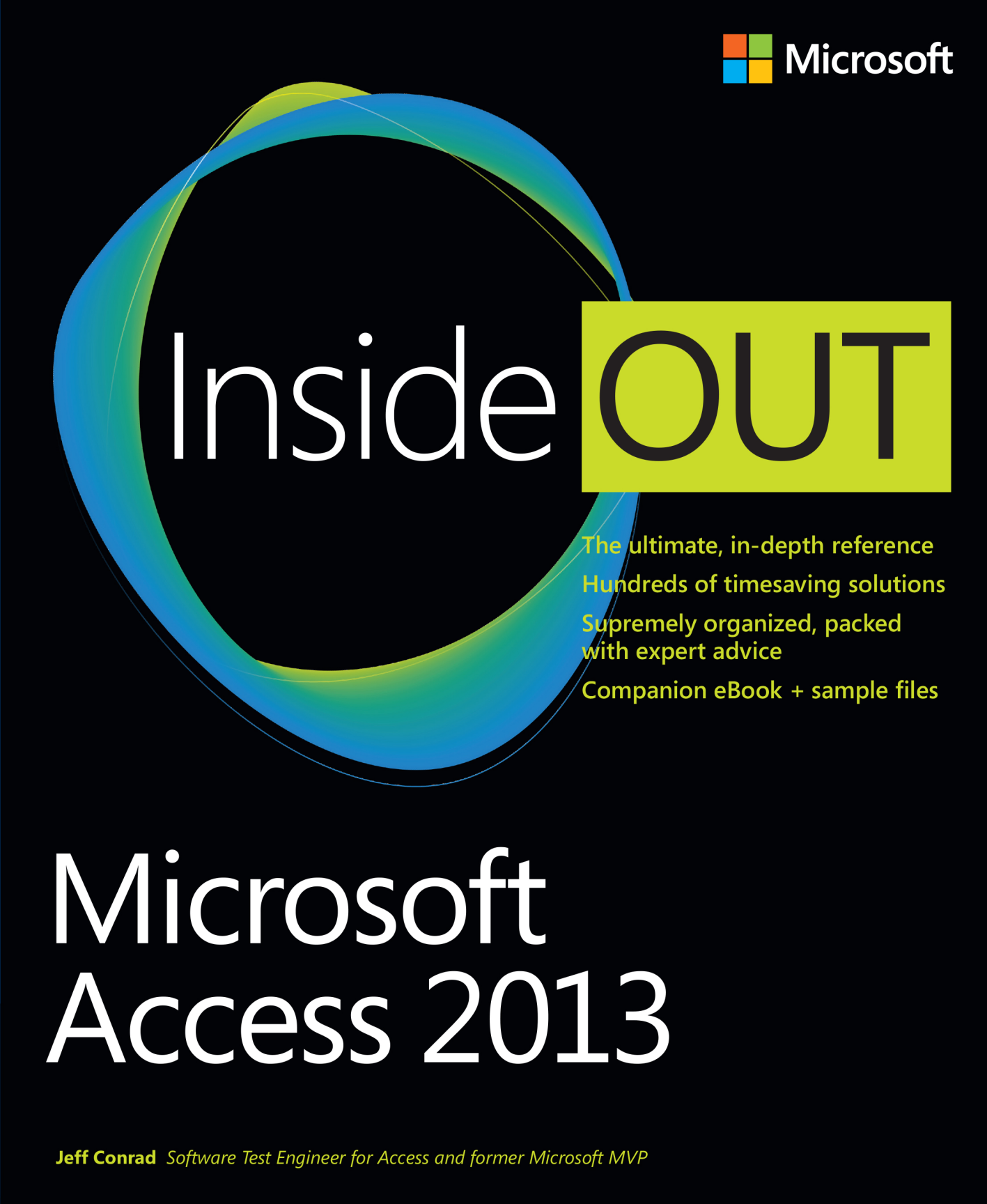

Microsoft
Inside OUT
The ultimate, in-depth reference
Hundreds of timesaving solutions
Supremely organized, packed with expert advice
Companion eBook + sample files
Microsoft Access 2013
Jeff Conrad Software Test Engineer for Access and former Microsoft MVP

Microsoft

Microsoft Access 2013 Inside Out

Jeff Conrad

ISBN: 978-0-7356-7123-2

2 3 4 5 6 7 8 9 10 LSI 8 7 6 5 4 3

Printed and bound in the United States of America.

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at *mspinput@microsoft.com*. Please tell us what you think of this book at *http://www.microsoft.com/learning/booksurvey*.

Acquisitions and Developmental Editor: Kenyon Brown

Production Editor: Christopher Hearse

Technical Reviewer: Andrew Couch

Copyeditor: Richard Carey

Indexer: BIM Publishing Services

Cover Design: Twist Creative • Seattle

Cover Composition: Ellie Volckhausen

Illustrator: Rebecca Demarest

[2013-08-14]

For my wonderful wife, Cheryl, and for Amy, Aaron, and Arica.
Thank you for your love, support, and encouragement.

—Jeff Conrad

Contents at a glance

Table of contents

Introduction

Microsoft Access 2013 is just one part of Microsoft's overall data management product strategy. Like all good relational databases, it allows you to link related information easily—for example, customer and order data that you enter. But Access 2013 also complements other database products because it has several powerful connectivity features. As its name implies, Access can work directly with data from other sources, including many popular PC database programs, with many SQL (Structured Query Language) databases on the desktop, on servers, on minicomputers, or on mainframes, and with data stored on Internet or intranet web servers.

Access provides a very sophisticated application development system for the Microsoft Windows operating system. This helps you build applications quickly, whatever the data source. In fact, you can build simple applications by defining forms and reports based on your data and linking them with a few macros or Microsoft Visual Basic statements; there's no need to write complex code in the classic programming sense. Because Access uses Visual Basic, you can use the same set of skills with other applications in the Microsoft Office system or with Visual Basic.

For small businesses (and for consultants creating applications for small businesses), the Access desktop development features are all that's required to store and manage the data used to run the business. Access coupled with Microsoft SQL Server—on the desktop or on a server—is an ideal way for many medium-size companies to build new applications for Windows quickly and inexpensively. To enhance workgroup productivity, you can use Access 2013 to create an Access Services web app using Microsoft's Office 365 service or on a server with SharePoint 2013, Access Services, and SQL Server 2012. Users of your web app can view, edit, and delete data from your app directly in their web browser. For large corporations with a large investment in mainframe relational database applications and a proliferation of desktop applications that rely on personal computer databases, Access provides the tools to easily link mainframe and personal computer data in a single Windows-based application. Access 2013 includes features to allow you to export or import data in XML format (the lingua franca of data stored on the web).

Who this book is for

If you have never used a database program—including Access—you'll find Access 2013 very approachable. The Backstage view and ribbon technology makes it easy for novice users to get acquainted with Access and easily discover its most useful features. To get a new user jump-started, Microsoft provides web app and desktop database templates

available for download that you can use to begin creating an application that helps solve your personal or business needs.

If you're developing a web app or desktop database application with the tools in Access 2013, *Microsoft Access 2013 Inside Out* gives you a thorough understanding of "programming without pain." It provides a solid foundation for designing web apps, desktop databases, forms, and reports and getting them all to work together. You'll learn that you can quickly create complex applications by linking design elements with macros or Visual Basic. This book will also show you how to take advantage of some of the more advanced features of Access 2013. You'll learn how to build an Access web app that you can use with Microsoft's Office 365 service offering. You'll learn all about the new design surfaces for creating objects in Access web apps and how to use apps in your web browser.

If you're new to developing applications, particularly web apps and database applications, this probably should not be the first book you read about Access. I recommend that you first take a look at *Microsoft Access 2013 Plain & Simple* or *Microsoft Access 2013 Step By Step*.

How this book is organized

Microsoft Access 2013 Inside Out is divided into eight major parts:

Part 1 shows you how to create and work with the all new Access Services web apps:

- Chapter 1, "What is Access," explains the major features that a database should provide, explores those features in Access, and discusses some of the main reasons why you should consider using database software.
- Chapter 2, "Exploring the Access 2013 web app interface," thoroughly explores the web app user interface introduced in the Access 2013 release. The chapter also explains working in the web app environment and installing web app packages.
- Chapter 3, "Designing tables in a web app," teaches you how to design web app tables and how to import and link data into web apps.
- Chapter 4, "Creating data macros in web apps," focuses on how to create data macros and work with table events to attach business logic to your tables.
- Chapter 5, "Working with queries in web apps," shows you how to build queries in web apps and work with data in query Datasheet view.
- Chapter 6, "Working with views and the web browser experience," and Chapter 7, "Advanced view design," exploree the new App Home View, show how to create all the view different view types, work with controls, and understand the properties you

can use with controls in web apps. You'll also learn how to create and work with views in a web browser, and how to manage external connections.

- Chapter 8, "Automating a web app using macros," shows how to work with view and control events to automate your web app.

Part 2 shows you how to create and work with tables in a desktop database:

- Chapter 9, "Exploring the Access 2013 desktop database interface," thoroughly explores the desktop database interface. The chapter also explains content security, working with the Backstage view, ribbon, and the Navigation pane, and setting options that customize how you work with Access 2013.

- Chapter 10, "Designing tables in a desktop database," and Chapter 11, "Modifying your table design," teach you how to design desktop databases and tables and show you the ins and outs of modifying tables, even after you've already begun to load data and build other parts of your application.

The Appendix explains how to install the Office 2013 release, including which options you should choose for Access 2013 to be able to open all the samples in this book.

Part 3 through Part 8, which includes Chapter 12 through Chapter 27, can be found in the Companion Content section on the book's catalog page. The Companion Content also includes seven additional articles with important reference information.

Note

This book is current as of the general availability release date of Microsoft Access 2013 and Office 365 in February 2013. Microsoft is continually updating the Office 365 service offerings, and new features could be implemented after this release date. As a result, some of the features in the product might not exactly match what you see if you are working through the book's examples at a later date.

This book does not discuss the following deprecated features in Access 2013: Access Data Projects (ADP), PivotCharts, PivotTables, Access data collection through email, support for Jet 3.x IISAM, support for dBASE, Access 2003 toolbars and menus, Access Replication Options, Access Source Code Control, Access Three-State Workflow, and the Access Upsizing Wizard. Also, Microsoft removed the ability to create new Access 2010-style web databases with Access 2013 in favor of the new Access 2013 web apps. You can edit existing 2010-style web databases with Access 2013, but you cannot create new ones. Therefore, this book does not discuss how to create and edit 2010-style web databases. If you want to learn about Access 2010-style web databases, see *Microsoft Access 2010 Inside Out*.

Features and conventions used in this book

The following conventions are used in the syntax descriptions for Visual Basic statements in Chapter 24, "Understanding Visual Basic fundamentals," Chapter 25, "Automating your application with Visual Basic," SQL statements in Article 2, "Understanding SQL," and any other chapter where you find syntax defined. These conventions do not apply to code examples listed within the text; all code examples appear exactly as you'll find them in the sample databases.

You must enter all other symbols, such as parentheses and colons, exactly as they appear in the syntax line. Much of the syntax shown in the Visual Basic chapter has been broken into multiple lines. You can format your code all on one line, or you can write a single line of code on multiple lines using the Visual Basic line continuation character (_).

Text conventions

Convention	Meaning
Bold	Bold type indicates keywords and reserved words that you must enter exactly as shown. Microsoft Visual Basic understands keywords entered in uppercase, lowercase, and mixed case type. Access stores SQL keywords in queries in all uppercase, but you can enter the keywords in any case.
Italic	Italicized words represent variables that you supply.
Angle brackets < >	Angle brackets enclose syntactic elements that you must supply. The words inside the angle brackets describe the element but do not show the actual syntax of the element. Do not enter the angle brackets.
Brackets [ ]	Brackets enclose optional items. If more than one item is listed, the items are separated by a pipe character (|). Choose one or none of the elements. Do not enter the brackets or the pipe; they're not part of the element. Note that Visual Basic and SQL in many cases require that you enclose names in brackets. When brackets are required as part of the syntax of variables that you must supply in these examples, the brackets are italicized, as in *[MyTable].[MyField]*.
Braces { }	Braces enclose one or more options. If more than one option is listed, the items are separated by a pipe character (|). Choose one item from the list. Do not enter the braces or the pipe.
Ellipsis ...	Ellipses indicate that you can repeat an item one or more times. When a comma is shown with an ellipsis (,...), enter a comma between items.

Convention	Meaning
Underscore _	You can use a blank space followed by an underscore to continue a line of Visual Basic code to the next line for readability. You cannot place an underscore in the middle of a string literal. You do not need an underscore for continued lines in SQL, but you cannot break a literal across lines.

Design conventions

INSIDE OUT **This statement illustrates an example of an "Inside Out" heading**

These are the book's signature tips. In these tips, you get the straight scoop on what's going on with the software—inside information about why a feature works the way it does. You'll also find handy workarounds to deal with software problems.

Sidebar

Sidebars provide helpful hints, timesaving tricks, or alternative procedures related to the task being discussed.

Troubleshooting

This statement illustrates an example of a "Troubleshooting" problem statement.

Look for these sidebars to find solutions to common problems you might encounter. Troubleshooting sidebars appear next to related information in the chapters. You can also use "Index to Troubleshooting Topics" at the back of the book to look up problems by topic.

Cross-references point you to locations in the book that offer additional information about the topic being discussed.

CAUTION!

Cautions identify potential problems that you should look out for when you're completing a task or that you must address before you can complete a task.

Reader Aid
Notes offer additional information related to the task being discussed.

Your companion ebook

With the ebook edition of this book, you can do the following:

- Search the full text
- Print
- Copy and paste

To download your ebook, please see the instruction page at the back of the book.

About the companion content

I have included companion content to enrich your learning experience. The companion content for this book can be downloaded from the following page:

http://aka.ms/Access2013IO/files

The companion content is organized as follows:

Part 3 focuses on how to build desktop database queries to analyze and update data in your tables.

- Chapter 12, "Creating and working with simple queries," shows you how to build simple desktop database queries and how to work with data in Datasheet view.
- Chapter 13, "Building complex queries," discusses how to design desktop database queries to work with data from multiple tables, summarize information, and build queries that require you to work in SQL view.
- Chapter 14, "Modifying data with action queries," focuses on modifying sets of data with desktop database queries—updating data, inserting new data, deleting sets of data, or creating a new table from a selection of data from existing tables.

Part 4 discusses how to build and work with forms in desktop databases.

- Chapter 15, "Using forms in a desktop database," introduces you to forms—what they look like and how they work.

- Chapter 16, "Building a form," Chapter 17, "Customizing a form," and Chapter 18, "Advanced form design," teach you all about form design in desktop databases, from simple forms you build with a wizard to complex, advanced forms that use embedded forms and navigation and web browser controls.

Part 5 explains how to work with reports in desktop databases.

- Chapter 19, "Using reports," leads you on a guided tour of reports and explains their major features.

- Chapter 20, "Constructing a report," and Chapter 21, "Advanced report design," teach you how to design, build, and implement both simple and complex reports in your application.

Part 6 shows you how to make your desktop database "come alive" using macros.

- Chapter 22, "Creating data macros in desktop databases," explores the macro Logic Designer and shows how to work with events and named data macros within desktop databases.

- Chapter 23, "Using macros in desktop databases," discusses the concept of event processing in Access, provides a comprehensive list of events, and explains the sequence in which critical events occur. It also covers user interface macro design in depth and explains how to use error trapping and embedded macro features.

Part 7 shows you how to use the programming facilities in Microsoft Visual Basic to integrate your database objects and automate your desktop database.

- Chapter 24, "Understanding Visual Basic fundamentals," is a comprehensive reference to the Visual Basic language and object models implemented in Access. It presents two complex coding examples with a line-by-line discussion of the code. The final section shows you how to work with 64-bit Access Visual Basic.

- Chapter 25, "Automating your desktop database with Visual Basic," thoroughly discusses some of the most common tasks that you might want to automate with Visual Basic. Each section describes a problem, shows you specific form or report design techniques you must use to solve the problem, walks you through the code from one or more of the sample databases that implements the solution, and discusses calling named data macros.

Part 8 covers tasks you might want to perform after completing your application.

- Chapter 26, "The finishing touches," teaches you how to automate custom ribbons, create a custom Backstage view, and how to set Startup properties.

- Chapter 27, "Distributing your desktop database," teaches you tasks for setting up your application so that you can distribute it to others. It also shows you how to create your own custom Data Type Parts, Application Parts, and application templates.

The companion content includes an additional seven articles that contain important reference information:

- Article 1 explains a simple technique that you can use to design a good relational database application with little effort. Even if you're already familiar with Access or creating database applications in general, getting the table design right is so important that this article is a "must read" for everyone.
- Article 2 is a complete reference to SQL as implemented in desktop databases. It also contains notes about differences between SQL supported natively by Access and SQL implemented in SQL Server.
- Article 3 explains how to link to or import data from other sources.
- Article 4 discusses how to export data and Access objects to various types of other data formats from your Access application.
- Article 5 lists the functions most commonly used in an Access application, categorized by function type. You'll also find a list of functions that you can use with Access web apps.
- Article 6 lists common color names and codes you can use in Access.
- Article 7 lists the macro actions for both desktop databases and web apps you can use in Access.

Using the sample files

Throughout *Microsoft Access 2013 Inside Out*, you'll see references to sample Access web apps and desktop databases. To access and download the sample applications, visit:

http://aka.ms/Access2013IO/files

For detailed instructions on where to place the sample files on your local computer, see the Appendix. For information on how to install the web app samples (discussed in Part 1 of this book) in your SharePoint site, see the section "Installing app packages," in Chapter 2.

The examples in this book assume you have installed the 32-bit version of Microsoft Office 2013, not just the 32-bit version of Access 2013. You can also download versions of the sample databases that have been modified to work with the 64-bit version of Access 2013. Several examples in this book assume that you have installed all optional features of Access

through the Office 2013 setup program. If you have not installed these additional features, your screen might not match the illustrations in this book or you might not be able to run the sample files. A list of the additional features you will need to run all the samples in this book is included in the Appendix.

A list of the key database files and their descriptions follows. (I have not listed all the smaller support files for the chapters or articles.)

- *Back Office Software System Restaurant Management Web App (BOSS.app)*. This comprehensive web app demonstrates how a restaurant might manage food orders, maintain employee records, and create weekly work schedules. Examples of nearly all features with Access web apps are contained in this large sample web app.

- *Auctions App (Auctions.app).* This sample web app demonstrates using Access to track donated items for auctions and the users bidding on the auction items. This sample contains examples of using data macros to control the data entry by applying logic at the table level.

- *Training Tracker App (TrainingTracker.app)*. This web app tracks different training courses completed by employees. You can also use the app to record employee feedback and the number of hours spent on each training.

- *Conrad Systems Contacts (Contacts.accdb and ContactsData.accdb)*. This desktop database application is both a contacts management and order entry database. This sample database demonstrates how to build a client/server application using only desktop tools. You'll also find a ContactsDataCopy.accdb file that contains additional query, form, and report examples.

- *Housing Reservations (Housing.accdb)*. This desktop database application demonstrates how a company housing department might track and manage reservations in company-owned housing facilities for out-of-town employees and guests. You'll also find HousingDataCopy.accdb and HousingDataCopy2.accdb files that contain many of the query, form, and report examples.

- *Back Office Software System Restaurant Management Application (BOSSDesktopDatabase.accdb)*. This desktop application contains similar functionality to the BOSS.app sample web app, but this sample utilizes desktop database objects and features.

- *Wedding List (WeddingMC.accdb and WeddingList.accdb)*. This application is an example of a simple desktop database that you might build for your personal use. It has a single main table where you can track the names and addresses of invitees, whether they've said that they will attend, the description of any gift they sent, and whether a thank-you note has been sent. Although you might be tempted to store such a simple list in an Excel spreadsheet or a Word document, this application demonstrates

how storing the information in Access makes it easy to search and sort the data and produce reports. The WeddingMC database is automated entirely using macros, and the WeddingList database is the same application automated with Visual Basic.

Here is a list of databases that are discussed in the chapters:

Chapter	Content
Chapter 1	ContactsMap.accdb and Contacts.accdb
Chapter 2	BOSS.app
Chapter 3	RestaurantData.accdb, Contacts.app, BOSS.app, and RestaurantSample.app
Chapters 4 and 5	BOSSDataCopy.app
Chapter 6	RestaurantSampleWithData.app, BOSS.app, and ControlDefinitions.accdb
Chapter 7	RestaurantSampleChapter7.app, BOSS.app, and BOSSReportsMaster.accdb
Chapter 8	RestaurantSampleChapter8.app, BOSS.app sample app, and Auctions.app
Chapter 9	TasksSample.accdb
Chapter 10	WeddingList.accdb, Housing.accdb, and Contacts.accdb
Chapter 11	Housing.accdb, Contacts.accdb, and ContactTracking.accdb
Chapters 12, 13, and 14	ContactsDataCopy.accdb and HousingDataCopy.accdb
Chapter 15	Contacts.accdb and ContactsNavigation.accdb
Chapter 16	ContactsDataCopy.accdb
Chapter 17	HousingDataCopy.accdb and ContactsNavigation.accdb
Chapter 18	HousingDataCopy.accdb, ContactsDataCopy.accdb, and ContactsNavigation.accdb
Chapter 19	ContactsDataCopy.accdb and Housing.accdb
Chapter 20	ContactsDataCopy.accdb
Chapter 21	HousingDataCopy2.accdb
Chapter 22	BOSSDesktopDatabase.accdb
Chapter 23	WeddingMC.accdb and BOSSDesktopDatabase.accdb
Chapter 24	Contacts.accdb and Housing.accdb

Chapter	Content
Chapter 25	Housing.accdb, Contacts.accdb, and WeddingList.accdb
Chapter 26	Contacts.accdb, Housing.accdb, HousingSP.accdb, and BOSSDesktopDatabase.accdb
Chapter 27	Contacts.accdb, Housing.accdb, and ContactsNavigation.accdb

Please note that the person names, company names, email addresses, and web addresses in all the databases are fictitious. Although I pre-loaded all databases with sample data, the Housing Reservations and Conrad Systems Contacts databases also include a special form (zfrmLoadData) that has code to load random data into the sample tables based on parameters that you supply.

> **Note**
>
> **All the screen images in this book were taken on a Windows 8 system with the Office theme set to White and using the Internet Explorer web browser. Your results might look different if you are using a different operating system, a different theme, or a different web browser. Also, the results you see from the samples might not exactly match what you see in this book if you have changed the sample data in the files.**

System requirements

The following are the system requirements you need to install Office 2013, Access 2013, and the sample files on a Microsoft Windows–compatible computer or device.

- A gigahertz (Ghz) or faster x86-bit or x64-bit processor with SSE2 instruction set.
- Microsoft Windows 7 (32-bit or 64-bit), Microsoft Windows 8 (32-bit or 64-bit), Windows Server 2008 R2, or Windows Server 2012 operating systems.
- At least 1 gigabyte (GB) of random access memory (RAM) for 32-bit operating system environments or 2 gigabytes (GB) of RAM for 64-bit operating systems.
- A hard drive with at least 3.0 gigabytes (GB) available.
- A DirectX10 graphics card and 1024 x 576 resolution for graphics hardware acceleration.
- Microsoft Internet Explorer 8, 9, 10, or a later version; Mozilla FireFox 10.x or a later version; Apple Safari 5; or Google Chrome 17.x or a later version.

- Microsoft .NET version 3.5, 4.0, or 4.5.
- A touch-enabled device for using any multi-touch functionality in Windows 8. (However, all features and functionality are always available by using a keyboard, mouse, or other standard or accessible input device.)
- Silverlight installed together with Office 2013 is recommended to improve the online experience.

Acknowledgments

Nearly every member of the Microsoft Access development team provided invaluable technical support as I worked through the finer details in Microsoft Access 2013. The program managers, developers, and test engineers on the team helped with suggestions, tips and tricks, and reviewing my material. You folks make an author's job so much easier. But any errors or omissions in this book are ultimately mine.

A book this large and complex requires a top-notch team to get what I put into Microsoft Word documents onto the printed pages you are now holding. I had some of the best in the business at Microsoft Press, Inc. Media to get the job done. Many thanks go to Kenyon Brown for serving as Acquisitions and Development Editor. Special thanks to Chris Hearse and Richard Carey for handling production and copy editing and to Andrew Couch for technical reviewing. Andrew Couch was especially gifted at not only pointing out any technical mistakes I made, but he was also helpful in offering suggestions for improvement in layout, material, and presentation. Also, thanks to John Viescas for his continued mentoring and friendship. I couldn't have done it without all of you!

And last, but certainly not least, I thank my wife and soul mate, Cheryl. She not only patiently stood by me as I cranked through over 1,900 pages of manuscript, but also helped behind the scenes reviewing and editing what I did. I could not have completed this book without her support.

Support and feedback

The following sections provide information on errata, book support, feedback, and contact information.

Errata

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site:

http://aka.ms/Access2013IO/errata

If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at:

mspinput@microsoft.com.

Please note that product support for Microsoft software is not offered through the addresses above.

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://www.microsoft.com/learning/booksurvey

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

Stay in touch

Let's keep the conversation going! We're on Twitter at: *http://twitter.com/MicrosoftPress*.

CHAPTER 4

Creating data macros in web apps

In Microsoft Access 2013, you can define a data macro to respond to different types of table events that would otherwise require the use of writing macros attached to view and control events. The unique power of data macros in Access 2013 is their ability to automate responses to several types of table events without forcing you to learn a programming language. The event might be a change in the data, the creation of a new record, or even the deletion of an existing record. Within a data macro, you can include multiple actions and define condition checking so that different actions are performed depending on the values in your table fields or criteria you specify.

Note

The examples in this chapter are based on the backup copy of the Back Office Software System sample web app (BOSSDataCopy.app), which can be downloaded from the book's catalog page at *http://aka.ms/Access2013IO/details*. To use the sample, you'll need to upload the app into your corporate catalog or Office 365 team site and install the app. Review the instructions at the end of Chapter 2, "Exploring the Access 2013 web app interface," if you need help with those tasks.

In this chapter, you will:

- Learn about the various types of actions that you can define in data macros and the table events that you can use.
- Tour the logic designer facility and learn how to build both a simple data macro and a data macro with multiple defined actions.
- Learn how to create local variables in data macros to store values temporarily or calculate a result.
- See how to define parameters and use them inside data macro actions.

- Learn how to create return variables in data macros to return data to the calling macro.
- See how to add conditional statements to a data macro to control the actions that Access performs.
- Learn how to create named data macros and execute them from other data macros or table events.
- Understand some of the actions automated with data macros in the Back Office Software System sample web app.

Uses of data macros

Access 2013 provides various types of data macro actions that you can attach to table events as well as inside named data macro objects to automate your web app. With data macros, you can do the following:

- Verify that an invoice is balanced with the invoice detail line items before saving the record.
- Mark an employee as inactive after you create a termination record.
- Prevent any data from being edited, added, or deleted from a table.
- Create new schedule records based on the previous week's schedule or a labor plan template.
- Delete all schedule records within a specific time frame.

As you'll learn in Chapter 8, "Automating a web app using macros," Access 2013 supports *user interface macros* to control application flow in your views and to respond to user actions. You can also utilize user interface macros to enforce complex business logic that might not be covered by table relationships, unique properties, validation rules, and required properties. However, the potential problem with using user interface macros to enforce complex business logic is that you don't always have complete control over how users interact with the data in your tables. For example, users can add, update, and delete data through table and query datasheets. (You'll learn about queries in Chapter 5, "Working with queries in web apps.") Users can also link to the tables in one Access app from an Access desktop database and add, update, and delete data from that database. In both of these examples, users can bypass your complex business logic rules normally stored in user interface macros.

Access 2013 web apps include data macros to provide a place for Access developers to centralize all their business logic and rules. Data macros get translated to triggers and stored procedures in Microsoft SQL Server, and they allow you to attach business logic directly to table events. Because data macros are translated into SQL Server triggers and stored procedures, they are performed within a transactional context—each operation is separate. Data macros attached to table events respond to data modifications, so no matter how users edit data in the web app, SQL Server enforces those rules. This means that you can write business logic in one place, and all the data entry views that update those tables inherit that logic from the data layer. After you create a data macro for a table event, Access runs the data macro no matter how you change the data.

Data macros in Access 2013 can be used in both web apps and desktops databases. However, the events, actions, and expressions that you can use in data macros are not identical between web apps and desktop databases. The Access database engine enforces data macros when you work with a desktop database. When you are using a web app, SQL Server enforces data macros on the server through the use of triggers and stored procedures. (In Chapter 22, "Creating data macros in desktop databases," which can be downloaded from the book's catalog page, you'll learn how to create data macros in desktop databases.)

Touring the Logic Designer

Install the Back Office Software System backup copy sample web app (BOSSDataCopy.app) on your team SharePoint site, and then download the app into Access so that you can follow along with all of the examples in this chapter.

To create data macros, you first need to open a table in Design view. To display all the tables in your BOSSDataCopy web app, click the Navigation Pane button in the Show group on the Home ribbon tab. Double-click the table called tblCompanyInformation to open it in Design view, and then click the Design contextual ribbon tab to see the data macro events, as shown in Figure 4-1.

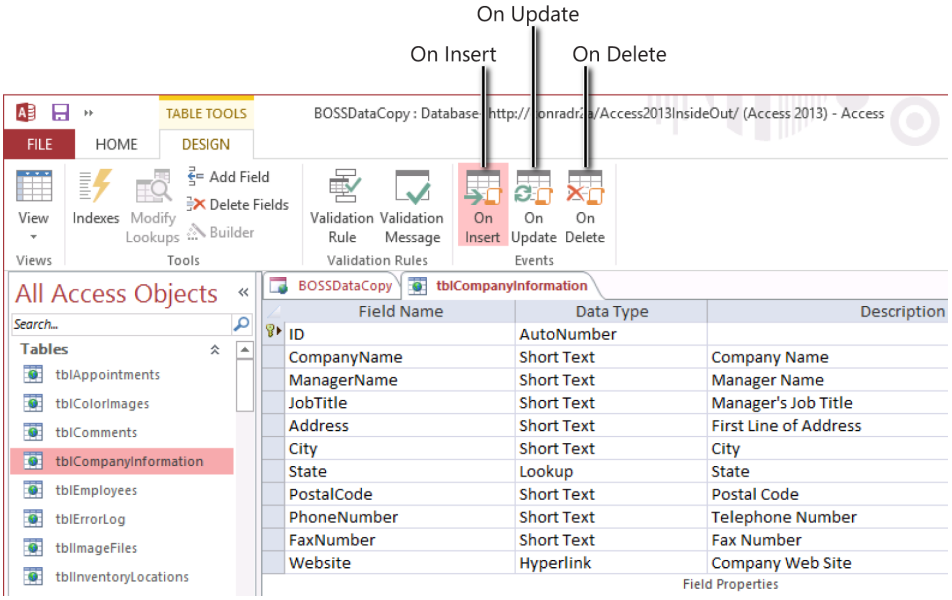


Figure 4-1 Data macro events are listed on the Design contextual ribbon tab under Table Tools in web apps.

You can attach data macros to the On Insert, On Update, and On Delete events of tables. In Figure 4-1, in the Back Office Software System sample web app, you can see that Access highlighted the On Insert and On Delete buttons on the Design contextual ribbon tab. When you create and save a data macro for a table event, Access highlights that event button in the ribbon as a visual cue for you to show that a data macro already exists for that event. To create a new data macro for a table event or edit an existing one, you click the corresponding event button in the ribbon.

Let's explore the existing data macro that I defined for the On Insert event in the tblCompanyInformation table to show you the Logic Designer for creating macros. Click the On Insert button on the Design contextual ribbon tab, and Access opens the Logic Designer, as shown in Figure 4-2.

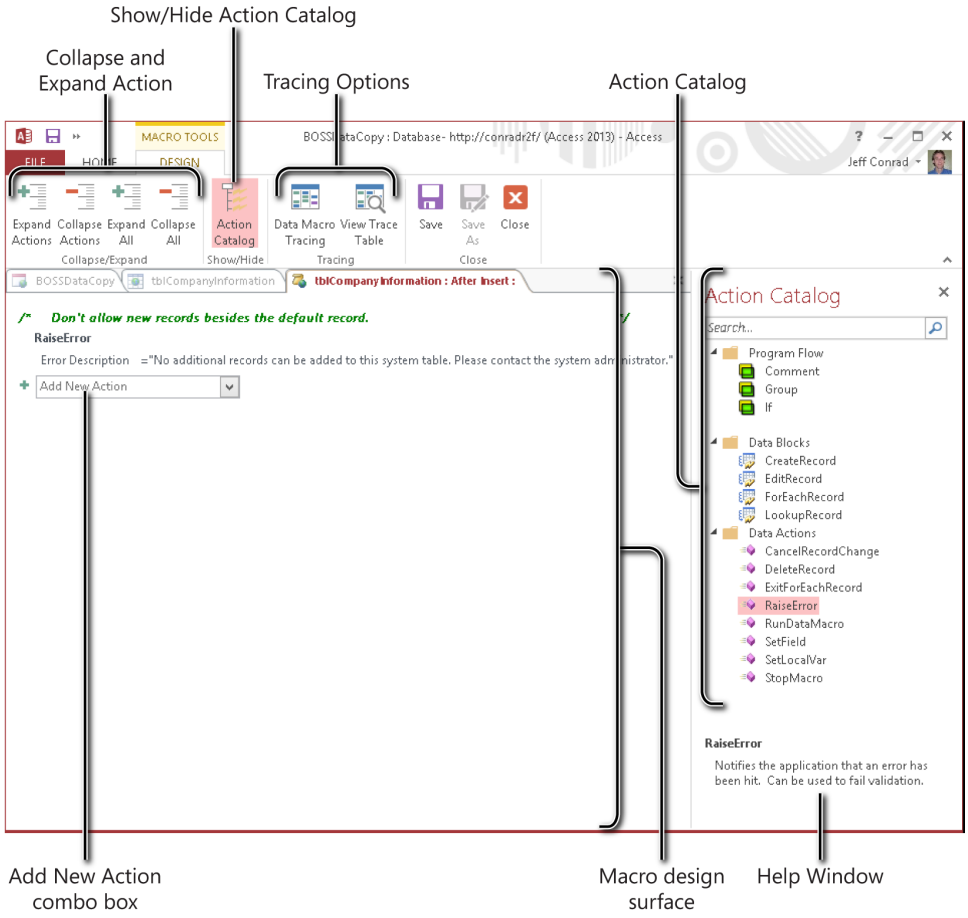


Figure 4-2 This is the Logic Designer, where you can create data and user interface macros.

Whenever you need to create or edit data macros or user interface macros in Access 2013, this is the design surface that you use. You'll notice that Access automatically collapsed the Navigation pane to show you more of the macro design surface. Access also opens the Logic Designer window modally, which means that you cannot open any other database objects until you close the designer window.

As you can see in Figure 4-2, the Logic Designer layout looks more like a Visual Basic code window in desktop databases. The Expand Actions, Collapse Actions, Expand All, and Collapse All buttons in the Collapse/Expand group selectively expand or collapse the actions listed in the macro design surface. In the Show/Hide group on the Design tab, you can choose to hide the Action Catalog shown on the right side of the Logic Designer window by clicking the Action Catalog toggle button. In the Tracing group, Access displays options

to turn on data macro tracing and to display the tracing table to analyze any issues you might have executing your data macro logic. In the Close group, you can click Save to save any changes to your data macro. Click Close to close the Logic Designer window. If you attempt to close the Logic Designer window with unsaved changes, Access asks whether you want to save your changes before closing the window.

On the right side of the Logic Designer window is the Action Catalog. The Action Catalog shows a contextual list of the program flow constructs, data blocks, and data actions that are applicable to the data macro event you are currently viewing. (When you create user interface macros, the Action Catalog similarly displays actions that you can use for user interface macros.) We'll discuss the Action Catalog in more detail in the next section.

In the middle of the Logic Designer window is the main macro design surface where you define your data macro. You add program flow constructs, macro actions, and arguments to the design surface to instruct Access what actions to take for the data macro. If you have more actions than can fit on the screen, Access provides a scroll bar on the right side of the macro design surface so that you can scroll down to see the rest of your actions. You'll notice in Figure 4-2 that Access displays any arguments directly beneath the action. Access displays a combo box called Add New Action at the bottom of the macro design surface. This combo box displays a list of all the actions you can use for the type of data macro you are creating and the specific context of where you are in the data macro logic.

In the lower-right corner of the Logic Designer window is the Help window. Access displays a brief help message in this window, depending on where the focus is located in the Action Catalog.

Click the Close button in the Close group on the Design contextual tab to return to the Design view of the tblCompanyInformation table, and then close the table.

Working with table events

As I mentioned in the previous section, you can attach data macros to the On Insert, On Update, and On Delete table events. In the following sections, you'll learn about each of these events, create new data macros attached to events, and examine other data macros attached to these events in the Back Office Software System sample web app.

In On Insert and On Update events, you can look at the incoming values in the current record and compare them with a record in other tables using the LookupRecord data block. You can use the SetField data action to alter data before Access commits the changes but only on the incoming row of data, not on a record returned from the LookupRecord data block. In all table events, you can prevent a record from being saved or deleted and display custom error messages to the user using the RaiseError data action.

Using On Insert events

The On Insert event fires whenever you add new records to a table. Let's create a new data macro attached to the On Insert event of the tblWeekDays table to illustrate the process of creating, saving, and testing a new data macro. Open the tblWeekDays table in Design view, click the Design contextual tab under Table Tools, and then click the On Insert button in the Events group to open the Logic Designer, as shown in Figure 4-3.

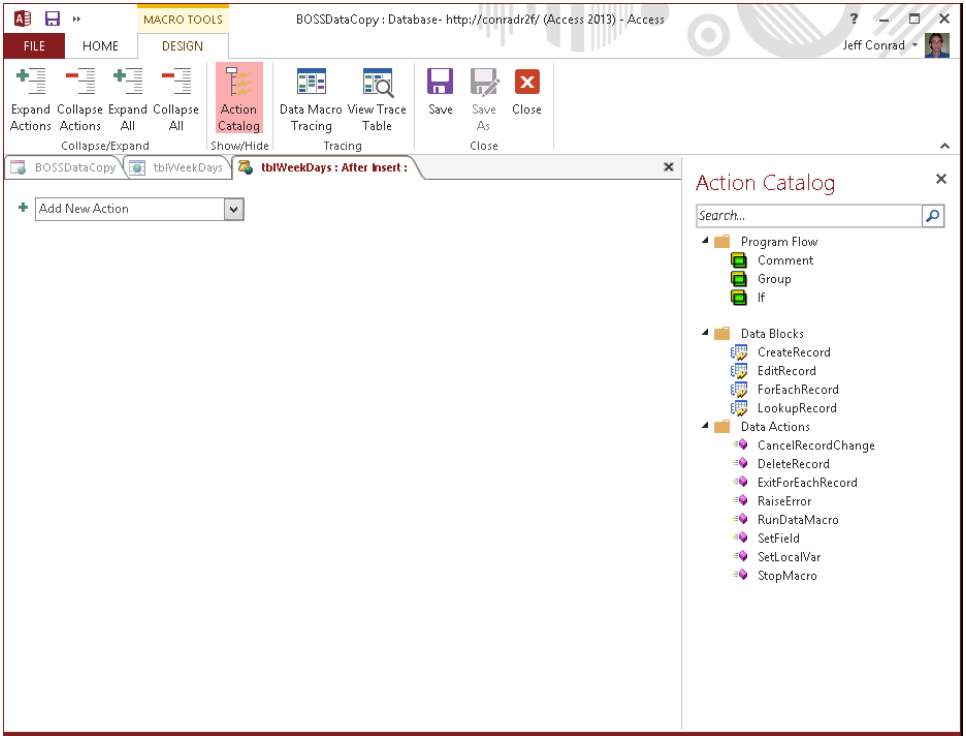


Figure 4-3 Click the On Insert button on the ribbon to begin creating your data macro.

Note

You might have noticed in Figure 4-3 when you started your new On Insert data macro that the caption on the top of the object window displays After Insert. The Logic Designer for data macros is shared between web apps and desktop databases. Although you're seeing a different caption, you are, in fact, creating an On Insert table event data macro.

Troubleshooting

Why can't I add data macros to linked SharePoint lists?

In Chapter 3, "Designing tables in a web app," you learned how to link SharePoint lists into your web app. In web apps, tables linked to SharePoint lists are read-only and cannot be opened in Design view. Therefore, you cannot attach data macros to any table events for linked SharePoint lists. You also cannot reference linked SharePoint lists in any LookupRecord, CreateRecord, or ForEachRecord data blocks attached to other web app table events or in any named data macros.

In the Action Catalog on the right side of the Logic Designer, you can see three options under Program Flow, four options under Data Blocks, and eight options under Data Actions. In web apps, program flow options (Comment, Group, and If), data blocks, and data actions are available in all data macro table events. (In Chapter 22, you'll learn that the options under Data Blocks and Data Actions change based on whether you are using a before event or an after event in desktop databases.) Table 4-1 summarizes the data blocks and data actions that you can use in the table events in web apps.

TABLE 4-1 Data blocks and data actions available in table events

Element	Name	Description
Data blocks	CreateRecord	Creates a new record in a table.
	EditRecord	Allows Access to edit a record. This data block must be used in conjunction with a ForEachRecord or LookupRecord data block.
	ForEachRecord	Iterate over a recordset from a table or query.
	LookupRecord	Instructs Access to look up a record in the same table, a different table, or a query.
Data actions	CancelRecordChange	Cancels any record changes currently in progress. You can use this action to break out of CreateRecord or EditRecord changes.
	DeleteRecord	Deletes the current record from the table. Access determines the current record based on the scope of where the action is called. For example, if you are inside a LookupRecord data block, Access deletes the record found in the Where condition argument.
	ExitForEachRecord	Exits the innermost ForEachRecord loop. You can use this action when you want to break out of a long-running loop if a condition is met.

Element	Name	Description
	RaiseError	Displays a custom message to the user interface level and cancels the event changes. You can use this action to manually throw an error and cancel an insert, update, or delete.
	RunDataMacro	Runs a saved named data macro. You can optionally pass parameters to the named data macro and return values.
	SetField	Changes the value of a field. For example, you can use the SetField action to change the value of another field in the same record before committing the changes.
	SetLocalVar	Creates a temporary local variable and lets you set it to a value that you can reference throughout the data macro execution. The value of the variable stays in memory as long as the data macro runs or until you change the value of the local variable by assigning it a new value. When the data macro completes, Access clears the local variable.
	StopMacro	Stops the current data macro.

The tblWeekDays table contains seven records, each record listing the name of a day of the week. This table helps build a linking table between the tblVendors table and the tblVendorOrderDays table. Each vendor in the app can have more than one day that they accept orders, and each weekday can be used by more than one vendor. Similarly, the tblWeekDays table also serves as a linking table between tblVendors and tblVendorDeliveryDays. For the purposes of this app, I consider tblWeekDays to be a system table: a table used by other parts of the app, but one in which I don't ever need to add, change, or delete data. (I can't foresee the names of the weekdays changing any time soon.) To prevent new records from being added to this table, we'll create a data macro in the On Insert event and include a RaiseError data action to stop the insert.

Including comments

To start creating your data macro in the On Insert event of the tblWeekDays table, let's first add a comment to the macro design surface. Comments are useful for documenting the purpose of your data macro and the various data actions within it. Access ignores any comments as it executes the actions within your data macro. Click the Comment element under the Program Flow node in the Action Catalog, hold the mouse key down, drag the Comment element onto the macro design surface, and then release the mouse button, as shown in Figure 4-4.

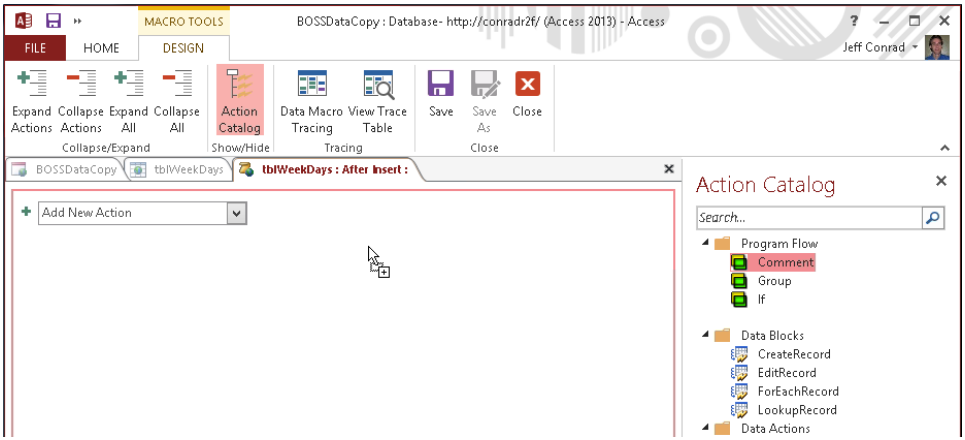


Figure 4-4 Drag the Comment program flow element from the Action Catalog onto the macro design surface.

Access creates a new Comment block on the macro design surface, as shown in Figure 4-5. If your cursor is not in the Comment block and you do not have any comments typed into the Comment block, Access displays the text Click Here To Type A Comment. You'll notice in Figure 4-5 that Access moved the Add New Action box below the Comment block. You'll also notice that Access places a delete button to the far right of the Comment block. (The delete button is a symbol shaped like an X.) If you want to remove the Comment block, click the delete button and Access removes the Comment block from the macro design surface. If you delete the Comment block in error, click the Undo button on the Quick Access Toolbar to restore the Comment block.

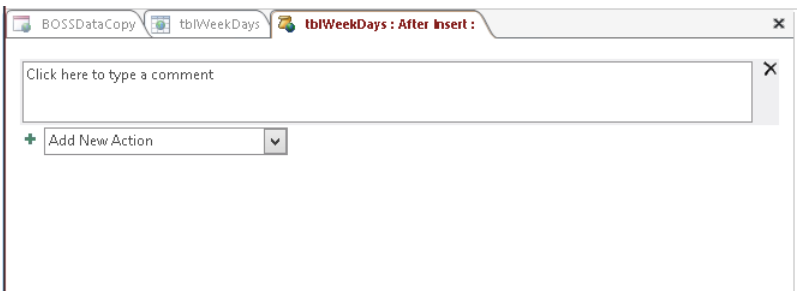


Figure 4-5 Access creates a new Comment block when you drag a Comment program flow onto the macro design surface.

Click inside the Comment block, and type the following text:

We don't want to allow additional records into this system table. If a new record is being added, raise an error and inform the user.

Click outside the Comment block onto the macro design surface. Access collapses the size of the Comment block to just fit the text you typed and displays the text in green, as shown in Figure 4-6. The /* and */ symbols mark the beginning and end of a block of comments. Access designates anything written between those symbols as a comment, which is there only to provide information about the purpose of the data macro or particular action to follow.

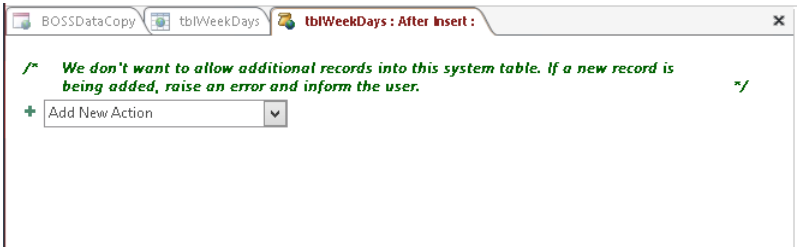


Figure 4-6 Access displays any comments inside comment block characters.

INSIDE OUT Take the time to include comments

You might be asking yourself whether it's really worth your time including comments in your data macros. While it's true that it takes additional time to include comments as you're creating your data macros, the investment of your time now pays off in the future. If you need to modify your app at a later date, you'll find it much easier to understand the purpose of your data macros if you include comments. This is especially true if someone else needs to make changes to your app. Trust me; it's worth your time to include comments when you design data macros.

Grouping macros

When you're creating data macros, you can use a program flow construct called Group. You use a Group construct to group a set of actions together logically to make your data macro actions easier to read. When you group macro actions inside a Group construct, you can also expand or collapse the entire group easily to see more of the macro design surface. It's not required to use the Group construct when you're creating data macros; however, grouping macro actions can be especially helpful if you have many disparate actions inside the same event or named data macro.

To add a Group construct to your data macro, click the Group element in the Action Catalog, hold down the mouse key, and drag the Group element to just beneath the comment block that you inserted previously. As you get close to the comment block, you'll notice

that Access displays a horizontal bar across the macro design surface, as shown in Figure 4-7. This horizontal bar is your insertion point for the new program flow, data block, or data action. If you want to drop your new Group above the comment block, position your mouse pointer above the comment block and Access displays the horizontal bar above the comments to indicate where it will drop your new Group. We want to have this Group positioned below the comment block, so place your mouse pointer below the comment block and then release the mouse.

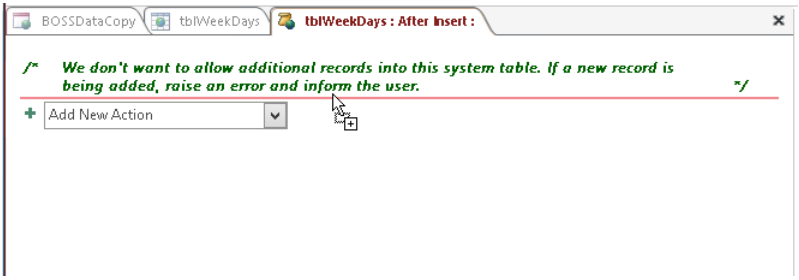


Figure 4-7 Access displays a horizontal bar on the macro design surface when you drag items from the Action Catalog.

Access displays a new Group block on the macro design surface, as shown in Figure 4-8. You need to provide a name for your new Group block, so type **PreventNewRecords** in the text box provided. You are limited to 256 characters, including any spaces, for the name of any Group block.

In Figure 4-8, you'll notice that Access denotes the end of the Group block by placing the words End Group at the bottom of the Group block. When you click on the Group block, Access highlights the entire block as a visual cue to indicate where the starting and ending points of the block are. You'll also notice that Access placed another Add New Action combo box inside the Group block when you dropped the Group construct onto the design surface. You can use this combo box to add new actions inside the Group block. (We'll do that in just a moment.) Next to the delete button on the right side of the Group block is a green up arrow button. Click this button if you want to move the entire Group block above the Comment block that you created earlier. For now, leave the Group block where it is.

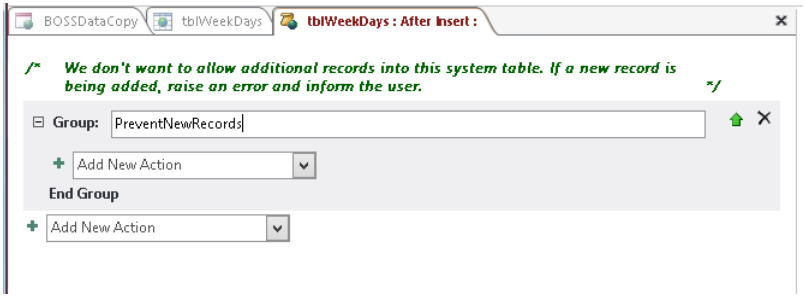


Figure 4-8 You can use a Group block to group a set of actions together logically.

Raising errors in data macros to cancel events

In Chapter 8, you'll learn that user interface macros can interact heavily with the user's experience working with views. With user interface macros, you can display message boxes, open pop-up views, and dynamically change properties on a view. Data macros, on the contrary, are limited to the data layer and cannot interact with the user interface level. For example, in a data macro you cannot display a custom message box to the user and perform different steps based on how the user responds to your message. The only tool you can use in data macros to display information to the user is the RaiseError data action.

You can use the RaiseError data action whenever you need to force an error to occur and display a non-actionable message to the user manually. When you use the RaiseError action in a data macro, Access cancels the pending insert, update, or delete if it reaches this action during the macro execution.

In the On Insert event that you've been building for the tblWeekDays table, we don't want to allow new records to be created in this table. To add a RaiseError action inside the Group block that you previously created, you could drag the RaiseError data action from the Action Catalog onto the macro design surface and place the insertion point inside the Group block. You've already done this type of procedure twice before when creating the Comment and Group blocks, so let's show you an alternative way of adding new elements to the macro design surface. Click the Add New Action combo box inside the Group block, and Access displays a context-sensitive drop-down list of all the program flow constructs, data blocks, and data actions that you can use, based on where your insertion point is located. Click the RaiseError option from the drop-down list, as shown in Figure 4-9, to add a RaiseError data action to the macro design surface.

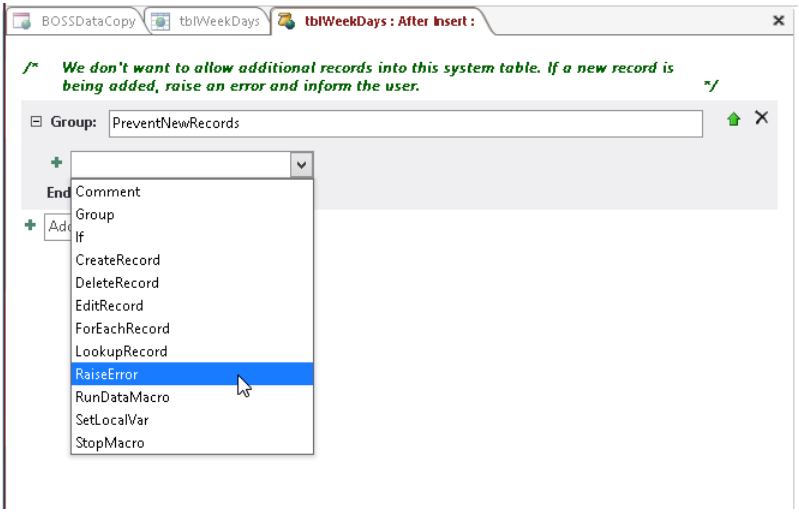


Figure 4-9 Select the RaiseError option from the Add New Action combo box inside the Group block.

Instead of using your mouse to select program flow constructs, data blocks, and data actions from the Add New Action combo box, you can also tab into the control and start typing the first letter or two of the element you want. Access highlights the first construct, data block, or data action that matches the letters you type. You can press Enter at any time, and Access adds the selected element to the macro design surface. (The macro design surface is flexible to allow you to use the mouse for selecting actions or just the keyboard if you prefer.) After you select RaiseError from the Add New Action combo box, Access displays the RaiseError data action inside the Group block, as shown in Figure 4-10.

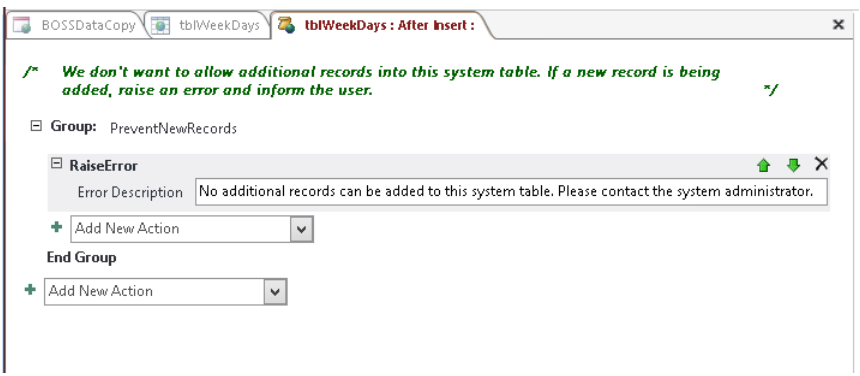


Figure 4-10 Use the RaiseError data action when you need to cancel an insert, update, or delete.

The RaiseError data action has one required argument—Error Description. The Error Description argument is the message displayed to the user if the RaiseError action is hit during execution of the data macro. You can type any custom message you want, up to 256 characters in length. You can also use an expression for the Error Description by typing the equal sign (=) as the first character. In the example earlier in this chapter, the text string started with an equal sign (=) and was enclosed within quotation marks. You're not required to use this technique with simple text strings. However, if you use an expression, you must start the expression with the equal sign (=) and enclose any text string within quotation marks. If you type an equal sign (=) at the beginning of the Error Description argument, Access displays the Expression Builder button on the far right of the text box if you need assistance creating your expression. (You'll see an example of using an expression in a RaiseError action later in this chapter.) For this example, you'd like to display a simple message to the user informing them that they cannot enter new records into this table. Type the following message, previously shown in Figure 4-10, into the Error Description argument:

No additional records can be added to this system table. Please contact the system administrator.

If you do not provide an error description in your RaiseError data action, Access displays an error message when you try to save your data macro logic, as shown in Figure 4-11. You must provide a message in the Error Description to save your data macro.

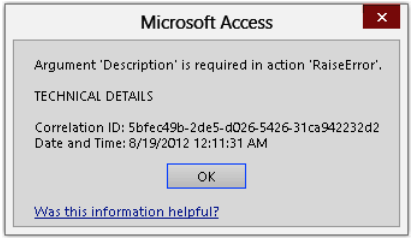


Figure 4-11 Access displays an error message if you leave the Error Description argument empty.

INSIDE OUT **Pause over elements to see Help information**

A very useful feature of the Logic Designer window is the ability to view Help information quickly no matter where you are. When you place your mouse over any element on the macro design surface, Access displays a tooltip with specific Help information covering the program flow, data block, data action, or argument that you are currently on. Similarly, Access displays tooltips with Help information when you pause over the elements displayed in the Action Catalog. This feature is especially useful as you are learning your way around the Logic Designer.

Testing your data macro

You've now completed all the steps necessary to prevent any new records from being added to the tblWeekDays table. To test the data macro that you've created so far, you first need to save your changes to the On Insert event. Click the Save button in the Close group on the Design contextual tab under Macro Tools, or click the Save button on the Quick Access Toolbar. Now click the Close button in the Close group to close the Logic Designer window, and return to the Design view of the tblWeekDays table. To test this On Insert event, you need to create a new record in this table. Switch to Datasheet view by right-clicking the tblWeekDays table in the Navigation pane and selecting Open from the shortcut menu or right-clicking the object tab at the top of the application window and selecting Datasheet view from the shortcut menu. Click in the WeekDayText field on the new record line of the table datasheet, enter any text other than one of the existing weekday names, and then tab or click outside of the new record line. Access displays the custom error that you created in the RaiseError data action, as shown in Figure 4-12.

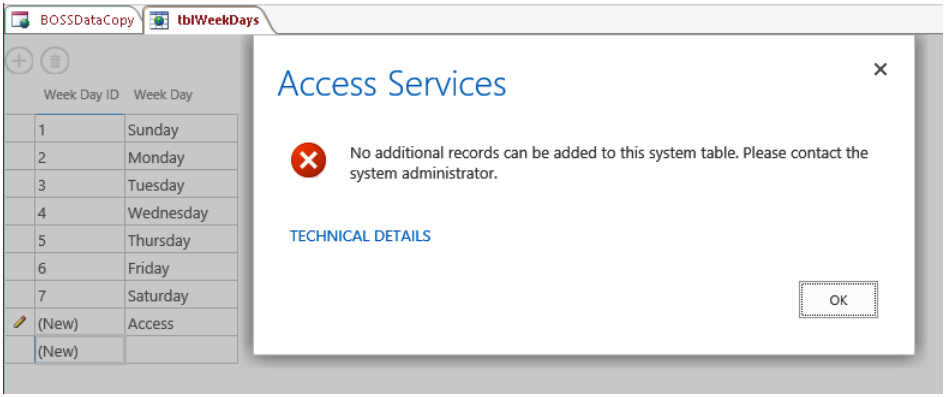


Figure 4-12 Access prevents you from adding new records with the data macro that you created for the On Insert event.

The On Insert event fires because you are inserting a new record into this table. In this event, Access checks to see what data macro logic, if any, to execute when you are creating new records. In this case, the RaiseError data action fires, Access displays the custom message that you created, and then Access cancels the insert. When you click OK in the message box, Access displays a pencil icon in the row selector on the left to indicate the new record is not saved yet. You now need to right-click the row selector and select Delete to remove that uncommitted record from the datasheet. You can also choose to close the table datasheet with this uncommitted record or click the Refresh command on the ribbon. Access prompts you that you have pending changes, as shown in Figure 4-13.

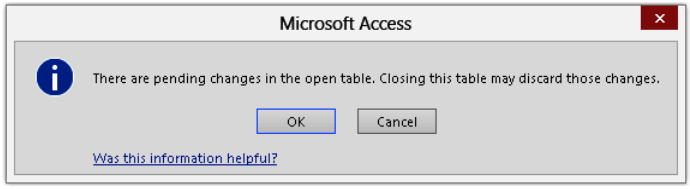


Figure 4-13 Access prompts you when you have unsaved records.

Access attempts to save any record changes when you move off a record or close the table datasheet, but in this case, Access cannot save your record changes because of the RaiseError action in the On Insert event. If you click OK on the pending changes dialog, Access cancels any pending records updates or inserts and then closes the table datasheet. If you click Cancel, Access stops the table datasheet from closing and returns focus to the datasheet; however, your record inserts or updates are still not saved. There is no way that you can add records to this table unless you remove the data macro that you defined in the On Insert event of the table. Access enforces this restriction no matter what the entry point is for creating a new record. As you can see, data macros are a very powerful feature in Access 2013 web apps.

Using If blocks to create conditional expressions

You can define more than one action within a data macro, and you can specify which actions get executed or not by adding conditional expressions into your data macro logic. For example, you might want to update a field in the same record, but only if a specific field was changed. Or, you might want to prevent an update to a record if a value in another field is a higher or lower value than you expect. In the preceding section, you designed a simple data macro in the On Insert event of the tblWeekDays table to prevent new records from being added to the table using a single action. In this section, we'll create data macro logic in the On Insert event of the tblEmployees table to update an image field each time you add a new employee record, using a conditional expression and multiple actions.

Open the tblEmployees table in Design view, click the Design contextual tab under Table Tools, and then click the On Insert button in the Events group to open the Logic Designer.

The employees table includes an image field—EmployeePicture—that I use to store the picture of each employee in our restaurant management app. If the data entry person entering a new employee record into the app does not currently have a picture for the new employee, we want to save a default generic picture for the new employee record to indicate that we don't have a current picture.

Let's begin creating our data macro logic by first adding a new Comment block to the macro design surface. Click inside the Add New Action combo box on the macro design surface, type **Comment**, and then press Enter to create a new Comment block. Type the following text into the Comment block to identify easily the logic that we are going to add to this data macro:

If no picture was assigned for this new employee, use the generic default image instead from tblImageFiles. First check to see if the EmployeePicture field is Null.

Your changes to the On Insert event should now look like Figure 4-14.

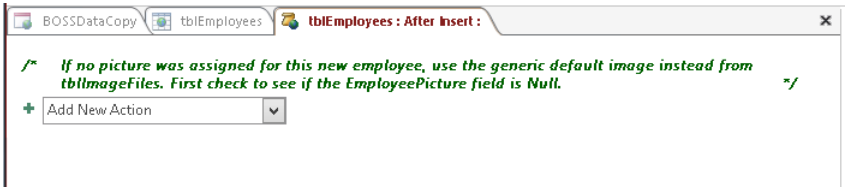


Figure 4-14 Add a Comment block to the macro design surface to document the purpose of this set of actions.

INSIDE OUT

Shortcut keys to adding Comment blocks

To add a new Comment block onto the macro design surface quickly, you can simply type two forward slashes (//) when you are in any Add New Action combo box and press Enter. Alternatively, you can type a single apostrophe (') when you are in an Add New Action combo box and press Enter. In both cases, Access creates a new Comment block on the macro design surface.

In the Add New Action combo box, type **If** and press Enter to create a new If block. Access creates a new If block under the Comment block, as shown in Figure 4-15. The text box next to If is where you type your conditional expression. Each condition is an expression that Access can evaluate to True or False. A condition can also consist of multiple comparison expressions and Boolean operators. If the condition is True, Access executes the action or actions immediately following the Then keyword. If the condition is False, Access evaluates the next Else If condition or executes the statements following the Else keyword,

whichever occurs next. If no Else or Else If condition exists after the Then keyword, Access executes the next action following the End If keyword.

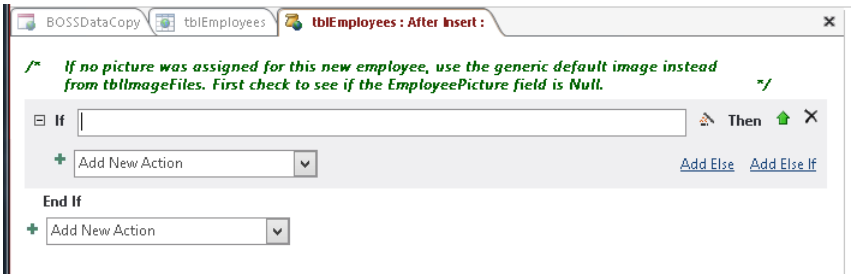


Figure 4-15 Use an If block when you want to execute actions only if a certain condition is met.

If you need help constructing your conditional expression, you can click the button that looks like a magic wand to the right of the expression text box. When you click this button, Access opens the Expression Builder, where you can build your conditional expression. (You learned about the Expression Builder in Chapter 3.) To the right of the word Then, Access displays a green up arrow. You can click this button if you want to move the position of the If block. (If there are actions below the If block, Access also displays a green down arrow.) If you move a block in error, you can click the Undo button on the Quick Access Toolbar. If you want to delete the If block, you can click the Delete button to the right of the up arrow. Below the arrow button are two links—Add Else and Add Else If. If you click the Add Else link, Access adds an Else branch to the If block, and if you click the Add Else If link, Access adds an Else If branch to the If block. (We'll explore these two conditional elements in just a moment.)

For the On Insert data macro that you have been building, we can use the Is Null phrase in our conditional expression to test whether the EmployeePicture field in the tblEmployees table has a value, an image file in this case, before Access saves the new employee record. In the conditional expression text box in the If block, type the letters **tblE** and notice that Access provides IntelliSense options for you, as shown in Figure 4-16.

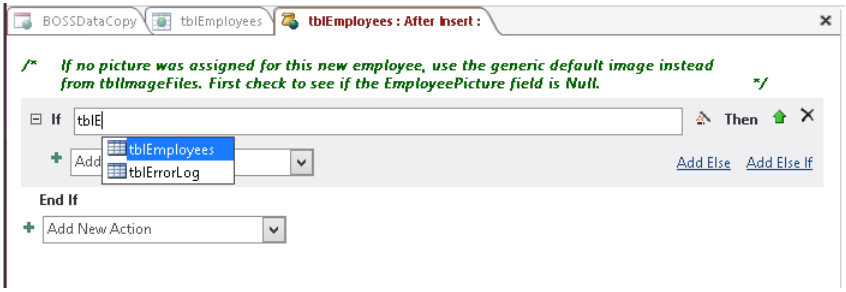


Figure 4-16 Access provides IntelliSense options whenever you are writing expressions in data macros.

You can continue to type tblEmployees, or use the down arrow to highlight the tblEmployees option from the IntelliSense drop-down list and then press Tab or Enter. Notice that after you select tblEmployees, Access adds brackets around the table name. Now type a period, and IntelliSense provides a list of all the field names in the tblEmployees table, as shown in Figure 4-17.

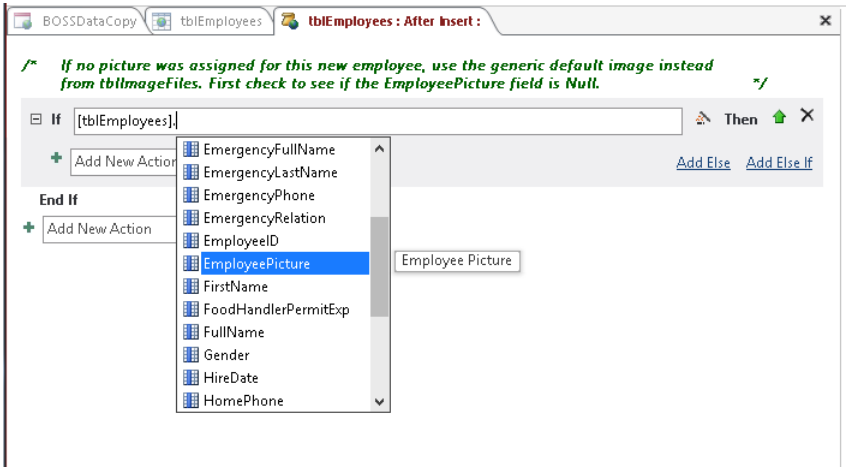


Figure 4-17 Access displays a list of all the fields in the tblEmployees table by using IntelliSense.

You can continue to type **EmployeePicture**, or use the down arrow to highlight the EmployeePicture field name from the IntelliSense drop-down list and then press Tab or Enter. Access also adds brackets around the EmployeePicture field name after you select it from the drop-down list. (Because our table name and field name contain no spaces, the brackets are not required, but its good practice to include them anyway.) Complete the entire expression by typing **Is Null**. Your completed expression should be `[tblEmployees].[EmployeePicture] Is Null`, as shown in Figure 4-18. Note that I also like to include the table name so that I know exactly what I'm referencing in my data macro logic, and I also get the benefit of being able to use IntelliSense.

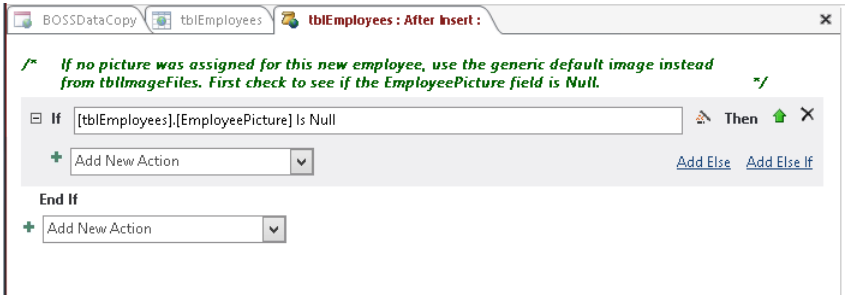


Figure 4-18 Your completed conditional expression should now look like this.

With your completed conditional expression for the If block, Access executes actions after the Then keyword and before the End If keywords only, if any employee record contains no data in the EmployeePicture image field.

INSIDE OUT **Nesting limitations in the Logic Designer**

The Logic Designer supports only 10 levels of nesting program flow constructs and data blocks. That is, you can nest up to nine additional constructs or data blocks inside a single top-level construct or data block (each one nested deeper inside the previous one).

Using LookupRecord data blocks to find records

The next step in our logic for the On Insert event of tblEmployees is to find a specific record in the tblImageFiles table where a default picture graphic is stored. To do this, tab or click into the Add New Action combo box that is inside the If block you completed in the previous section, type **LookupRecord**, and press Enter to add this data block inside the If block, as shown in Figure 4-19.

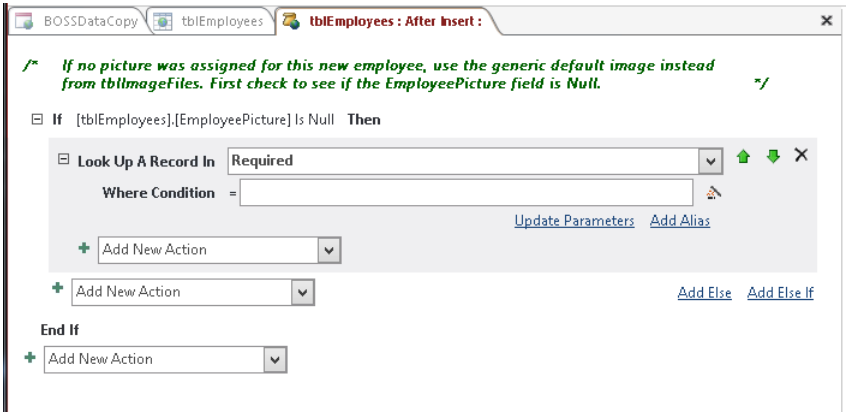


Figure 4-19 Add the LookupRecord data block inside the If block.

The LookupRecord data block takes four arguments:

- **Look Up A Record In.** Required argument. The name of a table or query to look up a record in.
- **Where Condition.** Optional argument. The expression that Access uses to select records from the table or query.

- **Update Parameters.** Optional argument. If you're looking up records in a query that requires parameters, you can provide them here.
- **Alias.** Optional argument. A substitute or shorter name for the table or query.

The only required argument for the LookupRecord data block is Look Up A Record In. Access provides a drop-down list for this argument that includes the names of all tables and saved query objects in your web app. If you want Access to look up a specific record in the specified table or query, you must provide a valid Where clause expression to find the record. If you leave the Where Condition argument blank, Access finds the first record in the specified table or query. You can click the button with the magic wand on it to open the Expression Builder to assist you with creating a Where clause if you'd like. The Update Parameters and Alias optional arguments are accessible through two links below the Where Condition argument on the right side. When you click these links, Access displays additional text boxes for you to enter these arguments. If you are looking up a record in a table, clicking the Update Parameters link does nothing, because tables do not contain parameters.

Before Access enters the LookupRecord block, the default data context is the incoming or changed record. The incoming record is either a new record or changes to an existing record. Within the LookupRecord block, Access creates a new data context. Access evaluates the Where condition of a data block with the same default context as when you are inside the data block. This means that if you do not use an alias as the table qualifier for field names in the Where condition argument, you are referring to a field within the new data context that you just created by using the data block.

Understanding alias and context

Using an alias is required when using a LookupRecord, ForEachRecord, EditRecord, or CreateRecord data block or DeleteRecord action, if you are trying to refer to a different data context other than the default. LookupRecord, ForEachRecord, and CreateRecord data blocks always create a new data context. EditRecord and DeleteRecord use only the current data context, unless you specify a different context to use. Consider the following example data macro logic:

```
ForEach Record in TableA Alias A
  LookupRecord in TableB Alias B WHERE B.TableBField1 = A.TableAField2
    EditRecord Alias A
      SetField TableAField3 = "Something"
```

In this example, the EditRecord's default context is on TableB's qualified row, so you have to use an alias to specifically indicate that the EditRecord is targeting TableA's looped row. You also need to use an alias to differentiate the data context for the same table. Consider the following example data macro logic:

```
On Insert Table1
  LookupRecord in Table1 Alias Lookup
    WHERE Lookup.ID <> Table1.ID AND Lookup.UserName = Table1.UserName
      RaiseError The user has already been added.
```

In this example, Table1 is the alias for the newly inserted row, while Lookup is the alias for the row being looked up in Table1.

Here are some considerations when working with data blocks:

- **When inside a LookupRecord or ForEachRecord data block, the default context is the active row in the looped table.**
- **When inside a CreateRecord data block, the default context is the new row Access is creating.**
- **In On Insert event data macros, the default data context, outside any data block, is the row that Access is inserting.**
- **In On Update event data macros, the default data context, outside any data block, is the new value of the updated row.**
- **In On Delete event data macros, the default context, outside any data block, is the row that Access is deleting.**

The tblImageFiles table is a system table that I use in this web app to hold any image files that I want to use in the app. In the On Insert event macro, you want to look up a record in this table, so click inside the Look Up A Record In argument and select tblImageFiles from the drop-down list. Currently, this table contains only one image file, but more images could be added over time. The specific image file you need for this example is the first record with ID=1. To make sure you look up the correct record, you should provide a Where clause that locates the first record every time. To do that, enter `[tblImageFiles].[ID]=1` in the Where Condition argument, as shown in Figure 4-20. When you start typing, IntelliSense helps you along and you can easily see and select the correct field name that holds the ID value. In this example, you already know that the default image needed is in the record that has ID=1. You could also use a Where clause that looks up the specific image description provided in the ImageDescription field.

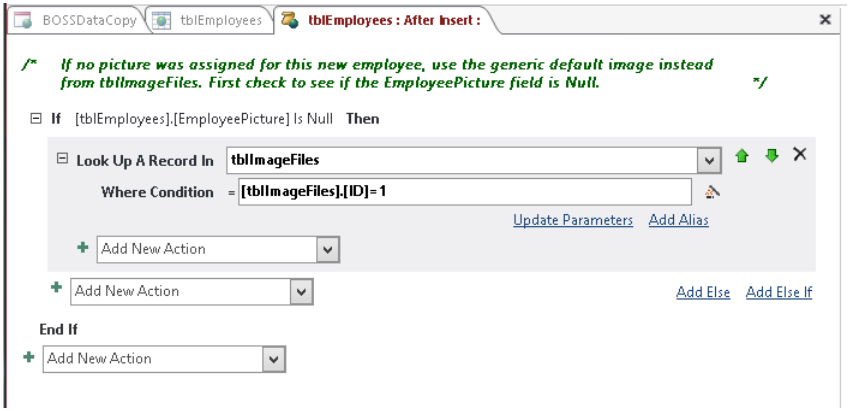


Figure 4-20 Add a Where clause to find a specific record using LookupRecord.

Using local variables

You can use a local variable in data macros to store a value that can be used throughout the execution of the data macro. Local variables are very useful when you need Access to calculate values during the execution of the data macro or remember something for later use in the data macro. You can think of a local variable in a data macro as writing yourself a note to remember a number, a name, or an email address so that you can recall it at a later time in the data macro. All local variables must have a unique name in the context of the data macro. To fetch, set, or examine a local variable, you reference it by its name. Local variables stay in memory until the data macro finishes executing, you assign it a new value, or until you clear the value.

In the previous section, you added logic for Access to look up a specific record in the tblImageFiles table. We now need to copy the contents of the image field, ImageFile in this case, to a local variable so that we can use it later in the event. The reason for this is because the code in this block is now executing in a different context and when Access finishes, we cannot make the outer code block refer to this context. Creating a local variable here allows us to pass a value back to a different context during the data macro execution. To create a local variable, click or tab into the Add New Action combo box that is inside the LookupRecord block, enter **SetLocalVar**, and press Enter to add this action inside the LookupRecord block, as shown in Figure 4-21.

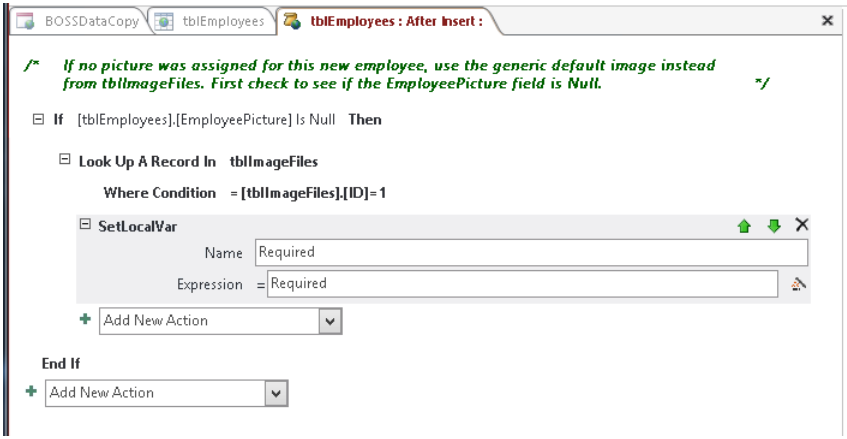


Figure 4-21 Add the SetLocalVar action inside the LookupRecord block.

The SetLocalVar action takes two required arguments:

- **Name.** Required argument. The name of the local variable you want to use to refer to during data macro execution.
- **Expression.** Required argument. The expression that Access uses to define the local variable.

For the Name argument, you can enter a name up to 64 characters. For the Expression argument, you can click the button that looks like a magic wand to open the Expression Builder to assist you with creating an expression. In this example, enter **varImage** into the Name argument and then enter **[tblImageFiles].[ImageFile]** into the Expression argument, as shown in Figure 4-22.

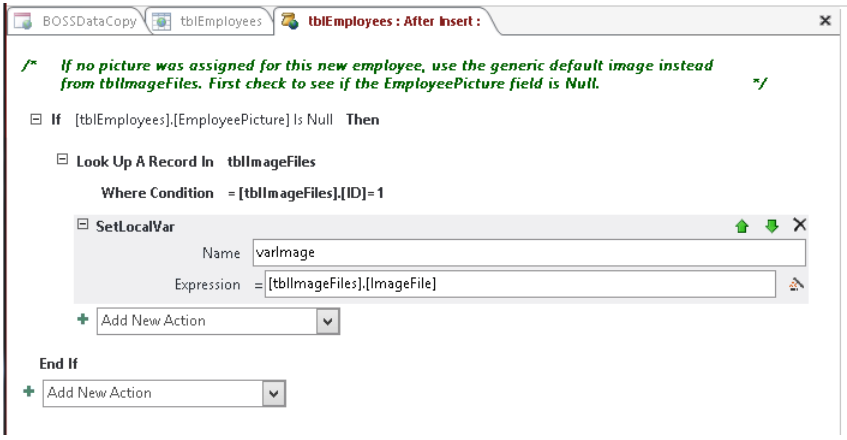


Figure 4-22 Enter a name and valid expression into the SetLocalVar arguments.

Choosing variable names in web apps

Access 2013 gives you lots of flexibility when it comes to naming your local variables, parameters, and return variables in web apps. (You'll learn about parameters and return variables later in this chapter.) A variable name can be up to 64 characters long and can include any combination of letters, numbers, and special characters except a period (.), exclamation point (!), square brackets ([]), leading equal sign (=), or nonprintable character such as a carriage return. You cannot use spaces in any part of variable names in web apps. The name also cannot contain any of the following characters: / \ : ; * ? "" < > | # <TAB> { } % ~ &. In general, you should give your variables meaningful names. You should also avoid using variable names that might match any name internal to Access. For example, all objects have a Name property, so it's a good idea to qualify a variable containing a name by calling it varVendorName or varCompanyName. You could also preface the variable name with the data type, such as strVendorName for text and imgEmployeeImage for image data types. You should also avoid names that are the same as built-in functions, such as Date, Time, Now, or Space. See Access Help for a list of all the built-in function names.

When Access finds the record in the tblImageFiles table where the ID field equals 1, it creates a local variable named varImage, reads the current value in the ImageFile field for that specific record, and then assigns the value of that field (a picture file, in this case) to the local variable. You can now reference and use this value in other areas of this same table event by referencing the variable by its name. We'll do that in just a moment. Let's save the logic we've created so far by clicking the Save button on the Quick Access Toolbar.

Note

You cannot save any data macro logic if any If, Else If, Or Else blocks are empty and have no actions inside them.

Collapsing and expanding actions

Now that you have the varImage local variable currently storing the contents of an image file, it's time to save that data to the EmployeePicture field in the tblEmployees table. To do this, you'll use the EditRecord data block. The tricky part of this next procedure though is to make sure you place the EditRecord data block in the correct place on the macro design surface. If you click anywhere on the LookupRecord data block you currently have on the macro design surface, you'll notice there are three Add New Action combo boxes near the bottom of the screen, as shown in Figure 4-23.

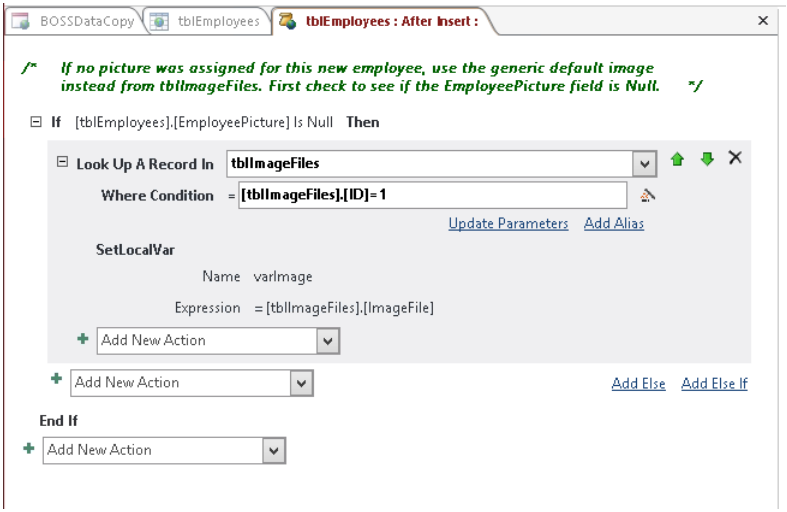


Figure 4-23 There are three Add New Action combo boxes at the bottom of the macro design surface.

We want to place the EditRecord data block outside and below the LookupRecord data block. Because you have the LookupRecord data block selected right now, it's a little easier to tell that the topmost Add New Action combo box is inside the LookupRecord data block, but if you did not have it selected, you might find it more difficult trying to decide where to place your next action. For example, compare the screen shots in Figure 4-22 and Figure 4-23 shown previously. In Figure 4-22, I selected the SetLocalVar data action and you'll notice that you can see only two Add New Action combo boxes. In Figure 4-23, I selected the LookupRecord and you can see three Add New Action combo boxes.

When you have complex data macros with many program flow constructs, data blocks, and data actions, you might find it harder to understand everything happening with the structure of your data macros, especially if you have to scroll the macro design surface to see everything. Fortunately, the Logic Designer includes features that can make these tasks easier.

To the left of the LookupRecord data block and the If block on the macro design surface, you'll notice that Access displays a box with a dash inside. If you place your mouse over the SetLocalVar data action, you can also see a similar box. (For data actions, Access shows this box only when you hover over the action.) You can use this box to expand and collapse the group or action. By default, the Logic Designer displays all group blocks and data actions in expanded mode so that you can see all actions and arguments. To collapse the LookupRecord data block, click inside the box. Access changes the dash inside the box to a plus symbol and then collapses the data block onto two lines, as shown in Figure 4-24.

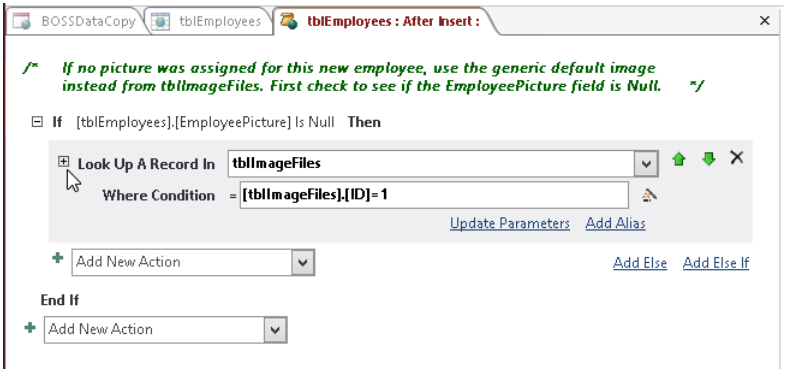


Figure 4-24 Click the box next to an action to collapse it.

Access displays the data block on two lines, and all actions contained inside the data block are hidden. It is much easier now to distinguish that the Add New Action combo box, directly below the highlighted LookupRecord data block, is outside that block. If you collapse a data action, such as the SetLocalVar action, Access displays the action without the argument names—Name and Expression for SetLocalVar—and separates the argument values with a comma. By collapsing data blocks and data actions, you can see more of the macro design surface. To expand the data block or data action again, click inside the box, now displaying a plus symbol, and Access expands the data block or data action.

You can collapse an entire Group block or If block as well using the same technique. If you want to collapse all data actions showing on the macro design surface at the same time, you can click the Collapse Actions button in the Collapse/Expand group on the ribbon. Click the Expand Actions button in the Collapse/Expand group on the ribbon to expand all data actions showing on the macro design surface.

For the maximum amount of space on the macro design surface, click the Collapse All button in the Collapse/Expand group on the ribbon. Access collapses all groups onto one line, as shown in Figure 4-25. You can't see very much with this view, of course. However, you can then selectively expand Groups, If blocks, and Data Blocks one at a time to work on specific parts of the data macro. Click the Expand All button on the ribbon to expand all Group blocks, If blocks, Data Blocks and Data Actions.

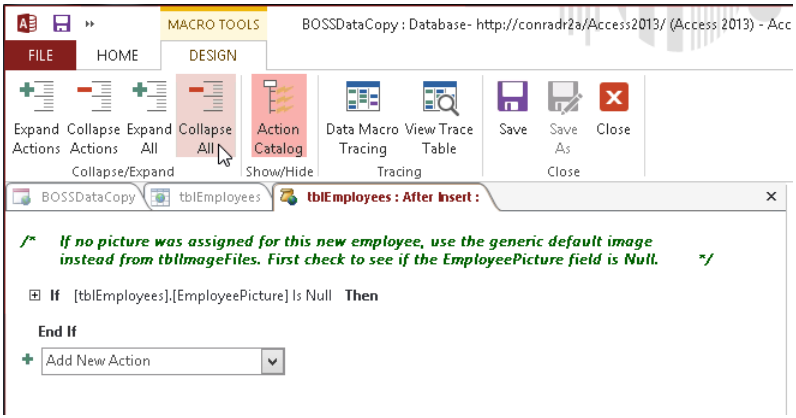


Figure 4-25 When you click the Collapse All button, Access collapses everything on the macro design surface except Comment blocks.

INSIDE OUT Viewing super tooltips

If you hover your mouse over a collapsed data action, Access displays a super tooltip with all the arguments. You can then view all the argument values of the data action easily, without having to expand the data action.

Note

When you expand or collapse Group blocks, If blocks, Data Blocks, or Data Actions, Access marks the macro design surface as dirty, even if you did not make any other changes. If you attempt to close the Logic Designer window, Access prompts you to save your changes. Access remembers the state of any expanded or collapsed elements when you save changes and reopen the data macro. Also, when you click Expand All after previously clicking Collapse All, Access displays all Comment blocks in a narrower box than before you collapsed everything. After you close and reopen the macro design surface, the width of the Comment blocks return to their normal size.

Now that you've collapsed the LookupRecord data block, let's continue adding our EditRecord data block. Click inside the Add New Action combo box below the LookupRecord data block, type **EditRecord**, and then press Enter. Access adds a new EditRecord data block onto the macro design surface, as shown in Figure 4-26.

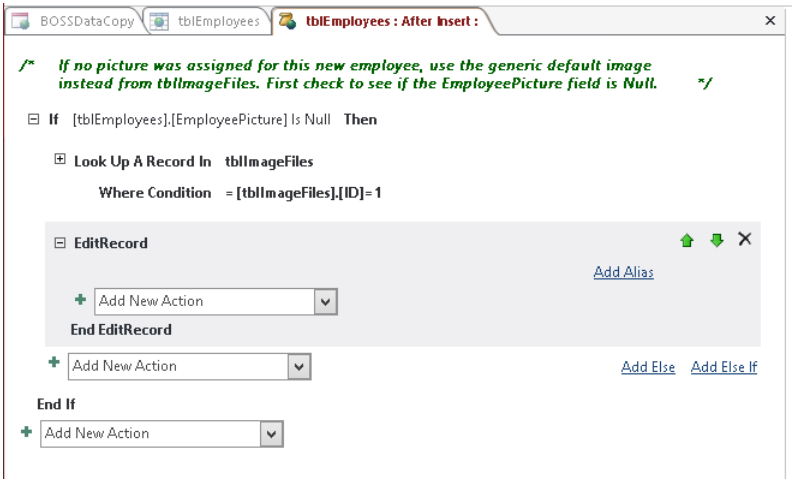


Figure 4-26 Add an EditRecord data block beneath the LookupRecord data block.

Whenever you want to change data in a table, you must use the SetField data action inside an EditRecord data block. Because our EditRecord data block is not inside any other data block such as ForEachRecord or LookupRecord, the context of the EditRecord block acts on the new record being created in the current table. For our example, we want to change the EmployeePicture field of the new employee record being created in tblEmployees to the local variable we defined earlier—varImage. Click inside the Add New Action combo box that is inside the EditRecord data block, type **SetField**, and then press Enter to add this new action to the macro design surface, as shown in Figure 4-27.

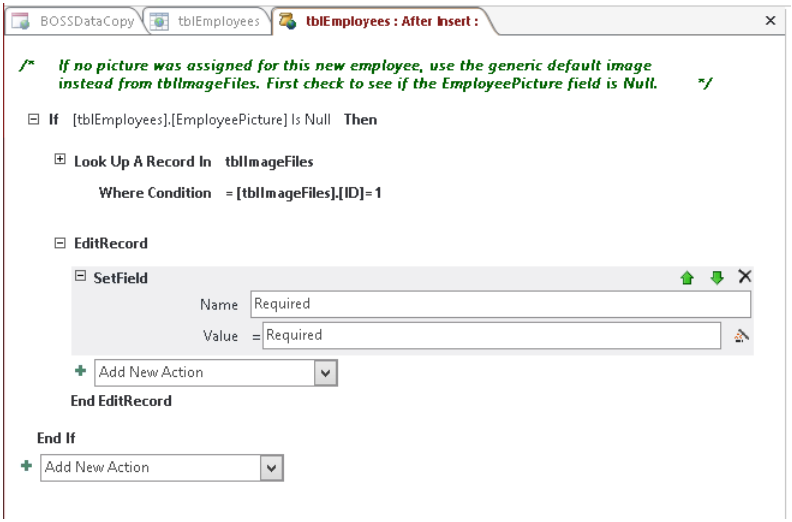


Figure 4-27 The SetField data action allows you to commit data to fields inside data macros.

The SetField action takes two required arguments, Name and Value. In the Name argument, we use the full table name and field name to clearly indicate which field we want to update. Enter **[tblEmployees].[EmployeePicture]** into the Name argument, and enter **[varImage]** into the Value argument. Notice that when you start typing the table name in the Name argument, Access provides IntelliSense to help you pick the correct table and field name you want. Also, you'll notice that Access does not add brackets around the table name and field name when using IntelliSense in this context, but it's a good idea to always include them even if you don't have spaces in your table and field names. If you do not provide brackets around the local variable name in the Value argument, Access adds them when you save and re-open the macro design window.

Click Save on the Quick Access Toolbar to save your changes to the On Insert event. Your completed changes to the data macro should now match Figure 4-28. Notice that, in Figure 4-28, I expanded all the actions again by clicking Expand All button in the ribbon.

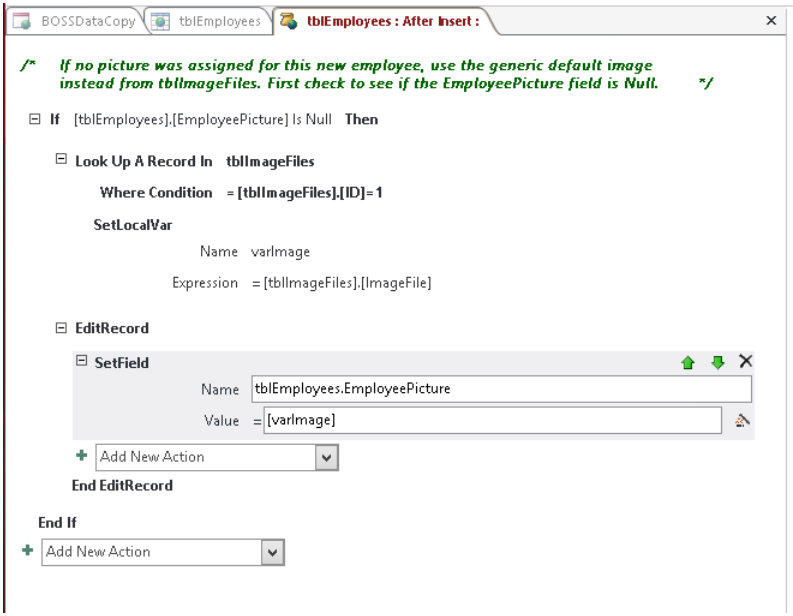


Figure 4-28 Your On Insert data macro up to this point should now look like this.

The data macro logic you've now defined instructs Access to check every new employee record entered into this table. If no picture is provided in the EmployeePicture at the time you create a new employee record, Access looks up a record in the tblImageFiles table where the ID value equals 1, stores the value of the ImageFile default picture into a local variable called varImage, and finally saves that default picture into the EmployeePicture field for that new record using the local variable. Note that datasheets do not support

displaying image fields, so you would have to verify this using your web browser and a List Details or Blank view.

Moving actions

As you design data macros or user interface macros in the Logic Designer, you might find that you need to move actions around as the needs of your application change. In the On Insert event for the tblEmployees table you've been working on, it would be good to add in some comments for the extra actions you just finished. As with many areas of Access, there is usually more than one way to accomplish a task. You could drag a Comment block from the Action Catalog onto the macro design surface, or you could add comments anywhere on the macro design surface and then move them into different positions. The Logic Designer makes the task of moving data blocks, data actions, and all other elements around the macro design surface very easy.

Open the tblEmployees table in Design view if you closed it, click the Design contextual ribbon tab under Table Tools, and then click the On Insert button in the Events group. You should now see the data macro that you created previously for saving a default picture graphic for each new employee record if you don't provide one. Click into the Add New Action combo box at the bottom of the macro design surface, type **Comment**, and then press Enter to add a new Comment block to the macro design surface. Type the following text into the Comment block to identify one of the tasks in this data macro:

It is Null so lookup the default image in tblImageFiles and set a local variable to the picture.

Add one more new Comment block as well to the bottom of the macro design surface using the same technique, and then type the following text into this new block:

Now update the EmployeePicture field with that image data.

Your macro logic should now match Figure 4-29.

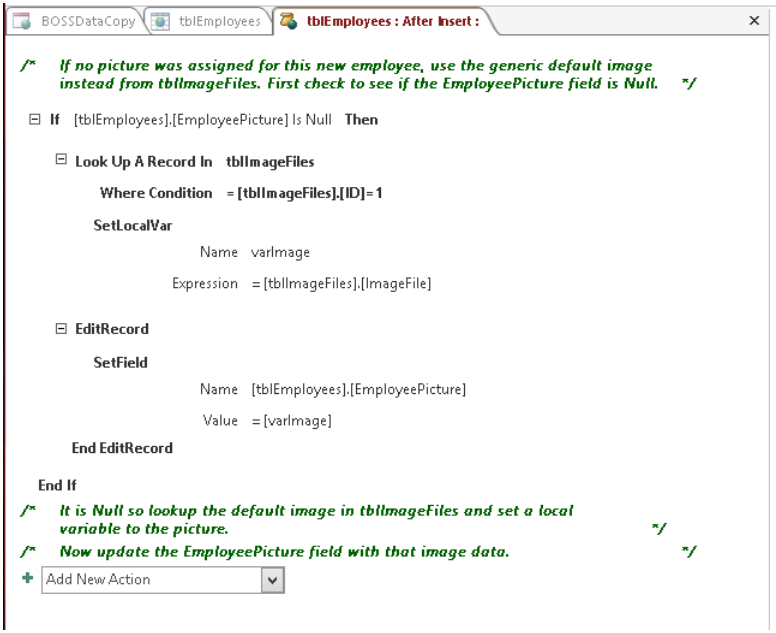


Figure 4-29 Your macro logic should now have two Comment blocks at the bottom of the macro design surface.

We want to move the first Comment block above the LookupRecord block and below the If condition line. To move the first Comment block you just added, click anywhere on the Comment block, hold the mouse key down, drag the Comment block up above the LookupRecord block until Access displays a horizontal bar above the LookupRecord block, as shown in Figure 4-30, and then release the mouse.

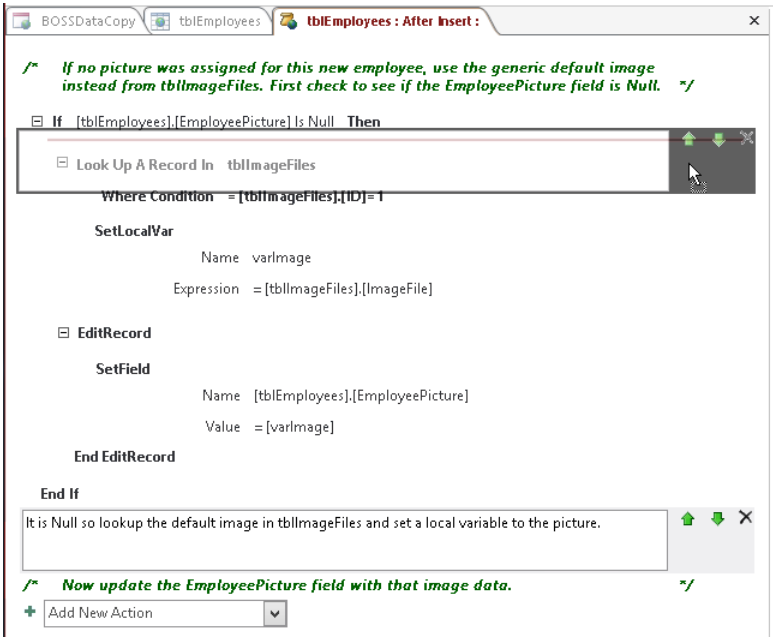


Figure 4-30 Drag the Comment block up above the LookupRecord block.

Access places the Comment block inside the If block and above the LookupRecord block. Instead of using the drag technique, you could also click the up arrow button on the far side of the Comment to move it up into the correct position. When you click the up arrow button, Access moves the selected action up one position in the macro design surface. In our example, it would take seven clicks of the up arrow to move the first Comment block action up above the LookupRecord block.

INSIDE OUT

Creating a duplicate copy of logic

To duplicate any logic on the macro design surface, you can hold the Ctrl key down and then drag to a different location. Access creates an exact copy of the program flow construct, data block, or data action, including any argument information.

You might find it easier to use the keyboard rather than the mouse to move actions around the macro design surface. Table 4-2 lists the keyboard shortcuts for working inside the Logic Designer.

TABLE 4-2 Keyboard shortcuts for logic designer

Keys	Action
Ctrl+F2	Opens the Expression Builder dialog box if you are in an expression context
Ctrl+Space	Calls up IntelliSense in expression contexts
Ctrl+Up arrow	Moves selected action up
Ctrl+Down arrow	Moves selected action down
Shift+F2	Opens the Zoom Builder dialog box
Shift+F10	Opens a context-sensitive shortcut menu
Left arrow	Collapses action
Right arrow	Expands action

Now that you've moved the first new Comment block to the correct position, let's move the last Comment block as well. Highlight the Comment block at the bottom of the macro design surface, hold the mouse key down, drag the Comment block up above the EditRecord, and then release the mouse. Your completed data macro should now look like Figure 4-31.

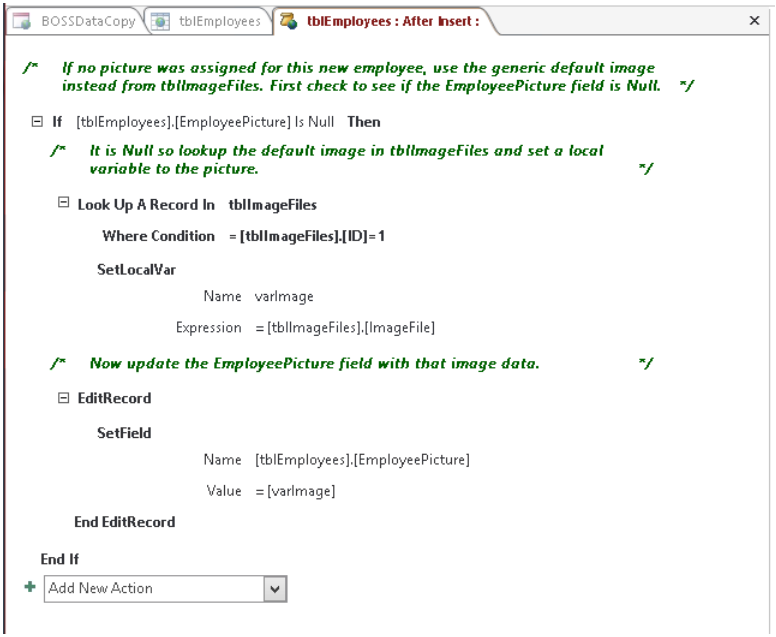


Figure 4-31 Your data macro should now look like this after you move the last Comment block.

You've now successfully revised the data macro logic by adding in more Comment blocks and moving them around the macro design surface. You've completed all the steps necessary to ensure that every new employee record added to this table contains an image in the EmployeePicture field. If the user creating a new employee record provides an image for the EmployeePicture field, Access evaluates the If block condition as False and then takes no action. If the new record does not contain an image for the EmployeePicture field, Access reads the contents of the tblImageFiles table and copies an image from that table into the new employee record. Save your changes, and then close the Logic Designer window.

Studying other On Insert events

The Back Office Software System sample web app includes On Insert events attached to other tables besides the two examples you've already seen. You can explore the data macros attached to these events for additional examples.

- **tblAppointments.** Syncs two time display fields with values from the tblTimeLookups table. This breaks normalization, but it is needed to work around some user interface limitations.
- **tblCompanyInformation.** Prevents additional records from being added to this system table.
- **tblEmployees.** Ensures that each new employee record contains an employee picture. Uses LookupRecord to insert a default image if no picture exists.
- **tblInventoryLocations.** Finds the next highest sort order number and sets the SortOrder field to that value for the new record.
- **tblInvoiceDetails.** Checks to see whether the invoice is balanced with the invoice details after each new record is created. Uses a RunDataMacro action to execute a named data macro and passes in a parameter with each new record.
- **tblLaborPlanDetails.** Syncs two time display fields with values from the tblTimeLookups table. This breaks normalization, but it is needed to work around some user interface limitations.
- **tblSchedule.** Syncs two time display fields with values from the tblTimeLookups table. This breaks normalization, but it is needed to work around some user interface limitations.
- **tblSettings.** Prevents additional records from being added to this system table.
- **tblTerminations.** Whenever a new termination record is created for an employee, this data macro marks the employee record as inactive. The data macro logic looks

up the employee's record in the tblEmployees table and sets the Boolean Active field to False for that specific employee.

- **tblTimeLookups.** Prevents additional records from being added to this system table.
- **tblTrainedPositions.** Ensures that each employee has only one trained position marked as their primary position. Uses a RunDataMacro action to execute a named data macro and passes in two parameters with each new record.
- **tblWeekDays.** Prevents additional records from being added to this system table.

Using On Update events

The On Update event fires whenever Access completes the operation of committing changes to an existing record in a table. In the tblTerminations table, I have a data macro defined in the On Insert event to mark an employee's Active field to False whenever I create a termination record. In Figure 4-32, you can see the data macro logic for the On Insert of the tblTerminations table. When you create a new termination record in the Back Office Software System web app, Access looks up the corresponding employee's record in tblEmployees using the LookupRecord data block and then changes the Yes/No Active field in that table to No using EditRecord and SetField.

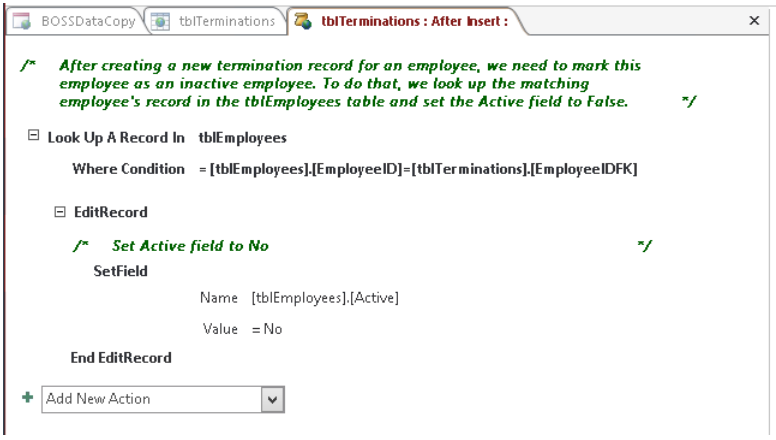


Figure 4-32 The On Insert event of tblTerminations includes logic to mark an employee inactive.

However, what happens if we accidentally select the wrong employee when we save the new termination record? We now have a situation where two employee records are inaccurate. We have one employee marked as inactive, which shouldn't be the case, and another employee still marked as active even though he or she should not be active. To fix this discrepancy manually, you would need to change the data in the existing termination

record to use the correct employee, change the Active field of the employee's record to Yes for the employee to whom you first assigned the termination record, and also change the Active field to No for the employee who now has the termination record assigned to him or her. Instead of doing all these steps manually, we can use the On Update event to fix both employee records.

Open the tblTerminations table in Design view. Next, click the Design contextual tab under Table Tools, and then click the On Update button in the Events group to open the Logic Designer, as shown in Figure 4-33.

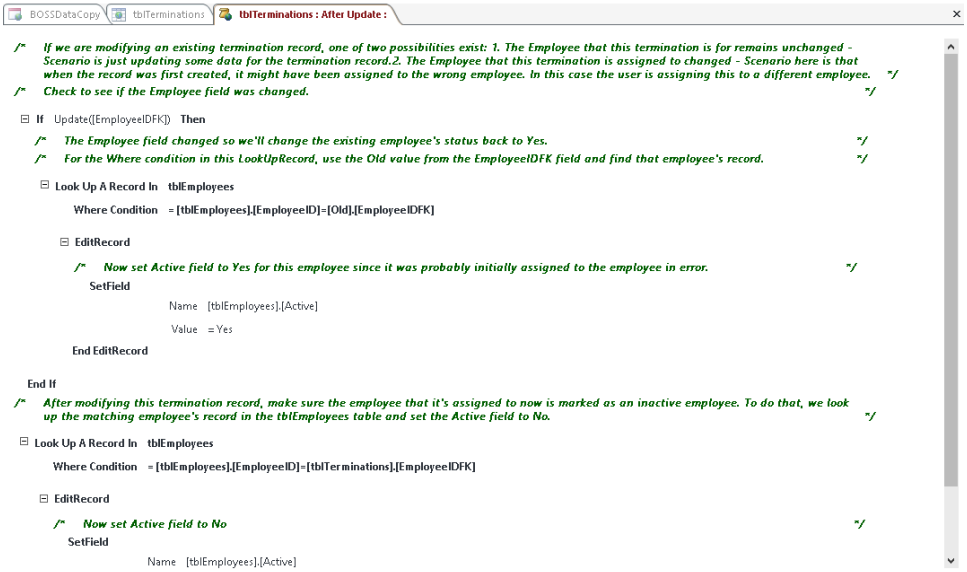


Figure 4-33 Click the On Update button on the ribbon to examine the On Update event of the tblTerminations table.

The data macro logic for the On Update event is as follows:

```
Comment Block: If we are modifying an existing termination record, one of two
possibilities exist: 1. The Employee that this termination is for remains unchanged -
Scenario is just updating some data for the termination record. 2. The Employee that
this termination is assigned to changed - Scenario here is that when the record was
first created, it might have been assigned to the wrong employee. In this case the
user is assigning this to a different employee.
Comment Block: Check to see if the Employee field was changed.
If Update([EmployeeIDFK]) Then
  Comment Block: The Employee field changed so we'll change the existing employee's
  status back to Yes.
  Comment Block: For the Where condition in this LookupRecord, use the Old value from
  the EmployeeIDFK field and find that employee's record.
  Look Up A Record In tblEmployees
```

```
    Where Condition = [tblEmployees].[EmployeeID]=[Old].[EmployeeIDFK]
      EditRecord
        Comment Block: Now set Active field to Yes for this employee since it was
        probably initially assigned to the employee in error.
        SetField
          Name: [tblEmployees].[Active]
          Value: Yes

      End EditRecord
End If
Comment Block: After modifying this termination record, make sure the employee that
it's assigned to now is marked as an inactive employee. To do that, we look up the
matching employee's record in the tblEmployees table and set the Active field to No.
Look Up A Record In tblEmployees
    Where Condition = [tblEmployees].[EmployeeID]=[tblTerminations].[EmployeeIDFK]
      EditRecord
        Comment Block: Now set Active field to No.
        SetField
          Name: [tblEmployees].[Active]
          Value: No

      End EditRecord
```

The first part of the data macro includes two Comment blocks to indicate the purpose of this event. Next, I use an If condition using the *Update* function to see whether the EmployeeIDFK field changed. The Update function takes one argument, a field name, and returns True if the field is dirty and returns False if the field is not dirty during the record update. For this On Update data macro, I can use the Update function in a conditional expression to test whether a user is attempting to change the value of the EmployeeIDFK field. If the EmployeeIDFK field changed, I know the user is assigning this existing termination record to a different employee. I then go into a LookupRecord data block and use the tblEmployees as the source. In the Where condition argument for the LookupRecord data block, I want to look up the EmployeeID in the table that matches the EmployeeIDFK field found in the tblTerminations table that Access is committing. When Access finds the matching record, it enters into the EditRecord block. Whenever you want to change data in another table in data macro events, you must use the SetField action inside an EditRecord block. For this example, I want to change the Active field of the matching employee to No to indicate that he or she is no longer an active employee in the app. In the Name argument for the SetField action, I use the table and field name, tblEmployees and Active, respectively, for the LookupRecord block. My Where condition argument for the LookupRecord uses the *Old* property. The Old property returns the value of the field before Access changed its value in the process of saving the record. My Where condition argument is therefore the following:

```
[tblEmployees].[EmployeeID]=[Old].[EmployeeIDFK]
```

To help understand this concept, imagine the value of the EmployeeIDFK field is currently 13, the record for Mario Kresnadi, in the existing termination record. If you change the EmployeeIDFK field to Jeff Conrad, EmployeeID of 31, the Old value for that field is 13 and the new value after saving the record is 31. By referencing the Old value of the EmployeeIDFK field, I can determine which employee this termination record used to be assigned to. (There is no New property available when creating data macros because the new value is simply the committed value of the field, and you can refer to it by using the field name.)

After Access finds the EmployeeIDFK that the termination record used to be assigned to, I use a SetField data action to set the Active status of that employee back to Yes. It's my assumption that if the user is assigning the termination record to a different employee, I'll error on the side of caution and assume this employee's status should be changed back to Yes.

The first part of the data macro logic is inside an If block. Based on the logic, if the user did not change the EmployeeIDFK field, Access does not change anything in the first part of the data macro. The second part of the On Update event is outside the If block, which means this part of the data macro logic runs every time a user changes anything about a termination record. I use another LookupRecord data block to look up a different employee record in the employee table. In this case, the Where condition argument is the following:

```
[tblEmployees].[EmployeeID]=[tblTerminations].[EmployeeIDFK]
```

This time, Access looks for the EmployeeID in the table that matches the now-committed value in the EmployeeIDFK field in the tblTerminations table. In the previous example, this means Access looks for the EmployeeID of Jeff's record, which is 31. Finally, I set the Active status of that employee's record to No because this termination record is now assigned to that employee.

To test this On Update event, close the Logic Designer window by clicking the Close button in the Close ribbon group. Open the tblTerminations table in Datasheet view now by right-clicking the tblTerminations object tab in the application window and selecting Open from the shortcut menu or clicking the View button in the Views ribbon group and selecting Datasheet view from the drop-down menu. Find the existing termination record in this table—the one assigned to Mario Kresnadi. Tab over to the EmployeeIDFK for this record (the datasheet caption of the field displays Employee), type **Conrad** into the control where it currently says Mario Kresnadi, and then select Jeff Conrad from the drop-down list of employee names displayed in the EmployeeIDFK field, as shown in Figure 4-34. Now, click or tab off the record, and Access saves the record with Jeff Conrad's EmployeeIDFK number.

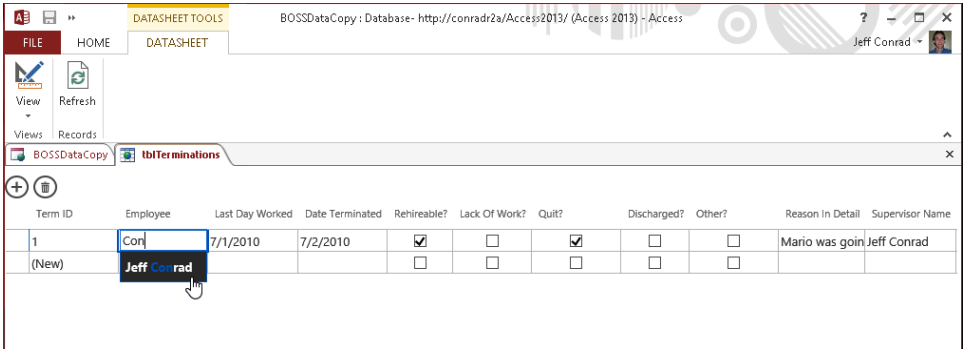
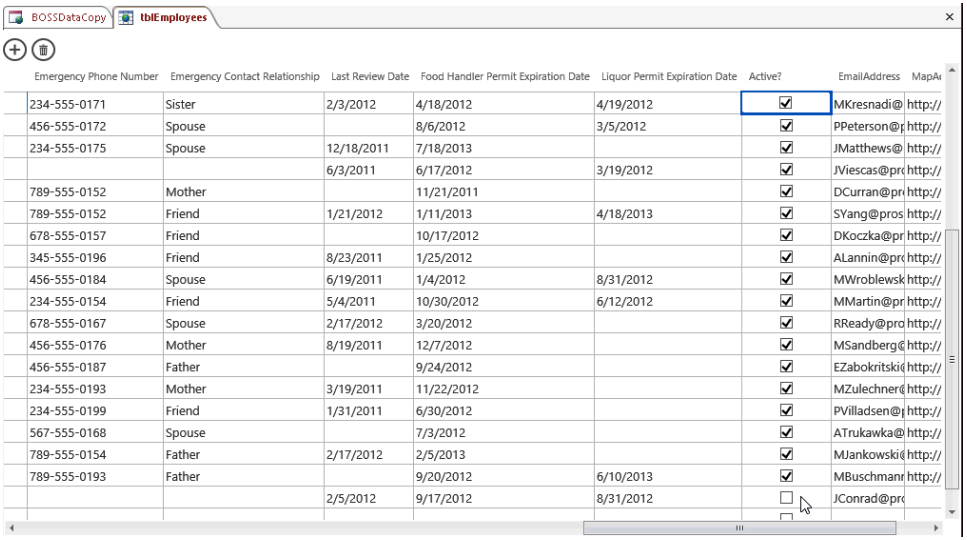


Figure 4-34 Change the EmployeeIDFK field from Mario Kresnadi to Jeff Conrad, and then save the record.

The control in the datasheet shown in Figure 4-34 for the EmployeeIDFK field is an autocomplete control, which is new in Access 2013. You'll learn more about this control in Chapter 6, "Working with views and the web browser experience."

To see the effects of this On Update event, open the tblEmployees table in Datasheet view by right-clicking the tblEmployees object in the Navigation pane and selecting Open from the shortcut menu. After you have the tblEmployees table open in datasheet view, scroll down to the record for the employee record for Mario Kresnadi. You'll notice that the Active field for Mario Kresnadi is now set to Yes, as shown in Figure 4-35. You'll also notice that Jeff Conrad's record shows his Active status is now set to No. In Figure 4-35, Mario's record is the record at the top (the highlighted record) and Jeff's record is at the bottom (where the mouse cursor is pointing).



Emergency Phone Number	Emergency Contact Relationship	Last Review Date	Food Handler Permit Expiration Date	Liquor Permit Expiration Date	Active?	EmailAddress	MapAd
234-555-0171	Sister	2/3/2012	4/18/2012	4/19/2012	☑	MKresnadi@	http://
456-555-0172	Spouse		8/6/2012	3/5/2012	☑	PPeterson@	http://
234-555-0175	Spouse	12/18/2011	7/18/2013		☑	JMatthews@	http://
		6/3/2011	6/17/2012	3/19/2012	☑	JViescas@pr	http://
789-555-0152	Mother		11/21/2011		☑	DCurran@pr	http://
789-555-0152	Friend	1/21/2012	1/11/2013	4/18/2013	☑	SYang@pros	http://
678-555-0157	Friend		10/17/2012		☑	DKoczka@pr	http://
345-555-0196	Friend	8/23/2011	1/25/2012		☑	ALannin@pr	http://
456-555-0184	Spouse	6/19/2011	1/4/2012	8/31/2012	☑	MWroblewsk	http://
234-555-0154	Friend	5/4/2011	10/30/2012	6/12/2012	☑	MMartin@pr	http://
678-555-0167	Spouse	2/17/2012	3/20/2012		☑	RReady@pro	http://
456-555-0176	Mother	8/19/2011	12/7/2012		☑	MSandberg@	http://
456-555-0187	Father		9/24/2012		☑	EZabokritski	http://
234-555-0193	Mother	3/19/2011	11/22/2012		☑	MZulechner@	http://
234-555-0199	Friend	1/31/2011	6/30/2012		☑	PVilladsen@	http://
567-555-0168	Spouse		7/3/2012		☑	ATrukawka@	http://
789-555-0154	Father	2/17/2012	2/5/2013		☑	MJankowski	http://
789-555-0193	Father		9/20/2012	6/10/2013	☑	MBuschmann	http://
		2/5/2012	9/17/2012	8/31/2012	☐	JConrad@pr	

Figure 4-35 Access changes the Active field for both Jeff's and Mario's records from the On Update event of the tblTerminations table.

With the data macro logic that we have defined in the On Update event, Access automatically maintains the Active status of the employee records. If the user assigns the termination record to a different employee, Access changes the Active status of two different employees. If the user changed information other than the EmployeeIDFK field, Access marks that employee as inactive again just to be safe.

The Back Office Software System sample web app includes On Update events attached to ten tables. You can explore the data macros attached to these events for additional examples of using the On Update event.

- **tblAppointments.** Syncs two time display fields with values from the tblTimeLookups table. This breaks normalization, but it is needed to work around some user interface limitations. It uses the Update function to determine whether the time fields changed.

- **tblEmployees.** Ensures that each employee record contains an employee picture. Uses Update function and LookupRecord to insert a default image if you remove the existing employee picture.

- **tblInvoiceDetails**. Checks to see whether the invoice is balanced with the invoice details after any record changes. Uses a RunDataMacro action to execute a named data macro and passes in a parameter with each record update.

- **tblInvoiceHeaders.** Checks to see whether the invoice is balanced with the invoice details but only if the InvoiceTotal field is changed by using the Update function. Uses a RunDataMacro action to execute a named data macro and passes in a parameter with each record update.
- **tblLaborPlanDetails.** Syncs two time display fields with values from the tblTimeLookups table. This breaks normalization, but it is needed to work around some user interface limitations. It uses the Update function to determine whether the time fields changed.
- **tblSchedule.** Syncs two time display fields with values from the tblTimeLookups table. This breaks normalization, but it is needed to work around some user interface limitations. It uses the Update function to determine whether the time fields changed.
- **tblTerminations.** Ensures that the correct employee records are marked as active or inactive if the existing record is assigned to a different employee.
- **tblTimeLookups.** Prevents any changes to existing records in this system table.
- **tblTrainedPositions.** Ensures that each employee has only one trained position marked as their primary position. Uses a RunDataMacro action to execute a named data macro and passes in two parameters with record change.
- **tblWeekDays.** Prevents any changes to existing records in this system table.

Using On Delete events

The On Delete event fires whenever Access attempts the operation of deleting a record from the table. There are many entry points for deleting a record when you are working with Access web apps. For example, you can delete a record in a table or query datasheet from within Access, you can run a named data macro that deletes a record, you can delete a record when using a view in your web browser, or you can delete records using user interface macros. When you attach a data macro to the On Delete event, Access runs the data macro logic no matter where the entry point is for deleting a record.

Earlier in the chapter, you created a data macro attached to the On Insert event of the tblWeekDays system table for the Back Office Software System sample web app data copy (BOSSDataCopy.app). The data macro you created prevents any additions to this system table. There is data macro logic attached to the On Update event that prevents any changes to the existing data as well. You can also lock tables down further by preventing any records from being deleted by using a data macro attached to the On Delete event.

For this example, open the tblCompanyInformation table in Datasheet view, click the Design contextual tab under Table Tools, and then click the On Delete button in the Events group to open the Logic Designer, as shown in Figure 4-36. This table contains only one record to hold important company information. We don't want any new records added to this table, and we also don't want to delete the existing record.

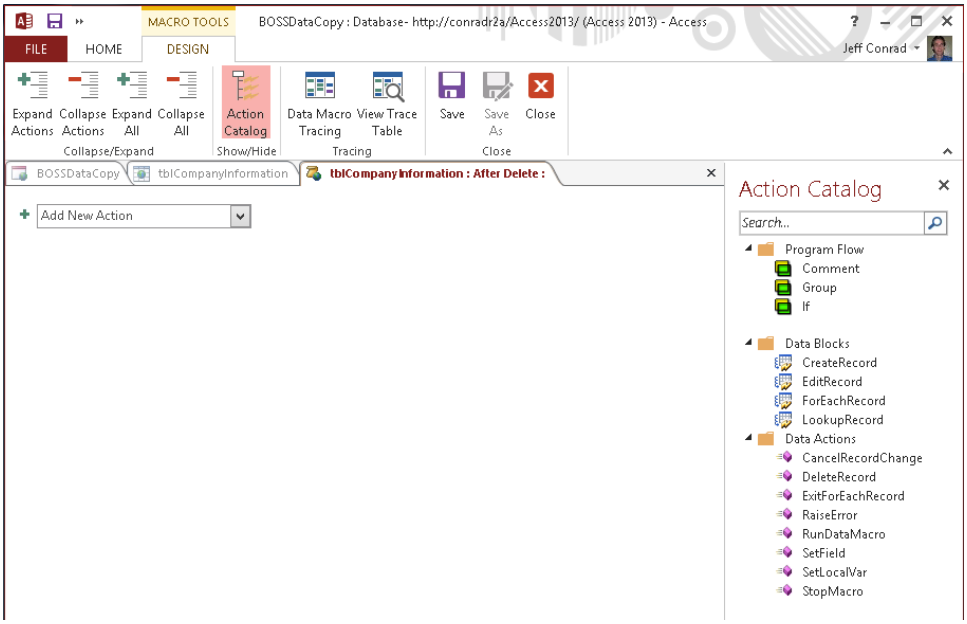


Figure 4-36 Click the On Delete button on the ribbon to open the Logic Designer.

We should first add a Comment block to this data macro so that anyone looking at it can understand the purpose of the logic in this On Delete event. You should now be familiar with the different methods of adding a new Comment block to the macro design surface. Drop a new Comment block onto the macro design surface, and enter the following text:

Don't allow the default record to be deleted.

Now add a RaiseError data action below the Comment block. For the Error Description argument, enter the following text:

You cannot delete the record from this system table; it is used in other areas of the application.

Your completed changes to the On Delete event should match Figure 4-37.

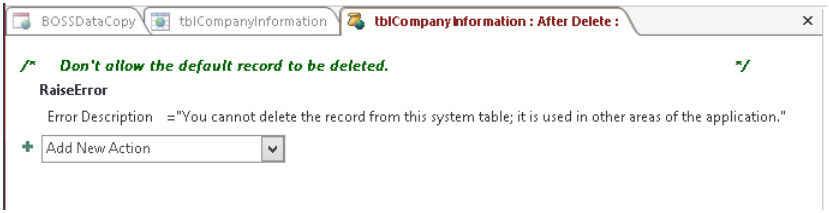


Figure 4-37 Your completed On Delete event logic should match this.

Seems almost too simple doesn't it? Simple, yes, but completely effective. We don't need to test for any special conditions for our scenario; we just need to throw an error if this event ever occurs. To try this, save the changes to this data macro by clicking the Save button in the Close group or the Save button on the Quick Access Toolbar. Next, close the Logic Designer window by clicking the Close button in the Close group. Finally, click the record selector next to the existing record in the tblCompanyInformation table in Datasheet view and press Delete. Access first displays a confirmation dialog asking you to confirm that you want to delete the record. Click Yes to confirm the deletion, and then Access displays the custom message in the RaiseError data action, as shown in Figure 4-38.

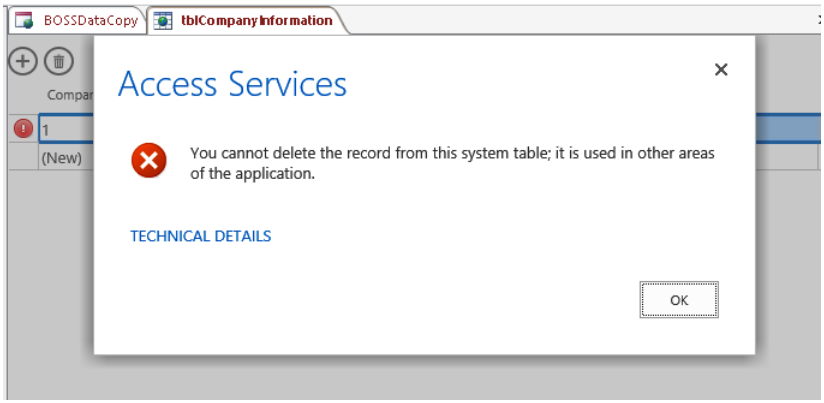


Figure 4-38 When you attempt to delete a record in the tblCompanyInformation table, Access displays your error message.

Note

In the On Delete event example we just discussed, the tblCompanyInformation table contains no relationships to other tables. If you have a Restrict Delete relationship enforced on any related tables, such as the tblWeekDays table has with other tables, Access prevents deletes and displays an internal message about not being able to delete the record. In this case, Access does not even show your On Delete RaiseError message. You might be asking why this is even necessary to put an On Delete data macro to prevent deletes if a Restrict Delete relationship is enforced on any related tables. You are correct that Access prevents deletes in this case; however it is possible that for a specific record in tblWeekDays, no related records exist in the other tables. In that case, a user could still delete a record from a static table that you don't want modified. Also, you might have other tables in your web app that do not have relationships with other tables and want to prevent any records from being deleted. Both the tblCompanyInformation and tblSettings tables in the Back Office Software System sample web app are two such examples where no relationships exist with other tables, but I want to prevent any record deletions.

The Back Office Software System sample web app includes On Delete events attached to other tables that use this same technique to prevent records from being deleted as well as other scenarios involving updating other tables when you delete records. You can explore the following data macros attached to these events for additional examples of using the On Delete event.

- **tblCompanyInformation.** Prevents deletion of existing records.
- **tblInvoiceDetails.** Checks to see whether the invoice is balanced with the invoice details after any record changes. Uses a RunDataMacro action to execute a named data macro and passes in a parameter with each record update. Uses the Old property to determine the ID of the invoice during the delete and passes that into the named data macro.
- **tblSettings.** Prevents deletion of existing records.
- **tblTerminations.** Ensures that the employee record is marked as active when deleting the termination record. Uses the Old property to determine the ID of the employee during the delete and finds the correct record using a LookupRecord data block.
- **tblTimeLookups.** Prevents deletion of existing records.
- **tblWeekDays.** Prevents deletion of existing records.

Deleting table events

If you want to delete a table event in a web app, you'll have to manually delete all of the data macro logic yourself. In Chapter 22, you'll learn that desktop databases include a dialog where you can quickly view all of the table events attached to tables in your application and delete any table event using this dialog. However, Access 2013 web apps do not include a similar type of dialog. To delete a table event in a web app, you need to open the table in Design view, delete each program construct, data block, and data action, and then save and close the Logic Designer. When you remove everything from the macro design surface for the specific table event, Access no longer executes that table event. Although it might seem tedious to delete each element on the macro design surface one by one, you can select everything currently displaying on the macro design surface by pressing Ctrl+A, as shown in Figure 4-39. When you have all data macro logic selected, press the Delete key to remove all logic from the macro design surface in one quick step. Now that you have everything removed, you can then save and close the Logic Designer.

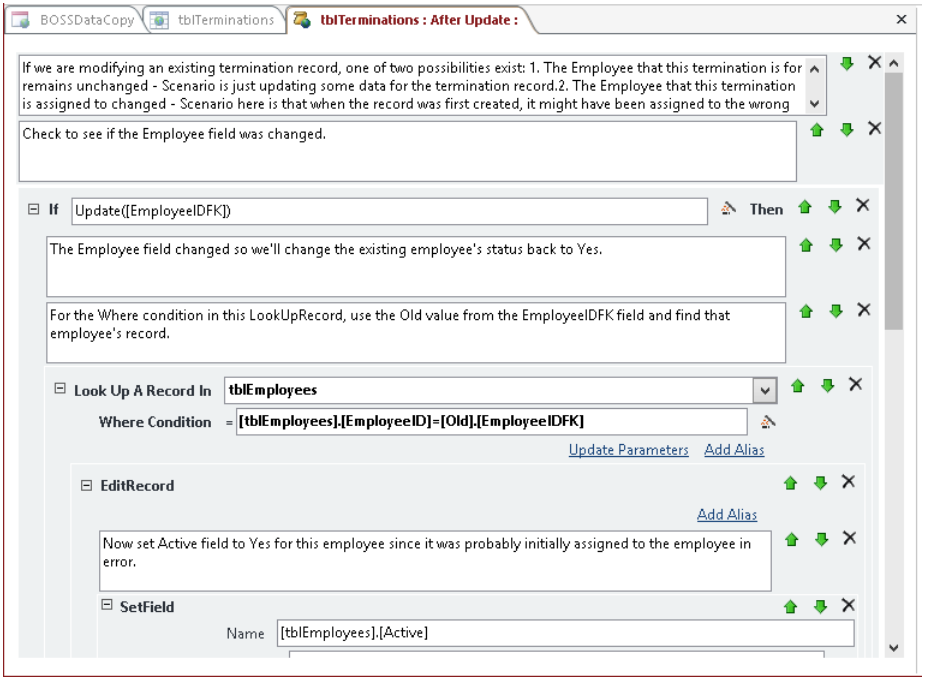


Figure 4-39 You can highlight all data macro logic in a table event and press Delete to quickly remove a table event.

Working with named data macros

So far in this chapter, you've been studying data macros attached to specific table events. Access also supports creating named data macros in web apps. A named data macro appears in the Navigation pane under the Macros group and is not attached directly to a specific table event. Named data macros in web apps execute only when called from another data macro or a user interface macro. Logic that is in a named data macro can interact with data in any table, require parameters before executing, and return data to the calling data macro or user interface macro. The Back Office Software System sample web app includes more than a dozen named data macros in the Navigation pane. In the next sections, you'll explore a few of these named data macros, as well as create a new named data macro.

Creating named data macros

In the Back Office Software System sample data copy web app (BOSSDataCopy.app), a table called tblTrainedPositions is used to track all the job positions each specific employee is trained to perform. A multiple-field index on this table ensures that each employee cannot be listed as trained in the same job position more than once. However, we also want to ensure that each employee has only one position marked as their primary job position. We can create a named data macro for this purpose, which can then be called from other areas of the app. To accomplish this goal, we'll create a new named data macro not attached to any table event and then call this named data macro from both the On Insert and On Update events of the tblTrainedPositions table.

Open the BOSSDataCopy.app sample web app within Access by downloading it from the Access Services site if you've closed the app. Now click the Advanced button in the Create group on the Home ribbon, and then click the option called Data Macro in the drop-down list, as shown in Figure 4-40.

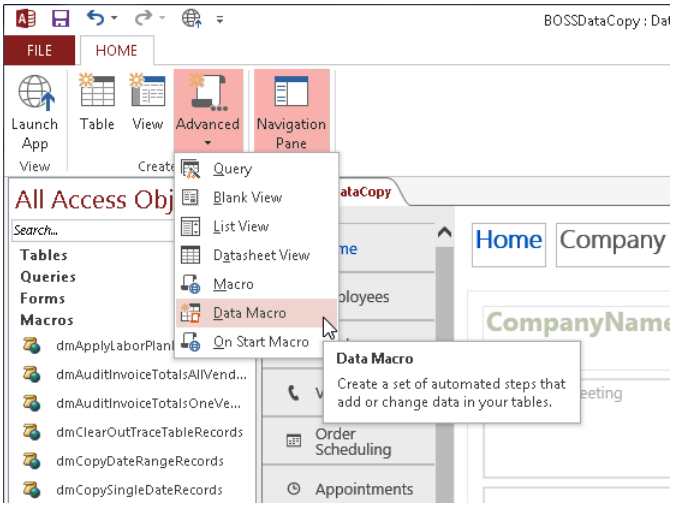


Figure 4-40 Click the Data Macro option under the Advanced button to start creating a new named data macro not attached to any table.

In Chapter 5, "Working with queries in web apps," you'll learn how to use the Query option in the drop-down list under the Advanced button in the ribbon. In Chapter 7, "Advanced view design," you'll learn how to work with the Blank View, List View, and Datasheet View options in this drop-down list. Finally, in Chapter 8, "Automating a web app using macros," you'll learn how to use the Macro and On Start Macro options under the Advanced button.

Access opens the Logic Designer with an empty macro design surface, as shown in Figure 4-41. You'll notice several differences on the macro design surface immediately that you did not see when creating data macros attached to table events in the preceding sections. When you're creating named data macros, the Logic Designer window is not modal. What this means is that you can see the Navigation pane and the App Home View, and you can interact with other objects without having to close the Logic Designer. Also, at the top of the macro design surface, you can see a section called Parameters. Named data macros allow you to create parameters, which you can use to pass information into the data macro. Creating parameters for named data macros is optional, but Access always displays the Parameters block at the top of the macro design surface whenever you are working with named data macros. The list of program flow constructs, data blocks, and data actions that you can use in named data macros is the same for table events except with the addition of one more data action called SetReturnVar. (We'll discuss the SetReturnVar action later in this chapter.) See Table 4-1 if you want to review the list of elements available in table events.

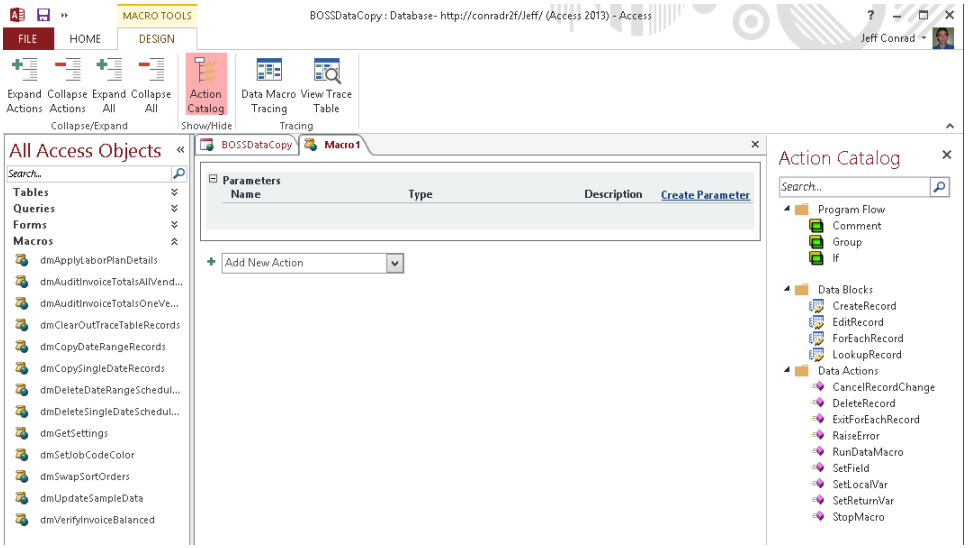


Figure 4-41 When you create named data macros, Access displays a Parameters block at the top of the macro design surface.

Let's first add a couple of Comment blocks to this named data macro to document its purpose. Drag a Comment block from the Action Catalog onto the macro design surface. Enter the following text into the new Comment block:

In this named data macro we want to make sure that only one job code is marked as the primary position for a specific employee. It is OK to not have any assigned primary positions for an employee but we do not want multiple primary positions defined.

Drag another Comment block onto the macro design surface below the first one, and enter the following text into this second Comment block:

This named data macro will run on the On Insert and On Update event for the tbl-TrainedPositions table. The employee and job code of the new or updated record will get passed in as parameters here. In the Where condition we will skip over the newly added or updated record and only touch the possible one other record that is marked as the primary position for the specific employee.

These two Comment blocks should give you an idea already of the type of logic we need to add to this named data macro as well as the reasoning behind the logic we will add.

Using parameters

In named data macros, you can define parameters to pass in information to the named data macro and use them in the data blocks and data actions. With parameters, you can pass in information to the named data macro from other data macros, views, and user interface macros. In the Back Office Software System sample web app, many of the named data macros include parameters. For the named data macro you are currently creating, we need to define two parameters—one for the employee we want to check and the second for the job code of the current record.

To create a new parameter in a named data macro, click the Create Parameter link on the right side of the macro design surface, as shown in Figure 4-42. You need to select the Parameters section to see the Create Parameter link. Access expands the Parameters section at the top of the macro design surface and inserts one new row for a parameter.

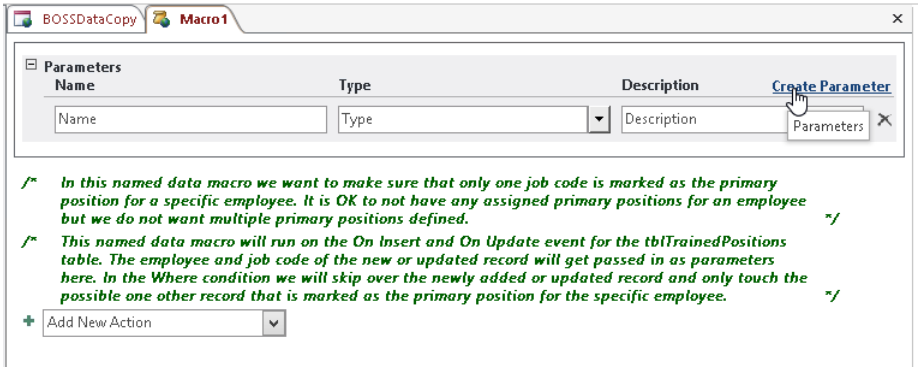


Figure 4-42 Click the Create Parameter link to create new parameters in named data macros.

Each parameter takes three arguments:

- **Name.** Required argument. The name of the parameter you want to use to refer to during named data macro execution.
- **Type.** Required argument. The data type that Access uses to define the parameter.
- **Description.** Optional argument. A description for you to document the purpose of the parameter.

For the Name argument, you can enter a name up to 64 characters. The restrictions for naming parameters are the same as for local variables, which you learned about earlier in this chapter. In this example, enter **ParamEmployeeID** into the Name argument, which we'll use to denote the ID of the employee to search for in the named data macro. For the Type argument, you can choose from one of ten data type options—Short Text, Long

Text, Number (Floating Decimal), Number (No Decimal), Number (Fixed Decimal), Date With Time, Date, Time, Currency, or Yes/No. In this example, select Number (No Decimal) from the drop-down list of data type options. The employee ID values that we will be passing into this named data macro should not have any decimal places, because they are ID values, so the Number (No Decimal) data type should suffice for this named data macro parameter. For the Description argument, enter **Employee ID record to look for** into the text box to describe the purpose of this parameter value. Your completed changes for the first parameter should now match Figure 4-43.

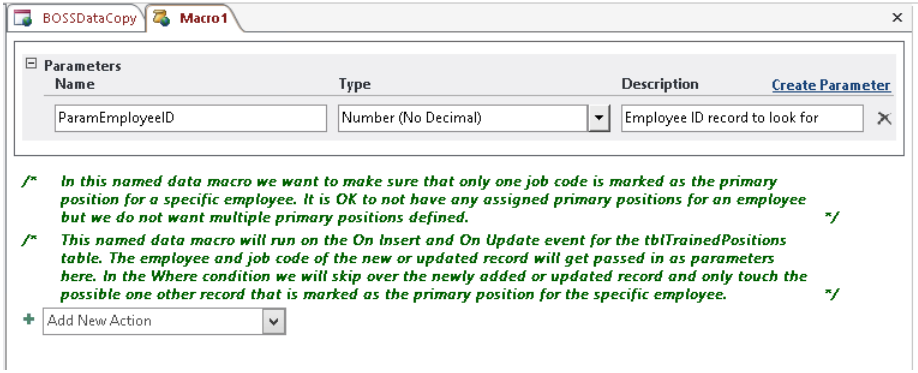


Figure 4-43 Enter the parameter information into the three arguments.

We need to define one additional parameter for this named data macro to track the job code ID of the record just created (the On Insert case) or the record just updated (the On Update case). To define another parameter, click the Create Parameter link again on the right side of the macro design surface in the Parameters section. Access inserts a new parameter row beneath the existing one. For this second parameter, enter **ParamJobCodeID** in the Name text box, select Number (No Decimal) from the drop-down list in the Type argument, and enter **Job Code ID to ignore** in the Description text box. Your completed two parameters should match Figure 4-44.

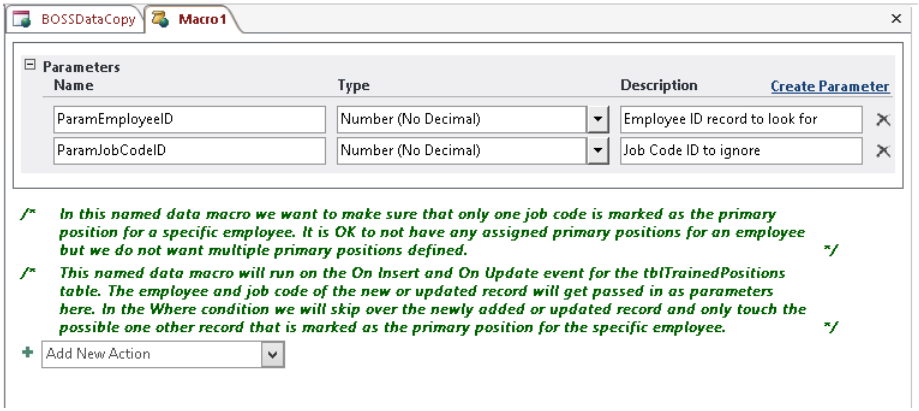


Figure 4-44 You should have two completed parameters defined in the new named data macro.

> **Note**
>
> **If you need to delete an existing parameter, click the delete button to the far right side of the specific Parameter row. The delete button has a symbol shaped like an X.**

Now that you've defined the two parameters we need, it's time to add the actions necessary to perform our task. In this named data macro, we want to loop through records in the tblTrainedPositions table looking for specific records. You've previously seen how the LookupRecord data block searches for a specific record in a table or saved query. In this case, we need to use the ForEachRecord data block to search through more than one record potentially. Drag a ForEachRecord data block from the Action Catalog to beneath the two Comment blocks, or select ForEachRecord from the Add New Action box at the bottom of the macro design surface. Access creates a new ForEachRecord block, as shown in Figure 4-45.

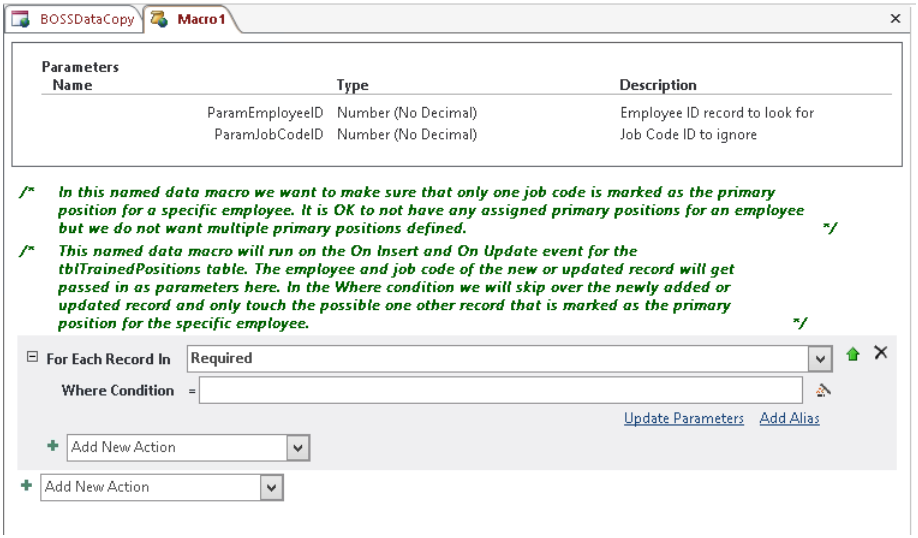


Figure 4-45 Drag a ForEachRecord data block onto the macro design surface.

The ForEachRecord data block takes four arguments:

- **For Each Record In.** Required argument. The name of a table or query to look up a record in.
- **Where Condition.** Optional argument. The expression that Access uses to select records from the table or query.
- **Update Parameters.** Optional argument. If you're looking up records in a query that requires parameters, you can provide them here.
- **Alias.** Optional argument. A substitute or shorter name for the table or query.

The only required argument for the ForEachRecord data block is For Each Record In. Access provides a drop-down list for this argument that includes the names of all tables and saved query objects in your web app. If you want Access to find a subset of specific records in the specified table or query, you must provide a valid Where clause expression to find the records. If you leave the Where Condition argument blank, Access loops through all records in the specified table or query. You can click the button with the magic wand on it to open the Expression Builder to assist you with creating a Where clause if you'd like.

The Update Parameters and Alias optional arguments are accessible through two links below the Where Condition argument on the right side. When you click these links, Access displays additional text boxes for you to enter these arguments. If you are running a ForEachRecord data block against a table, clicking the Update Parameters link does

nothing, because tables do not contain parameters. If you are using a query for your data source that includes parameters, you can update the parameters using this link.

The tblTrainedPositions table contains one record for each job code that a specific employee is trained to perform. Each employee could have multiple records in this table. In an extreme case, one employee could be trained in every position in the restaurant, so that person could have one record in the tblTrainedPositions table for each job code in the app. The PrimaryPosition field in this table is a Yes/No data type that denotes whether the specific job code is the employee's primary position. In this scenario, we need to use the ForEachRecord data block instead of the LookupRecord data block to search over each record for a specific employee, so click inside the For Each Record In argument and select tblTrainedPositions from the drop-down list.

To make sure we are searching for all correct matches in the tblTrainedPositions table, we need to utilize the values passed in from the parameters in the Where condition argument. The final expression I used to accomplish this task, which you'll build in a moment, is as follows:

```
[tblTrainedPositions].[EmployeeIDFK]=[ParamEmployeeID] And [tblTrainedPositions].
[JobCodeIDFK]<>[ParamJobCodeID] And [tblTrainedPositions].[PrimaryPosition]=Yes
```

This expression contains three distinct clauses all joined together with AND operators. In the first part of the expression, we are trying to find all records where the EmployeeIDFK field in tblTrainedPositions matches the parameter ParamEmployeeID that we will pass in to this named data macro. Enter the first part of this expression into the Where condition argument. When you start typing the parameter name, IntelliSense helps you along and displays all parameter names so that you can easily see and select the parameter name that holds the employee ID value, as shown in Figure 4-46.

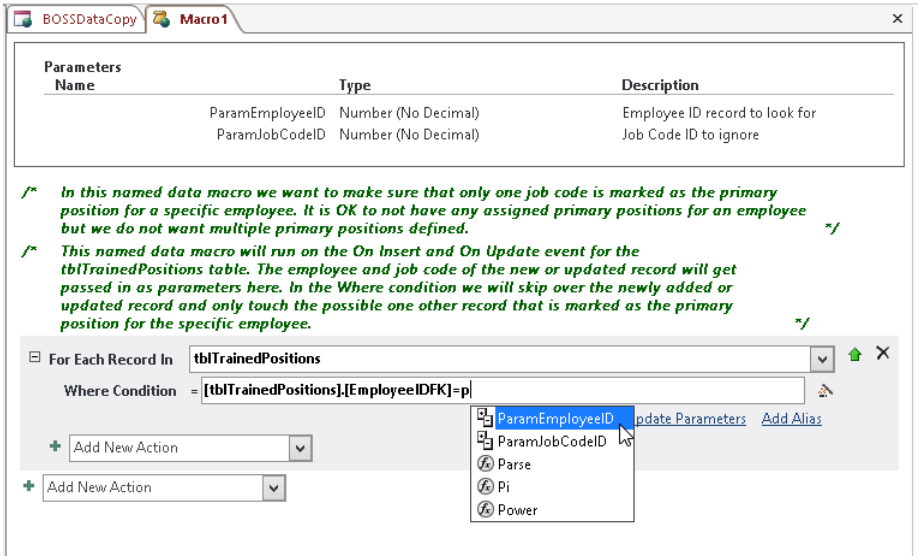


Figure 4-46 IntelliSense provides parameter names when you are building expressions in named data macros.

After you type the first part of the expression, add a space, type **And**, and then enter the second part of the expression:

```
[tblTrainedPositions].[JobCodeIDFK]<>[ParamJobCodeID]
```

In the second part of this expression, we are instructing Access to exclude records where the JobCodeIDFK field matches the parameter ParamJobCodeID that we will pass in to this named data macro. You might be wondering why we want to do this. As you'll learn in the next few sections, whenever we create new records in this table or update existing ones, we will pass in the job code of the record just created or the record just updated but only if that record is designated to be the primary position. Because this new or revised record will now be the primary position, there is no need to inspect this current record during the ForEachRecord loop.

After you type the second part of the expression, add a space, type another **And**, then enter the last part of the expression:

```
[tblTrainedPositions].[PrimaryPosition]=Yes
```

In the last part of this expression, we are instructing Access to include only records where the PrimaryPosition Yes/No field equals Yes. During the ForEachRecord loop, Access could find several records for the employee we are looking for. We really need to identify only records where the PrimaryPosition field is already Yes, so we can then mark those records as No in the PrimaryPosition field because a user just created a new primary position record

or updated an existing record. This will all make sense when we complete all the tasks later in this section.

Now that you have the correct expression in place for the Where condition argument, we need to add one last step in this named data macro to update the PrimaryPosition field to No for any records Access finds during the ForEachRecord loop. To update the field, you need to use the SetField data action inside an EditRecord data block. Click inside the Add New Action combo box inside the ForEachRecord data block, type **EditRecord**, and then press Enter. Access adds a new EditRecord data block onto the macro design surface inside the ForEachRecord block. Next, click inside the Add New Action combo box inside the EditRecord data block, type **SetField**, and then press Enter to add this new action to the macro design surface. Finally, in the Name argument for the SetField action, enter **[tblTrainedPositions].[PrimaryPosition]** and No into the Value argument. Your completed changes to the named data macro should now match Figure 4-47.

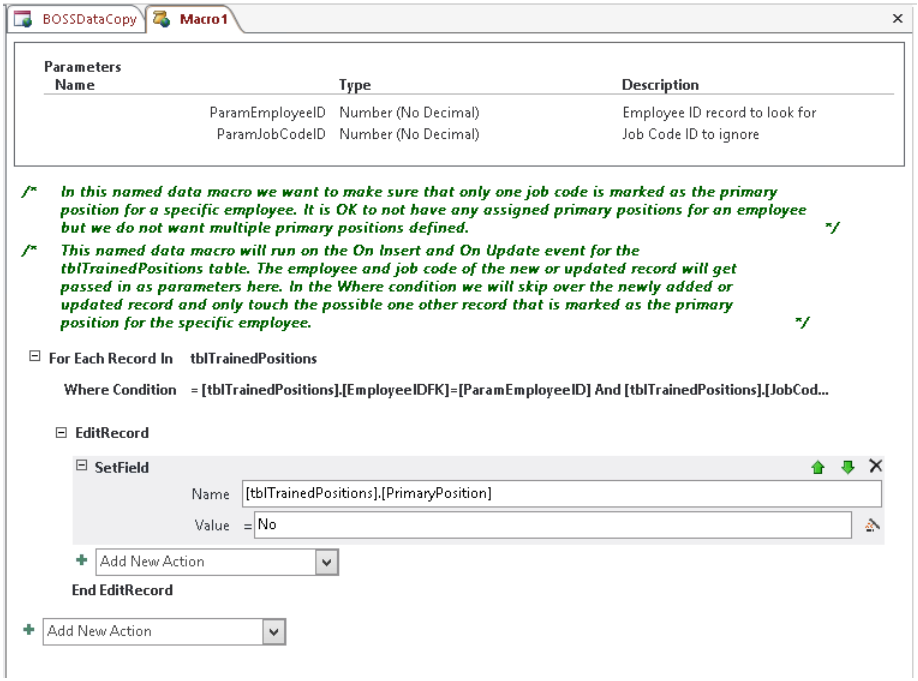


Figure 4-47 Your named data macro to maintain only one primary trained position should now look like this.

> **Note**
>
> **You might be wondering why I used a ForEachRecord data block in the named data macro, given that the expression in the Where condition argument should return only one record. You're correct that Access should find only one record based on the logic I've put in place. I'm being extra careful to make sure that only one job position is marked as the primary position by using a ForEachRecord data block to cover the off chance that two records for a specific employee are marked as primary positions.**

Saving named data macros

You've completed creating your first named data macro, but now you need to save it and give it a name. Unlike data macros attached to table events, named data macros require you to provide a unique name. To save your new named data macro, click the Save button on the Quick Access Toolbar. Access opens the Save As dialog box, as shown in Figure 4-48. Save the named data macro with the name **dmEnforceOnlyOnePrimaryPosition**.

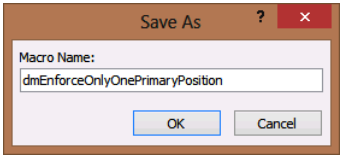


Figure 4-48 Provide a unique name for your new named data macro in the Save As dialog box.

If you attempt to save a named data macro with the same name as an existing named data macro in the Navigation pane, Access displays an error message, as shown in Figure 4-49.

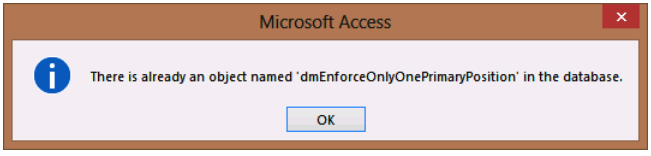


Figure 4-49 Access displays an error message if you try to save a named data macro with the same name as an existing named data macro.

Calling named data macros

I mentioned earlier that named data macros must be called for Access to execute them. If you want to test out a named data macro, you must therefore call a RunDataMacro action from a table event or from a user interface macro. In Chapter 8, you'll learn how to call named data macros from user interface macros and in Chapter 25, "Automating your

desktop database with Visual Basic," which can be downloaded from the book's catalog page, you'll learn how to call named data macros from Visual Basic.

Close the Logic Designer, if you still have it open, and then open the tblTrainedPositions table in Design view. We need to call the named data macro in both the On Insert and On Update events, so let's begin with the On Insert event. Click the On Insert button in the Events group on the Design contextual tab to open the Logic Designer. Start by adding a new Comment block to the macro design surface, and enter the following text into the Comment block:

After we commit this new record we need to make sure we do not have more than one primary position designated for the same employee. Run the named data macro if this new record is marked as primary to clear out any other possibilities.

When you enter a new record in the tblTrainedPositions, we don't need to run the named data macro if you set the PrimaryPosition field to No. Remember, we want to enforce only one primary position so that if the new record is not set as a primary position, we don't need to do any extra work. To account for this possibility, add an If block beneath the Comment block onto the macro design surface. In the conditional expression text box, enter **[tblTrainedPositions].[PrimaryPosition]=Yes**. Access does not run the next action we add inside the If block if the new record has the PrimaryPosition field set to No.

To call the named data macro to run, you need to use the RunDataMacro action. Click in the Add New Action combo box inside the If block, type **RunDataMacro**, and then press Enter. Access displays the RunDataMacro on the macro design surface, as shown in Figure 4-50.

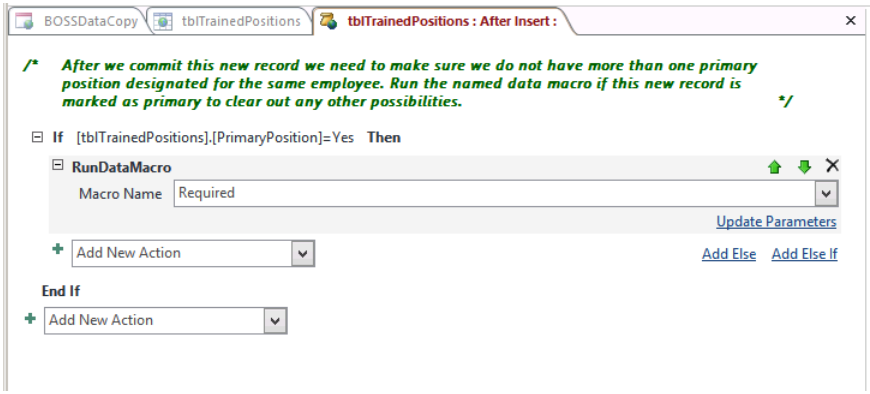


Figure 4-50 Add a RunDataMacro action inside the If block.

The only required argument for the RunDataMacro data action is Macro Name. Access provides a drop-down list for this argument that includes the names of all saved named

data macros in your web app. Click in the Macro Name box, and select the named data macro you created earlier from the drop-down list—dmEnforceOnlyOnePrimaryPosition. After you select the named data macro, Access displays the parameters you defined earlier in the named data macro. Access displays the two parameters in the underlying named data macro—ParamEmployeeID and ParamJobCodeID—as parameter boxes at the bottom of the action, as shown in Figure 4-51. You can enter a value you want to use for each parameter by typing the value into the parameter box or using an expression to derive that parameter value.

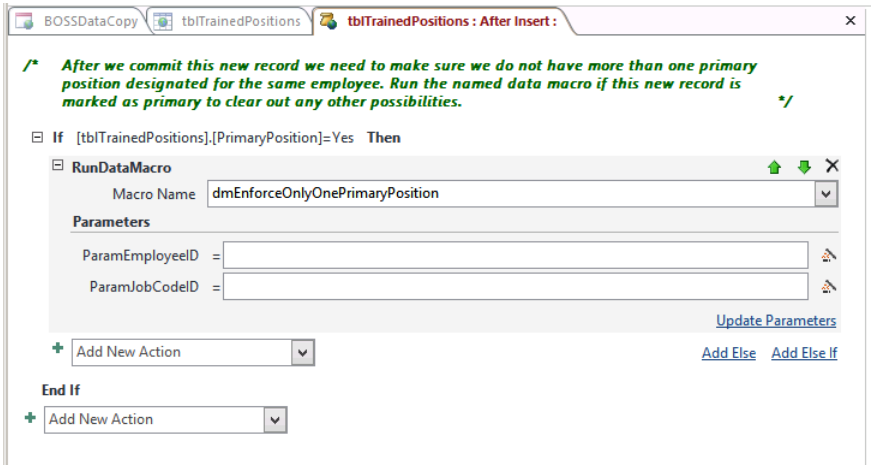


Figure 4-51 Access displays Parameter boxes on the macro design surface for any named data macros that require parameters.

The two parameters we need to pass into the named data macro come directly from the record Access just inserted. In the ParamEmployeeID parameter text box, enter **[tblTrainedPositions].[EmployeeIDFK]**, and in the ParamJobCodeID parameter text box, enter **[tblTrainedPositions].[JobCodeIDFK]**, as shown in Figure 4-52. When you create a new record in this table and set the PrimaryPosition field to Yes, Access takes the data stored in the EmployeeIDFK and JobCodeIDFK fields and passes those values into the named data macro you created earlier. Click Save in the Close group on the Design contextual tab, or click the Save button on the Quick Access Toolbar to save your changes to this On Insert table event but leave the Logic Designer window open.

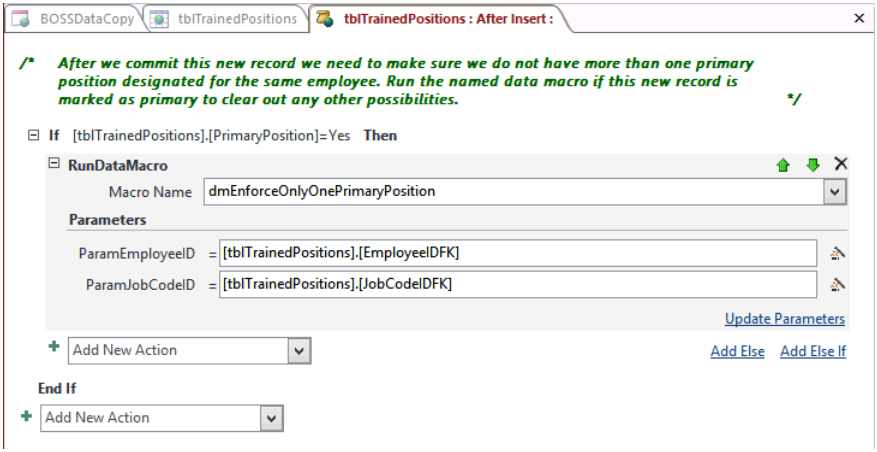


Figure 4-52 Enter field names into the parameter boxes in the RunDataMacro action.

We also need to add the same data macro logic to the On Update event of the tblTrainedPositions as well account for users of the app changing existing records. You should be very familiar now with adding data blacks, data actions, and filling in parameters in data macros manually, but this time we'll use a different technique. Because the logic currently showing in the On Insert event is the same as what we want to add to the On Update event, we can simply copy the data macro logic to the Windows Clipboard and then paste the contents into the On Update event. To do this, click inside the Logic Designer on the macro design surface, away from any commands, and then press Ctrl+A to highlight all of the logic currently showing in the On Insert table event, as shown in Figure 4-53.

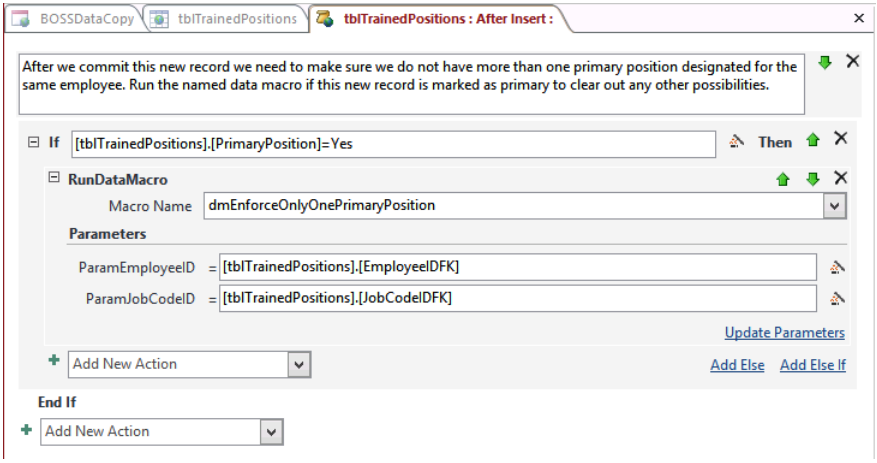


Figure 4-53 Press Ctrl+A to highlight all the data macro logic on the macro design surface.

Now that you have all the data macro logic highlighted, press Ctrl+C to copy all the Comment blocks, data blocks, and data actions to the Windows Clipboard. Next, click Close in the Close group on the Design contextual tab to close the On Insert table event. You should see the tblTrainedPositions table still open in Design view. Click the On Update button in the Events group on the Design contextual tab to open the Logic Designer window for this table event. Finally, click anywhere on the macro design surface and then press Ctrl+V. Access pastes all the data macro from the Windows Clipboard onto the macro design surface, as shown in Figure 4-54. As you can see, copying and pasting the data macro logic from the On Insert event to the On Update event using this technique is much faster than adding all of the actions manually one by one.

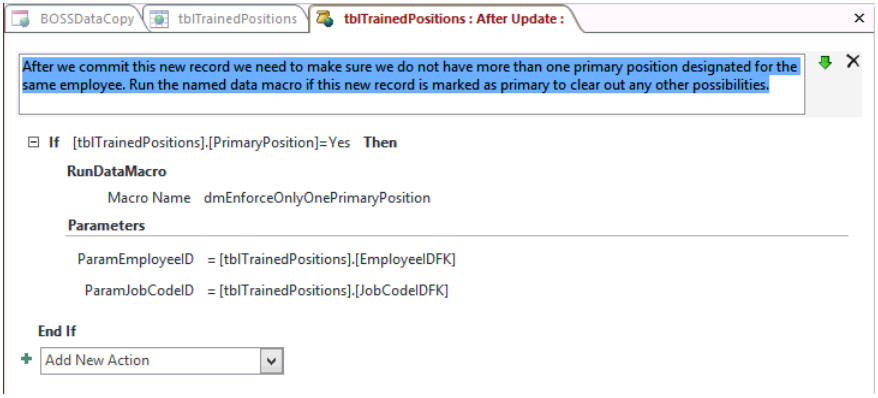
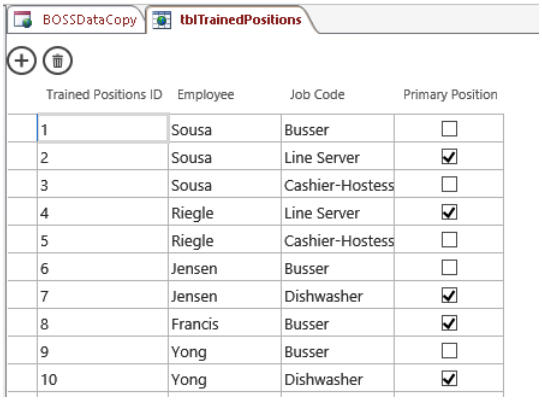


Figure 4-54 Press Ctrl+V to paste all the data macro logic from the Windows Clipboard into the On Update event of the tblTrainedPositions.

To test out the named data macro, save the changes to this On Update event and then close the Logic Designer. Switch to Datasheet view for the tblTrainedPositions table by clicking the View button in the Views group on the Design contextual tab, and then click Datasheet view on the drop-down menu. The first three records in this table display the trained positions for the employee with the last name of Sousa, as shown in Figure 4-55.



Trained Positions ID	Employee	Job Code	Primary Position
1	Sousa	Busser	☐
2	Sousa	Line Server	☑
3	Sousa	Cashier-Hostess	☐
4	Riegle	Line Server	☑
5	Riegle	Cashier-Hostess	☐
6	Jensen	Busser	☐
7	Jensen	Dishwasher	☑
8	Francis	Busser	☑
9	Yong	Busser	☐
10	Yong	Dishwasher	☑

Figure 4-55 In Datasheet view, you can see each trained position for the employees in the web app.

In Figure 4-55, you can see that the employee named Sousa is trained to be a Busser, a Line Server, and a Cashier-Hostess, with their primary position being the Line Server position. Change this employee's primary position to Busser by clicking into the first record and selecting the Primary Position check box, and then tab or click into a different record to commit the record update. Initially, you won't see any changes in any other records because Access caches the data locally. To see the most recent updates to other records, click the Refresh button in the Records group on the Datasheet contextual tab. (If you were using a view within your web browser, you would see the changes refreshed in the records.)

You'll now notice Access changed the second record after you updated the first record, as shown in Figure 4-56. Access automatically ran the named data macro after you changed the PrimaryPosition field to Yes in the first record. Access passed in the employee ID for Sousa, passed in the job code ID for the Line Server position, and then updated the second record by changing the PrimaryPosition in that record to No to maintain our goal of only one primary position for each employee. Access also runs the same named data macro whenever you add new records to this table and set the PrimaryPosition to Yes.

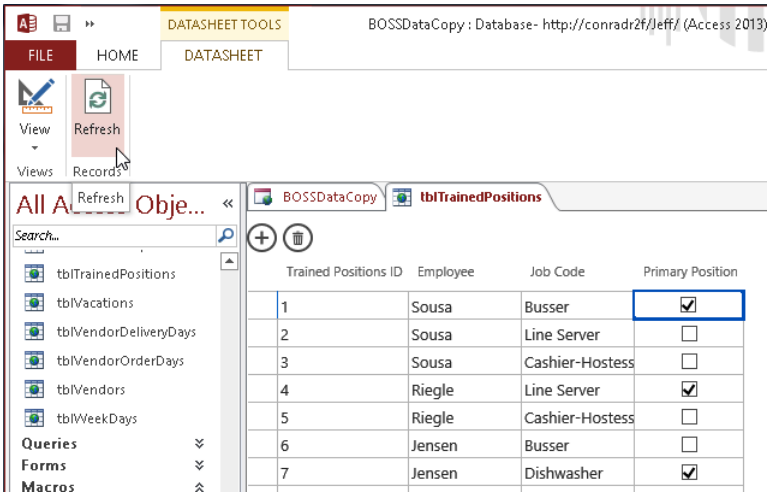


Figure 4-56 Access runs the named data macro after you update any record in the tblTrainedPositions table.

Renaming and deleting named data macros

When you need to rename or delete named data macros, you must do so from the Navigation pane. If you want to rename a named data macro, right-click the named data macro in the Navigation pane and select Rename from the shortcut menu, as shown in Figure 4-57.

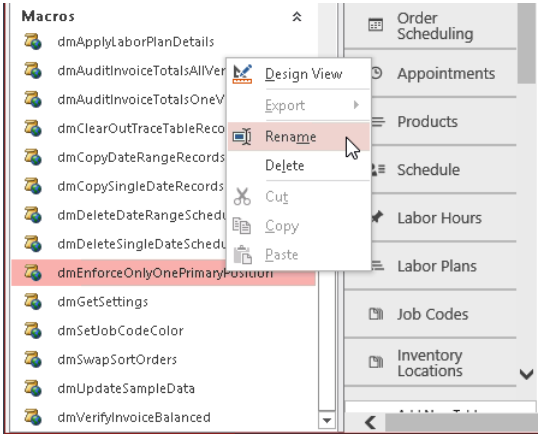


Figure 4-57 Click Rename on the shortcut menu to rename named data macros.

Access highlights the name of the named data macro in the Navigation pane and allows you to enter a new name for the named data macro, as shown in Figure 4-58. You must enter a unique name for your named data macro. If you enter the name of an existing

named data macro, Access displays a warning message indicating that there is already an object in the web app with the same name.

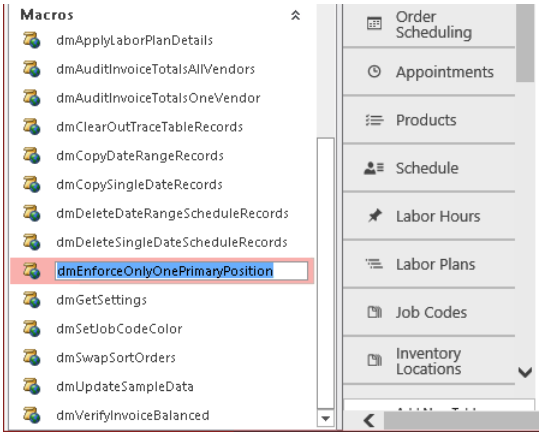


Figure 4-58 Enter a new name in the Navigation pane for the named data macro.

If you want to delete a named data macro, right-click the named data macro in the Navigation pane and select Delete from the shortcut menu. Access opens a confirmation message box, as shown in Figure 4-59. Click Yes if you want to permanently delete the named data macro.

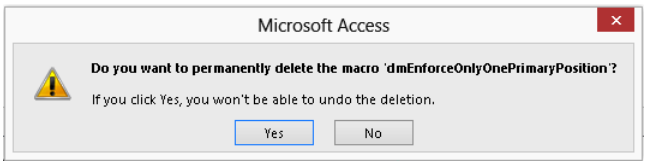


Figure 4-59 In the confirmation message, click Yes to delete the named data macro.

Note

You cannot rename table event data macros because they are attached directly to the table event.

CAUTION!

If you rename a named data macro or delete a named data macro, you must adjust any other areas of your web app that reference that named data macro; otherwise, you might encounter errors using areas of your web app that reference that named data macro. For example, if you rename or delete the dmEnforceOnlyOnePrimaryPosition named data macro you created earlier, Access displays an error whenever you add or edit existing records to the tblTrainedPositions, because Access cannot find the named data macro. You won't be able to add or edit any data in that table until you remove the reference to the named data macro in both the On Insert and On Update table events for tblTrainedPositions.

Working with return variables

You can use a return variable in data macros to return data to the object that called the named data macro. In a sense, you can think of a return variable as the opposite of a parameter. You use parameters to push data into a named data macro, and you use return variables to pull data out of named data macros. Return variables are very useful when you need Access to read values from a table or query during the execution of the named data macro and perhaps perform different steps based on that value. Return variables can even be returned from the data layer up to the user interface level. All return variables have a unique name. To fetch, set, or examine a return variable, you reference it by its name. Return variables stay in memory until the data macro finishes executing, you assign it a new value, or until you clear the value. You can set return variables only in named data macros; however, you can retrieve them from table events, other named data macros, or user interface macros.

Let's examine a named data macro that uses return variables so that you can understand how this works. Open the dmGetSettings named data macro in Design view from the Navigation pane. Access opens the Logic Designer and displays the logic that I created for this named data macro, as shown in Figure 4-60.

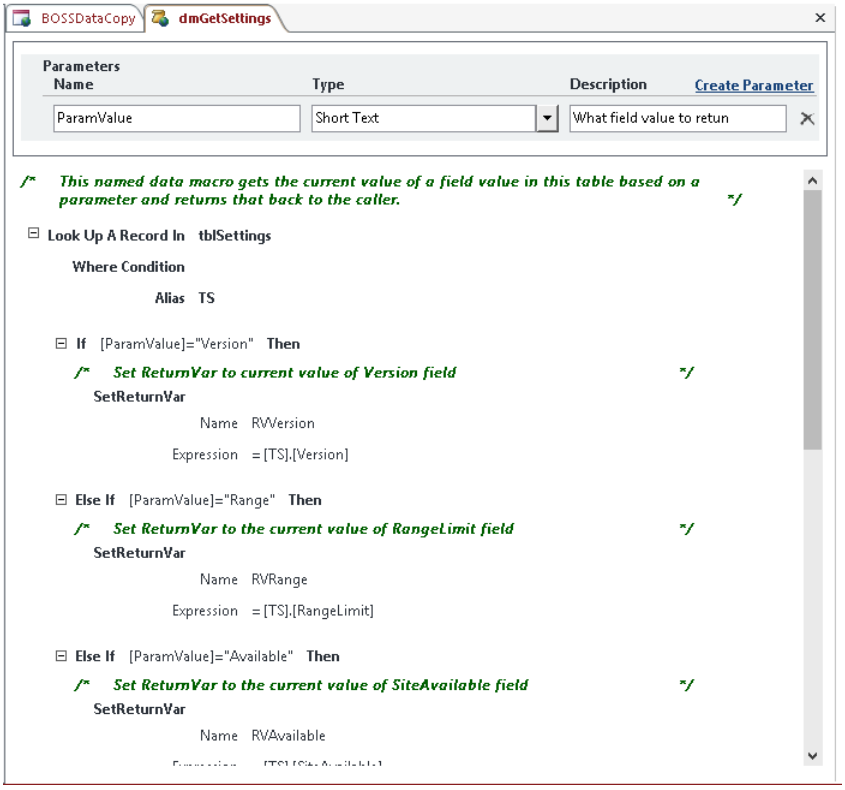


Figure 4-60 The dmGetSettings named data macro uses return variables to return data to the caller.

The logic for the dmGetSettings named data macro is as follows:

```
Parameter Name: ParamValue
Parameter Type: Short Text
Parameter Description: What field value to return
Comment Block: This named data macro gets the current value of a field value in this
table based on a parameter and returns that back to the caller.
LookupRecord In tblSettings
  Where Condition
  Alias: TS
  If [ParamValue]="Version" Then
     Comment Block: Set ReturnVar to current value of Version field
     SetReturnVar
       Name: RVVersion
       Expression: [TS].[Version]
  Else If [ParamValue]="Range" Then
     Comment Block: Set ReturnVar to the current value of RangeLimit field
     SetReturnVar
       Name: RVRange
```

```
      Expression: [TS].[RangeLimit]
  Else If [ParamValue]="Available" Then
     Comment Block: Set ReturnVar to the current value of SiteAvailable field
     SetReturnVar
       Name: RVAvailable
       Expression: [TS].[SiteAvailable]
  Else If [ParamValue]="SendEmailOnError" Then
     Comment Block: Set ReturnVar to the current value of the
                    SendEmailForAppErrors field
     SetReturnVar
       Name: RVSendEmailOnError
       Expression: [TS].[SendEmailForAppErrors]
  Else If [ParamValue]="AdminEmail" Then
     Comment Block: Set ReturnVar to the current value of the AdminEmailAddress field
     SetReturnVar
       Name: RVAdminEmailAddress
       Expression: [TS].[AdminEmailAddress]
  Else If [ParamValue]="AllEmailInfoForErrors" Then
     Comment Block: For this parameter value, send back the settings for both the
                    SendEmailOnError and AdminEmailAddress fields so the
                    caller doesn't need to make two trips.
     SetReturnVar
       Name: RVSendEmailForError
       Expression: [TS].[SendEmailForAppErrors]
     SetReturnVar
       Name: RVAdminEmailForErrors
       Expression: [TS].[AdminEmailAddress]
  End If
```


The tblSettings table holds application-specific settings in several fields. By storing these settings in the table, we can then use data macros to retrieve these values at any time. The dmGetSettings named data macro uses a large If block inside a LookupRecord data block. The If/Else If conditions check the value of the parameter ParamValue being passed in from the caller. We then use the SetReturnVar data action to define a new return variable. The SetReturnVar action takes two arguments:

- **Name.** Required argument. The name of the return variable.
- **Expression.** Required argument. The expression that Access uses to define the return variable.

I set a unique name for each return variable inside the various Else If condition blocks. For the Expression argument of each SetReturnVar action, I use an alias of the table name and read the data from a specific field. In the last Else If condition block, I return data from two fields with two different return variables to save the caller from having to make two RunDataMacro calls for related application settings. I could optionally create a named data macro that returns all data from the fields with return variables in one call, but I didn't want to be passing around data when it would not be needed. By itself, this named data macro

does not do anything more than read values from the tblSettings table. However, the real power of the return variables is the ability of the object calling this named data macro to use these values.

To see how this data in return variables can be used, close the Logic Designer for this named data macro. Now open in Design view the dmAuditInvoiceTotalsOneVendor named data macro. Access opens the Logic Designer and displays the logic that I created for this named data macro, as shown in Figure 4-61. This named data macro audits all invoice records for a specific vendor within a given date range. The named data macro starts by running a different named data macro to retrieve a date range number from a system table. The named data macro then loops through each invoice detail record for each invoice within the desired date range, adds up the total amount of the line item details, and compares it to the invoice total. If the line item details match the invoice total, Access marks the invoice balanced. If the line item details do not match the invoice total, Access marks the invoice as unbalanced. Finally, Access returns the total number of unbalanced invoices, if any, to the calling macro.

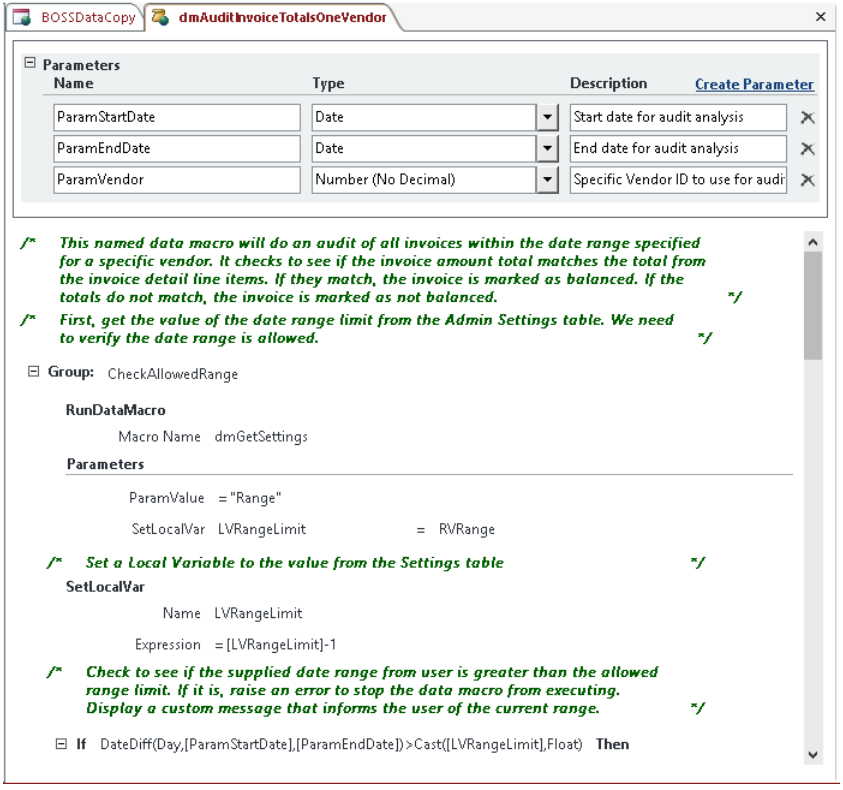


Figure 4-61 Open the dmAuditInvoiceTotalsOneVendor named data macro.

This named data macro is quite lengthy, so I'll break up our discussion of the logic behind this named data macro into several parts. The logic for the first part of the named data macro is as follows:

```
Parameter Name: ParamStartDate
Parameter Type: Date
Parameter Description: Start date for audit analysis
Parameter Name: ParamEndDate
Parameter Type: Date
Parameter Description: End date for audit analysis
Parameter Name: ParamVendor
Parameter Type: Number (No Decimal)
Parameter Description: Specific Vendor ID to use for audit analysis
Comment Block: This named data macro will do an audit of all invoices within the date
range specified for a specific vendor. It checks to see if the invoice amount total
matches the total from the invoice detail line items. If they match, the invoice is
marked as balanced. If the totals do not match, the invoice is marked as not
balanced.
Comment Block: First, get the value of the date range limit from the Admin Settings
table. We need to verify the date range is allowed.
Group: CheckAllowedRange
  RunDataMacro:
    Macro Name: dmGetSettings
    Parameters:
       ParamValue: "Range"
       SetLocalVar: LVRangeLimit = RVRange
  Comment Block: Set a Local Variable to the value from the Settings table
  SetLocalVar
    Name: LVRangeLimit
    Expression: [LVRangeLimit]-1
  Comment Block: Check to see if the supplied date range from user is greater than
  the allowed range limit. If it is, raise an error to stop the data macro from
  executing. Display a custom message that informs the user of the current range.
  If DateDiff(Day,[ParamStartDate],[ParamEndDate])>Cast([LVRangeLimit],Float)=True
    Raise Error:
      Error Description: =Concat("You have attempted to run an invoice audit with a
      date range larger than the allotted number of days. Please restrict your date
      range to ",(Cast([LVRangeLimit],Float)+1)," days.")
  End If
End Group
```

The dmAuditInvoiceTotalsOneVendor named data macro includes three parameters. I pass in all three of these values from a user interface macro to know what date range I want to audit invoice records and the specific vendor records to audit. Inside the Group block, I use the RunDataMacro action. For the Macro Name argument of the RunDataMacro action, I use the dmGetSettings named data macro, which you saw in the previous section.

You'll notice in Figure 4-61 that Access displays a Parameters section beneath the Macro Name argument. When you add a named data macro that includes parameters to the macro design surface, Access shows those parameters to you by providing a text box to

enter the parameters. In our example, I pass in the Range parameter to get the value of the RangeLimit text field from the tblSettings table. Beneath the parameter value on the macro design surface, Access displays a SetLocalVar action for each return variable in the dmGetSettings named data macro. When Access returns the variable, or potential variables as the case might be, back to the calling macro, you can assign a local variable to each of the return variables and use them during the execution of the named data macro. In our example, because I'm getting one return variable back, you see only one SetLocalVar action displayed on the macro design surface. After you save and close the named data macro, Access displays only SetLocalVar actions inside the Parameters block for variables you set to handle the return variables. If you click the Update Parameters link, Access displays a SetLocalVar action for each return variable. For our example, I set a local variable called LVRangeLimit, which holds the RVRange return variable received from the dmGetSettings named data macro.

After the RunDataMacro action completes and returns back the needed data through the return variable, Access subtracts one number from the local variable previously set by the return variable. In the If condition that follows, I define an expression to calculate the difference in days from the start date and end date parameters. In the second part of the If condition, I check to see whether that value exceeds the date range limit previously defined using the *Cast* function. If the date range exceeds the limit, I use a RaiseError data action to inform the user that the date range is too large and stop the named data macro from executing any further. The message I display to the user in the RaiseError action uses the *Concat* function to display a custom text message that includes the number of days they are allowed to use for the date range.

In Figure 4-62, you can see the second section of the dmAuditInvoiceTotalsOneVendor named data macro. In Figure 4-62, I collapsed the Parameters block so that you can see more of the logic.

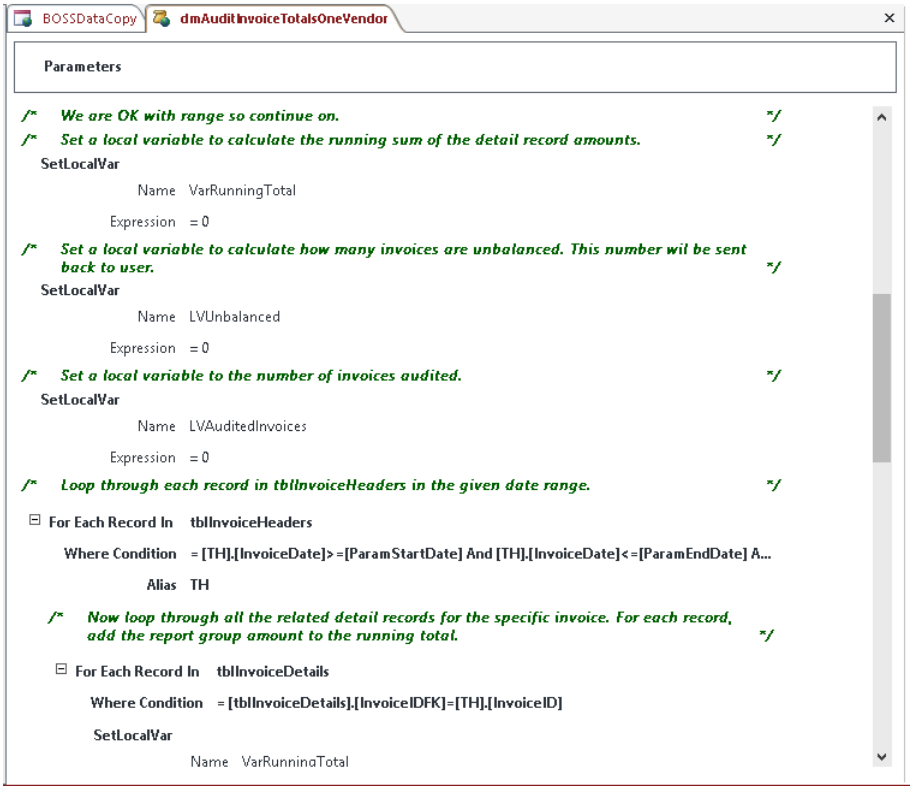


Figure 4-62 This is the second part of the dmAuditInvoiceTotalsOneVendor named data macro.

The logic for the second section of the dmAuditInvoiceTotalsOneVendor named data macro is as follows:

```
Comment Block: We are OK with range so continue on.
Comment Block: Set a local variable to calculate the running sum of the detail record
amounts.
SetLocalVar
  Name: VarRunningTotal
  Expression: 0
Comment Block: Set a local variable to calculate how many invoices are unbalanced.
This number will be sent back to user.
SetLocalVar
  Name: LVUnbalanced
  Expression: 0
Comment Block: Set a local variable to the number of invoices audited.
SetLocalVar
  Name: LVAuditedInvoices
  Expression: 0
Comment Block: Loop through each record in tblInvoiceHeaders in the given date range.
ForEachRecord In tblInvoiceHeaders
```

```
Where Condition = [TH].[InvoiceDate]>=[ParamStartDate] And
            [TH].[InvoiceDate]<=[ParamEndDate] And [TH].[VendorIDFK]=[ParamVendor]
Alias: TH
Comment Block: Now loop through all the related detail records for the specific
  invoice. For each record, add the report group amount to the running total.
ForEachRecord In tblInvoiceDetails
  Where Condition = [tblInvoiceDetails].[InvoiceIDFK]=[TH].[InvoiceID]
  SetLocalVar
   Name: VarRunningTotal
   Expression: [VarRunningTotal]+[tblInvoiceDetails].[ReportGroupAmount]
```

In this section of the named data macro, I define three local variables—VarRunningTotal, LVUnbalanced, and LVAuditedInvoices. The VarRunningTotal local variable tracks the running sum of the total invoice details line items for each specific invoice. The LVUnbalanced local variable tracks how many unbalanced invoices Access finds during the course of the named data macro execution. The LVAuditedInvoices local variable tracks how many invoices Access audits within the given parameters of data macro execution.

The named data macro then executes a ForEachRecord data block to loop through all records in the tblInvoiceHeaders table. For this ForEachRecord data block, I use TH as an alias to represent the name of the tblInvoiceHeaders table for brevity in subsequent areas of the named data macro. The expression I use in the Where condition argument for the ForEachRecord data block is as follows:

```
[TH].[InvoiceDate]>=[ParamStartDate] And [TH].[InvoiceDate]<=[ParamEndDate] And [TH].
[VendorIDFK]=[ParamVendor]
```

The Where condition restricts Access to look for invoice records between the start date and end dates passed in from the parameters. Access further restricts the records to loop through by looking for the specific vendor ID also passed in as a parameter.

Inside the ForEachRecord data block for tblInvoiceHeaders, I use another ForEachRecord data block to then loop through the invoice details records in the tblInvoiceDetails table for each invoice that Access finds in the first ForEachRecord data block. The expression I use in the Where condition argument for this second ForEachRecord data block is as follows:

```
[tblInvoiceDetails].[InvoiceIDFK]=[TH].[InvoiceID]
```

Inside the second ForEachRecord data block, I set a local variable called VarRunningTotal, previously set to zero, to increment itself by the line item total found in the ReportGroupAmount field. On each pass through tblInvoiceDetails table for each specific invoice, Access then keeps a running total of the amount spent on each invoice in this child table.

In Figure 4-63, you can see the third section of the dmAuditInvoiceTotalsOneVendor named data macro.

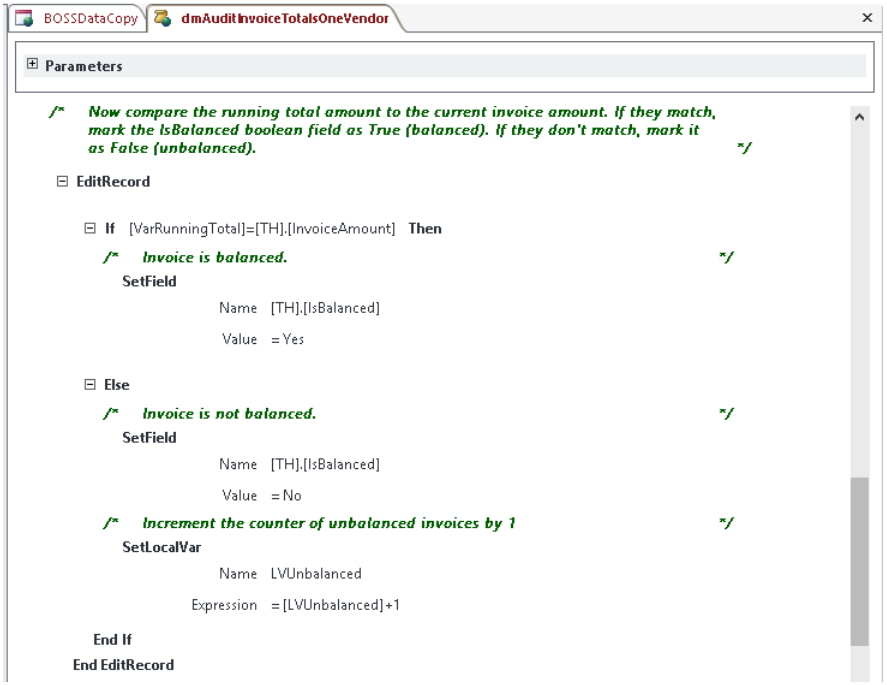


Figure 4-63 This is the third part of the dmAuditInvoiceTotalsOneVendor named data macro.

The logic for the third section of the dmAuditInvoiceTotalsOneVendor named data macro is as follows:

```
Comment Block: Now compare the running total amount to the current invoice amount. If
they match, mark the IsBalanced boolean field as Yes (balanced). If they don't match,
mark it as No (unbalanced).
  EditRecord
    If [VarRunningTotal]=[TH].[InvoiceAmount] Then
       Comment Block: Invoice is balanced.
       SetField
         Name: [TH].[IsBalanced]
         Value: Yes
    Else
       Comment Block: Invoice is not balanced.
       SetField
         Name: [TH].[IsBalanced]
         Value: No
      Comment Block: Increment the counter of unbalanced invoices by 1
      SetLocalVar
        Name: LVUnbalanced
        Expression: =[LVUnbalanced]+1
    End If
  End EditRecord
```

After Access finishes calculating the total from all the invoice details for one invoice, I then have an EditRecord block inside the first ForEachRecord data block. Inside the EditRecord block, I use an If block to test whether the running invoice total, tracked by the VarRunningTotal local variable, equals the amount stored in the InvoiceTotal field in the tblInvoiceHeaders table. If the two amounts match, I use a SetField data action to instruct Access to update the IsBalanced field in tblInvoiceHeaders to Yes. If the two amounts do not match, Access goes into the Else block and then uses SetField to change the IsBalanced field to No. Inside the Else block, I also increment the LVUnbalanced local variable, which is tracking the number of unbalanced invoices, by one.

In Figure 4-64, you can see the last section of the dmAuditInvoiceTotalsOneVendor named data macro.

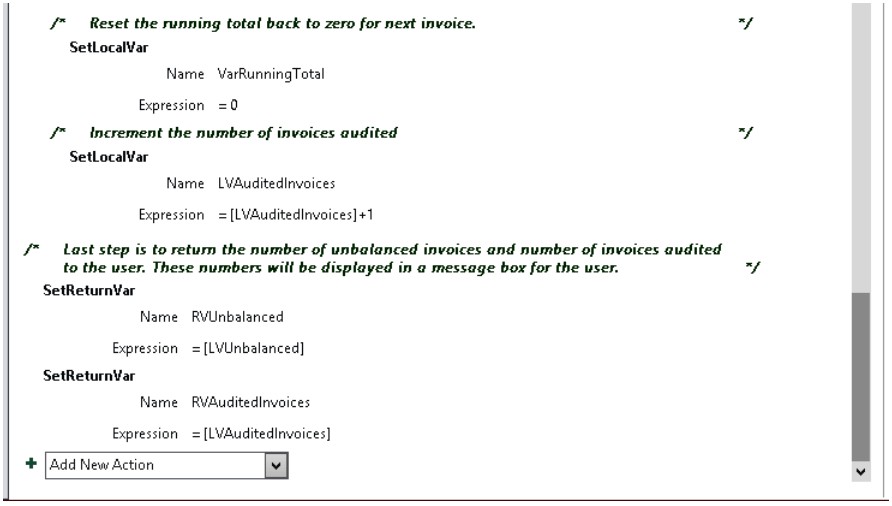


Figure 4-64 This is the last part of the dmAuditInvoiceTotalsOneVendor named data macro.

The logic for the last section of the dmAuditInvoiceTotalsOneVendor named data macro is as follows:

```
  Comment Block: Reset the running total back to zero for next invoice.
  SetLocalVar
    Name: VarRunningTotal
    Expression: 0
  Comment Block: Increment the number of invoices audited
  SetLocalVar
    Name: LVAuditedInvoices
    Expression: [LVAuditedInvoices]+1

Comment Block: Last step is to return the number of unbalanced invoices and number of
invoices audited to the user. These numbers will be displayed in a message box for
the user.
```

```
SetReturnVar
  Name: RVUnbalanced
  Expression: [LVUnbalanced]
SetReturnVar
  Name: RVAuditedInvoices
  Expression: [LVAuditedInvoices]
```

Now that we've completed checking one invoice inside the first ForEachRecord data block, we need to update two local variables before moving on to the next invoice. First, I need to update the VarRunningTotal local variable back to zero so that it is ready to start calculating the next invoice. Second, I need to update the LVAuditedInvoices local variable by one to account for the number of invoices Access audited. After this point, Access moves back to the beginning of the first ForEachRecord data block and completes the same steps previously outlined if another invoice exists within the given parameters. Access continues auditing each invoice one by one and updating all of the local variables as appropriate.

After Access completes auditing all invoices, the final piece of this named data macro is to set two return variables. As you might recall, this named data macro began with running a different named macro that used a return variable to bring data into this named data macro. I now end the logic in this named data macro by setting two return variables that any calling macro can use to see the results of this auditing macro. I set the first SetReturnVar data action—RVUnbalanced—equal to the local variable LVUnbalanced, which tracked the total number of unbalanced invoices. I set the second SetReturnVar data action—RVAuditedInvoices—equal to the local variable LVAuditedInvoices, which tracked the total number of audited invoices. In Chapter 7, you'll learn how to call this named data macro from a user interface macro and use the return variables in a message box.

As you can see, return variables are a very useful feature with data macros. When you use them in conjunction with parameters, you can create some very complex business logic at the data layer and even pass information back up to the user interface layer.

INSIDE OUT

Utilizing the Retrieve ID return variable

When you use the CreateRecord data block, Access displays a Retrieve ID link on the right side of the macro design surface. If you want to know the ID AutoNumber of the record Access creates inside a CreateRecord data block, you can click this link to retrieve the ID as a return variable. Access displays a SetLocalVar action inside a Parameters block where you can provide a name for the local variable. You can then use that local variable, passed from Access through the Retrieve ID return variable, in further actions of your data macro logic.

Studying other named data macros

The Back Office Software System sample web app includes many named data macros to automate various aspects of the app. Table 4-3 lists all the named data macros in the web app with a short description of their purpose. You can explore these samples for additional examples of how to design and use named data macros. In Chapter 7, you'll learn how to call some of these named data macros from user interface macros.

TABLE 4-3 Named data macros in the BOSS web app

Macro Name	Description
dmApplyLaborPlanDetails	Loops through all the labor plan details for a specific Labor Plan and creates new schedule records in tblSchedule.
dmAuditInvoiceTotalsAllVendors	Audits all invoices within a given date range.
dmAuditInvoiceTotalsOneVendor	Audits all invoices within a given date range for a specific vendor.
dmClearOutTraceTableRecords	Deletes all records from the Trace table.
dmCopyDateRangeRecords	Loops through all the schedule records within a date range and creates new schedule records in tblSchedule with the same information. The new schedule date to use comes from a parameter.
dmCopySingleDateRecords	Loops through all the schedule records for a specific date and creates new schedule records in tblSchedule with the same information. The new schedule date to use comes from a parameter.
dmDeleteDateRangeScheduleRecords	Deletes all records in tblSchedule within a given date range.
dmDeleteSingleDateScheduleRecords	Deletes all records in tblSchedule for a given date.
dmEnforceOnlyOnePrimaryPosition	Ensures that only one job code is marked as the primary position for a specific employee. This named data macro is called from both the On Insert and On Update tblTrainedPositions table events.
dmGetSettings	Gets application settings data from the tblSettings table.
dmSetJobCodeColor	Sets color choices in the tblJobCodes table from parameters passed in from user interface macros.
dmSwapSortOrders	Swaps sort order positions in the tblInventoryLocations table for two records. Uses saved query objects to find the highest and lowest values in the SortOrder field.

Macro Name	Description
dmUpdateSampleData	Adjusts date values of all sample data to work easily with data around the current time frame.
dmVerifyInvoiceBalanced	Checks to see whether a specific invoice is balanced.

Debugging data macros with the Trace table

You're likely to encounter unexpected errors or unintended results when you're designing data macros attached to table events and complex named data macros for the first time. You might even be wondering whether Access is even executing your data macros at all if you see no visible results. In Chapter 24, "Understanding Visual Basic fundamentals," which can be downloaded from the book's catalog page, you'll learn that you have several tools available in the Visual Basic Editor for debugging Visual Basic code in desktop databases. Data macros, unfortunately, do not have as rich a set of tools available for debugging purposes. For example, you cannot set breakpoints on data macro logic to halt execution. You also cannot single-step through the macro logic as you can with user interface macros in desktop databases.

Access can run into errors while you are in the development phase of creating, testing, and debugging your data macros. The best tool you have for debugging data macro logic is a special system table called the *Trace* table. Access manages any errors it encounters executing data macros through this system table. This Trace table serves two purposes:

- Access uses it to log any data macro failures that it encounters while executing data macros attached to table events and named data macros.
- You can use the table for debugging purposes when designing and testing data macros by viewing a history of everything Access executes while running your data macros in this table.

Earlier in this chapter, you studied the data macro logic attached to the On Update event of the tblTerminations table. When you update a termination record and assign the termination record to a different employee, the On Update logic looks up the previous employee's record in the tblEmployees table and sets the Active field back to Yes. In the first LookupRecord data block, I used the Old property to refer to the EmployeeID that Access just finished updating. In the second LookupRecord data block, Access sets the Active field to No for the employee you just selected. Our data macro logic, again, is as follows:

```
Comment Block: If we are modifying an existing termination record, one of two
possibilities exist: 1. The Employee that this termination is for remains unchanged -
Scenario is just updating some data for the termination record. 2. The Employee that
this termination is assigned to changed - Scenario here is that when the record was
first created, it might have been assigned to the wrong employee. In this case the
```

```
user is assigning this to a different employee.
Comment Block: Check to see if the Employee field was changed.
If Update([EmployeeIDFK]) Then
  Comment Block: The Employee field changed so we'll change the existing employee's
  status back to Yes.
  Comment Block: For the Where condition in this LookupRecord, use the Old value from
  the EmployeeIDFK field and find that employee's record.
  Look Up A Record In tblEmployees
    Where Condition = [tblEmployees].[EmployeeID]=[Old].[EmployeeIDFK]
      EditRecord
        Comment Block: Now set Active field to Yes for this employee since it was
        probably initially assigned to the employee in error.
        SetField
          Name: [tblEmployees].[Active]
          Value: Yes

      End EditRecord
End If
Comment Block: After modifying this termination record, make sure the employee that
it's assigned to now is marked as an inactive employee. To do that, we look up the
matching employee's record in the tblEmployees table and set the Active field to No.
Look Up A Record In tblEmployees
    Where Condition = [tblEmployees].[EmployeeID]=[tblTerminations].[EmployeeIDFK]
      EditRecord
        Comment Block: Now set Active field to No.
        SetField
          Name: [tblEmployees].[Active]
          Value: No

      End EditRecord
```

When you set up complex table events and named data macros like this example, you'll find it helpful to debug your logic to make sure everything is working just the way you want. For example, if you have only a few sample records in your tables, you might find it relatively easy to spot and fix any issues in your data macros with such a small data set. However, if your tables have many records, you might find it more difficult to spot any issues, or you might have a more difficult time tracking down what data Access updates. To help debug your data macro logic, you can take advantage of the built in Trace table.

For this specific On Update event example, it would be helpful to know which employee records, if any, Access updates during this table event. To start using the Trace table, the first thing that you need to do is to turn on data macro tracing for your web app. To do this, open any table event for any table in your web app or open any named data macro in Design view. After Access opens the Logic Designer, click the Data Macro Tracing button in the Tracing group on the Design contextual tab, as shown in Figure 4-65.

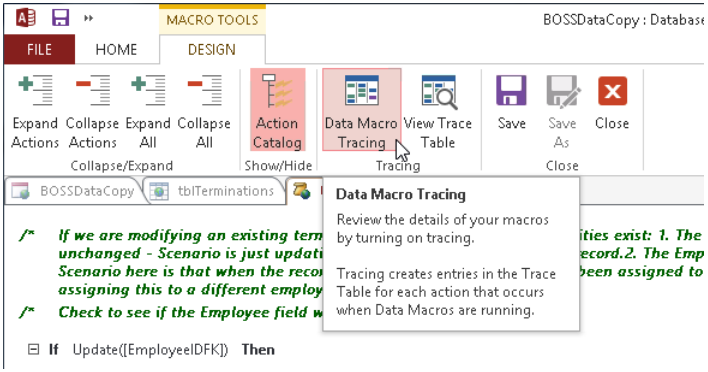


Figure 4-65 Click the Data Macro Tracing button to activate data macro tracing in your web app.

When you activate data macro tracing, Access records information to the Trace table. To see how useful the Trace table can be, let's change the existing termination record in the tblTerminations table. Switch to Datasheet view for the tblTerminations table. Next, find the existing termination record in this table—the one assigned to Mario Kresnadi. Next, tab over to the EmployeeIDFK for this record (the datasheet caption of the field displays Employee), type **Conrad** into the control where it currently says Mario Kresnadi, and then select Jeff Conrad from the drop-down list of employee names displayed in the Employee-IDFK field, as shown in Figure 4-66. Finally, click or tab off the record, and Access saves the record with Jeff Conrad's EmployeeIDFK number. (If you changed this record to Jeff Conrad previously in this chapter, change the value back to Mario Kresnadi. You'll be able to see results in the Trace table in either case.)

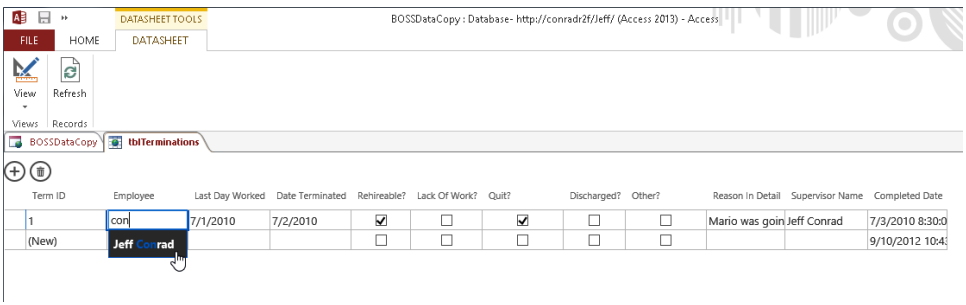


Figure 4-66 Change the EmployeeIDFK field from Mario Kresnadi to Jeff Conrad, and then save the record.

Note

Each web app you create includes the Trace system table, which is a hidden table. Therefore, you cannot create any table in your web app and name it Trace. If you do, Access informs you that an existing object with that name already exists in the web app.

To see the effects of this On Update event in the Trace table, switch to Design view for the tblTerminations table and then open any of three table events in Design view. After Access opens the Logic Designer, click the View Trace Table button in the Tracing group on the Design contextual tab, as shown in Figure 4-67.

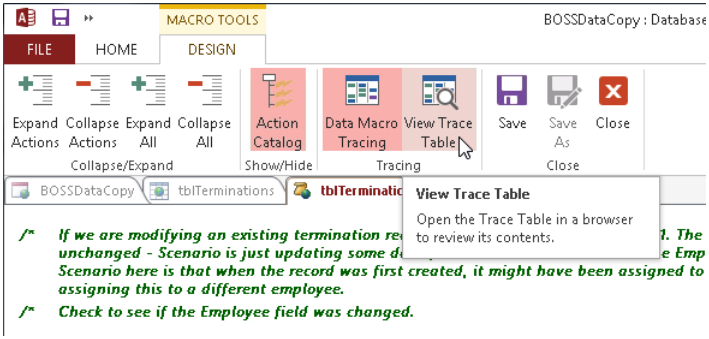


Figure 4-67 Click the View Trace Table button to open the data macro Trace table in your web browser.

Access opens the Trace table datasheet in your default web browser, as shown in Figure 4-68. The Trace table contains the following fields: ID, MacroName, ActionName, Operand, Output, TargetRow, Timestamp, and RuntimeErrorMessage. When you have data macro tracing turned on, Access creates a record in the Trace table for every action it runs during any table event or named data macro. Depending on the complexity of your data macro logic for a given table event or named data macro, you could see just a few new records in the Trace table or perhaps hundreds of new records. The Trace table holds a maximum of 1000 records. If the number of records in the Trace table exceeds 1000, Access begins deleting the oldest records as it creates new entries.

Note

In Figure 4-68, I resized several of the Trace table columns so that you could see more of the data in the various records. To resize a column in the Trace table, hover over the right edge of a column header until you see a double-sided arrow, click and hold your mouse, and drag the column to the right until you have the size you want.

ID	MacroName	ActionName	Operand	Output	TargetRow	Timestamp	RuntimeErrorMessage
1	tblTerminations:On Update	If	Update([EmployeeIDFK])			9/10/2012 11:04:	
2	tblTerminations:On Update	LookupRecord	tblEmployees;WHERE [tblEmployees].[EmployeeID]=[Old].[EmployeeIDFK]			9/10/2012 11:04:	
3	tblTerminations:On Update	LookupRecord			[EmployeeID] = 13 ; [Active] = 0	9/10/2012 11:04:	
4	tblTerminations:On Update	EditRecord				9/10/2012 11:04:	
5	tblTerminations:On Update	SetField	[tblEmployees].[Active]	1		9/10/2012 11:04:	
6	tblTerminations:On Update	LookupRecord	tblEmployees;WHERE [tblEmployees].[EmployeeID]=[tblTerminations].[Emplo			9/10/2012 11:04:	
7	tblTerminations:On Update	LookupRecord			[EmployeeID] = 31 ; [Active] = 1	9/10/2012 11:04:	
8	tblTerminations:On Update	EditRecord				9/10/2012 11:04:	
9	tblTerminations:On Update	SetField	[tblEmployees].[Active]	0		9/10/2012 11:04:	
(New)							

Figure 4-68 Access opens the Trace table in your web browser so that you can examine how your data macro logic executes.

The ID field in the Trace table is the AutoNumber field Access uses for this table. The MacroName field lists the name of the table and the specific event Access executed or the name of the named data macro Access executed. The ActionName field lists the name of the program construct, data block, or data action Access executed. In the Operand field, Access lists any conditional expressions or table and field references in the case of SetField data actions. In the Output field, Access lists data values it commits into a field. In the TargetRow field, Access lists identifying information about what record it is writing data to, such as the ID values. In the TimeStamp field, Access enters the current date and time of the specific action. In the RuntimeErrorMessage field, Access displays a SQL exception message if it encounters an error while performing the specific action. Access also logs any RaiseError messages that you define into the RuntimeErrorMessage field. Table 4-4 summarizes the important information Access logs to the Trace table.

TABLE 4-4 Trace table logging information

Action Name	Operand	Output	Target Row
If	Conditional expression		
Else If	Conditional expression		
CreateRecord	Table name		
EditRecord			
ForEachRecord	Table name, Where condition		Record identifiers
LookupRecord	Table name, Where condition		Record identifiers
CancelRecordChange			
DeleteRecord			
ExitForEachRecord			
RaiseError			

Action Name	Operand	Output	Target Row
RunDataMacro	Macro name	Computed expression value	
SetField	Table and field name	Computed expression value	
SetLocalVar	Variable name	Computed expression value	
SetReturnVar	Variable name	Computed expression value	
StopMacro			

In our example On Update event, you can see that Access logged nine records carrying out all the actions in the On Update table event. In the first record, Access displays the If block and the conditional expression. In the second record, you can see that Access displays the LookupRecord in the ActionName column along with the table name and Where condition in the Operand field. In the third record, Access repeats the LookupRecord data block but this time displays the ID values of the record it found that match the Where condition. In the fourth record, Access lists EditRecord, which indicates it is now entering this data block because it found a matching record in the Where condition of the LookupRecord data block. In the fifth record, Access displays the SetField action along with the table name and field in the Operand field. In the Output field for this record, Access displays 1, which, in this case, indicates a Yes value for the Active Boolean field. In the remaining four records in the Trace table, Access lists similar information detailing the second LookupRecord actions outlined in the On Update event of the tblTerminations table. In the SetField record, you can see that Access set the Active field to 0, which indicates No for the Boolean field.

INSIDE OUT Clearing the Trace table records

When you view the Trace table records in your web browser, you can delete the records if you no longer want to see them by highlighting the records and pressing the Delete key. Depending on how many records you have in your Trace table, this could be a tedious task. You can alternatively write a named data macro to perform the task very quickly. The Trace table is a hidden table in your web app, and therefore you cannot use the Trace table directly in a ForEachRecord or LookupRecord data block. However, you can create a saved query object that uses the Trace table as its source. You can then create a named data macro that includes a DeleteRecord action inside a ForEachRecord data block with the saved query as the source, which causes Access to delete all records in the Trace table when you call the named data macro.

In the Back Office Software System web app, I've included a saved query called qryTraceTable and a named data macro called dmClearOutTraceTableRecords that perform this task. If you have data macro tracing turned on while running that named data macro, Access records all the delete operations into the Trace table, which effectively cancels out what you're trying to do! To work around that issue, you must turn off data macro tracing first and then execute the named data macro to clear out all records in the Trace table using this technique.

Let's examine a different example of a table event that triggers a named data macro so that you can see how Access logs this type of scenario to the Trace table. Earlier in this chapter, you studied the On Update event of the tblTrainedPositions table. Open this table in Datasheet view within Access, and then change either the first or second record such that you've changed the primary position of the first employee listed in the table. After you update the record, refresh the Trace table in your web browser. In Figure 4-69, you can see the six records Access adds to the Trace table while executing the data macro logic in the On Update event.

	10	tblTrainedPositions:On Update	If	[tblTrainedPositions].[PrimaryPosition]=Yes			9/12/2012 10:39
	11	tblTrainedPositions:On Update	RunDataMacro	dmEnforceOnlyOnePrimaryPosition			9/12/2012 10:39
	12	dmEnforceOnlyOnePrimaryPosition	ForEachRecord	tblTrainedPositions;WHERE [tblTrainedPositi			9/12/2012 10:39
	13	dmEnforceOnlyOnePrimaryPosition	ForEachRecord			[TrainedPositionsID] = 2 ; [PrimaryPosition] = 1	9/12/2012 10:39
	14	dmEnforceOnlyOnePrimaryPosition	EditRecord				9/12/2012 10:39
	15	dmEnforceOnlyOnePrimaryPosition	SetField	[tblTrainedPositions].[PrimaryPosition]	0		9/12/2012 10:39
	(New)						

Figure 4-69 Refresh the Trace table in your browser to see new logging records Access adds to the table.

If you follow the information that Access displays in the Trace table after you updated the record, you can see that Access first indicates it fired the On Update event in

tblTrainedPositions and then executes the named data macro called dmEnforceOnlyOnePrimaryPosition, which you created earlier. You can see that Access executed the ForEachRecord data block in the named data macro, found a specific record that matched the Where condition, and then used a EditRecord with SetField action to update the PrimaryPosition field.

After Access finishes executing your data macro logic, you can examine the values of your local variables and return variables in the Trace table at different points in time to help determine what Access is doing during the data macro execution. You can use this information to assist with debugging your logic. For example, you can examine which record Access might be editing or attempting to find by looking at the values in the TargetRow column.

INSIDE OUT

Turn off data macro tracing in production apps

The Trace table can be very useful when you are designing and testing the logic in your table events and named data macros. When you have everything working just the way you want, you should turn off data macro tracing before putting your app in production for people to use. If you leave data macro tracing turned on in a production environment, Access continually logs information for all data macro logic. You'll see a slight improvement in app performance by turning this feature off in production, because Access does not need to spend extra time writing data to the Trace table for all of your actions.

To turn off data macro tracing, open any table event or named data macro in Design view and click the Data Macro Tracing button in the Tracing group on the Design contextual ribbon tab. This button is essentially a toggle button. When you have data macro tracing turned on in your web app, you'll see this button highlighted in the ribbon. Just click the button in the ribbon again to deselect it and turn off data macro tracing. If you are encountering errors in your production apps, you can turn the data macro tracing back on temporarily, diagnose the issues, fix the issues, and then turn it back off when your data macro logic is working again as you expect.

Note

If you save your web app as an app package, Access includes the Trace table, and any records included in it, into the app package.

Understanding recursion in data macros

When you're designing data macros, you have the potential to run into a recursion issue. Access runs into a recursion issue when it tries to execute the same data macro logic over and over in a repeated loop. For example, suppose that you created data macro logic attached to the On Update event of a table that changed data in the current record of the same table. Access makes the field changes and then commits the data. Access then fires the On Update again because data in the table changed. The On Update event fires again, changes the data, and the cycle begins again. Access is now in a perpetual loop executing the data macro in the On Update event. Access could also get into a loop, for example, when working with two tables that have On Update events that update each other, or even with complex named data macros that end up repeating themselves.

Data macros are limited to 32 levels of recursion, which means Access stops the data macro execution after 32 iterations through a recursive loop. If Access falls into a recursive loop, you'll see a runtime error message indicating that an endless loop was detected, as shown in Figure 4-70. Access logs a SQL Exception error into the RuntimeErrorMessage field of the Trace table if you have data macro tracing turned on. In the Trace table, you'll see essentially the same records getting updated over and over again by Access.

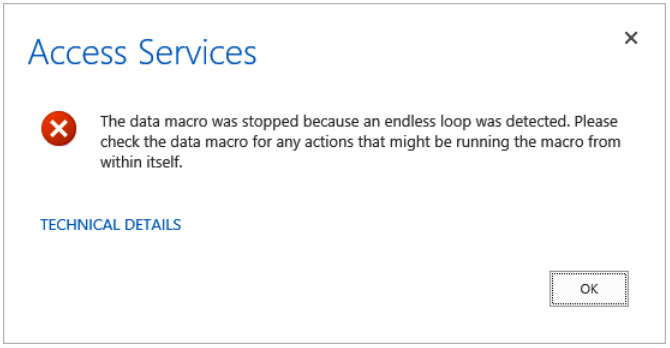


Figure 4-70 Access displays an error message if it gets into a data macro recursion loop.

In most cases, you can correct recursive calls by using the Update function to determine which field or fields Access changed in the last record update. You can add conditional logic with If blocks to determine whether a field was changed and perform different actions, or no actions, based on the evaluation of the condition. As you are designing and testing your data macro logic, it's a good idea to check the Trace table continually to help spot potential problems with recursion.

Sharing data macro logic

The Logic Designer in Access includes a very useful feature for sharing and reusing data macro logic. To illustrate this feature, open the tblTerminations table in the Back Office Software System data copy sample web app (BOSSDataCopy.app) in Design view. Next, click the On Insert button in the Events group to open the Logic Designer. You've already explored the data macro logic attached to this table event earlier in this chapter. Press Ctrl+A to highlight all the logic on the macro design surface, and then press Ctrl+C to copy the logic to the Windows Clipboard. Now open Notepad (or a different text editor), and then press Ctrl+V to paste all the logic into Notepad.

As you can see in Figure 4-71, Access copies the data macro logic from the Logic Designer as Extensible Markup Language (XML). You can send this XML to someone else, and that person can copy and paste the XML directly into a Logic Designer window for a data macro in his or her Access 2013 web app. This feature can be especially useful if you are trying to help someone else write or debug data macro logic, such as in an Access forum or newsgroup. You can create the logic for the person you're helping and explain how you structured the program flow constructs, data blocks, and data actions.

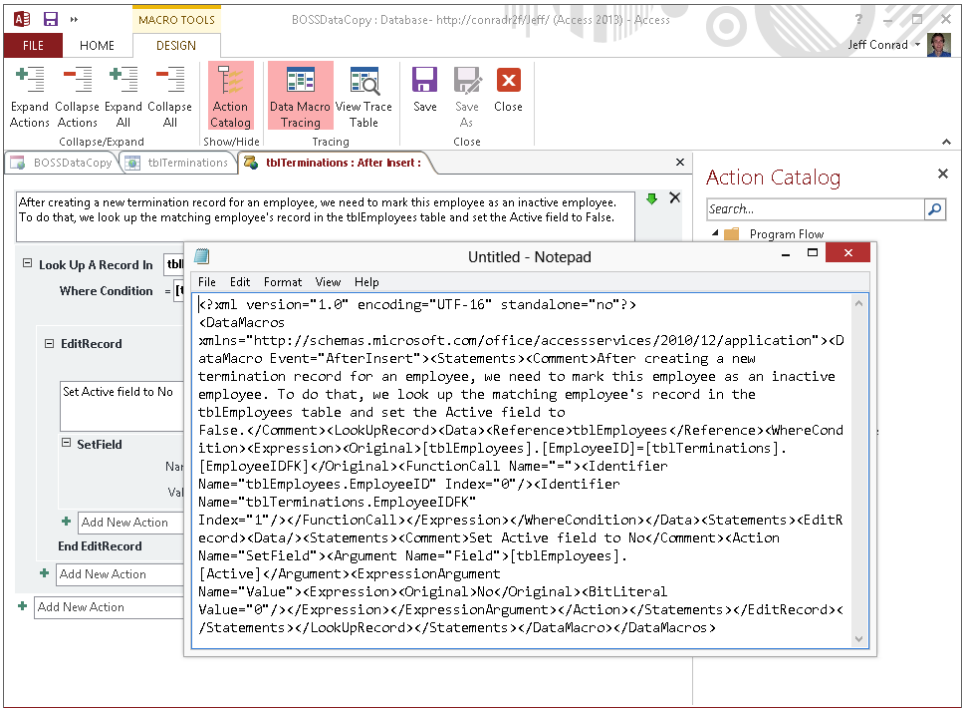


Figure 4-71 You can copy and paste data macro logic directly out of the Logic Designer.

You now have all the information that you need to modify and maintain your web app table definitions. You know how to build tables, modify them, import data and link them, and create data macros to automate them. In the next chapter, you'll learn how to extract data from tables by building queries.

Index

B

D

E

F

M

R

S

T

U

V

W

X

Y

Z

About the author

Jeff Conrad started working with Access when he saw a need at his full-time position for a database solution. He bought a book on Access and began teaching himself how to use the program to solve his business's needs. He immediately became hooked on the power and ease of working with Access.

Jeff found a home in the Microsoft Access newsgroups asking questions as he was learning the ins and outs of Access and database development. He now enjoys giving back to a community that helped him when he was first learning how to use Access. He has been an active participant for many years in the Access newsgroups and online forums where he is best known as the Access Junkie.

Jeff also was awarded Microsoft's Most Valuable Professional award from 2005 to 2007 for his continual involvement with the online Access community. He maintains a website with a wealth of information and resource links for those needing guidance with Access (*http://www.AccessJunkie.com*). He co-authored *Microsoft Office Access 2007 Inside Out* with John Viescas and authored *Microsoft Access 2010 Inside Out*. Jeff is currently employed by Microsoft as a Software Design Engineer in Test working with the Access development team.