QAD Enterprise Applications

Training Guide

QAD Introduction to Progress Programming

70-3259-2015
Progress Version 10
April 2015

This document contains proprietary information that is protected by copyright and other intellectual property laws. No part of this document may be reproduced, translated, or modified without the prior written consent of QAD Inc. The information contained in this document is subject to change without notice.

QAD Inc. provides this material as is and makes no warranty of any kind, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. QAD Inc. shall not be liable for errors contained herein or for incidental or consequential damages (including lost profits) in connection with the furnishing, performance, or use of this material whether based on warranty, contract, or other legal theory.

QAD and MFG/PRO are registered trademarks of QAD Inc. The QAD logo is a trademark of QAD Inc.

Designations used by other companies to distinguish their products are often claimed as trademarks. In this document, the product names appear in initial capital or all capital letters. Contact the appropriate companies for more information regarding trademarks and registration.

Copyright © 2015 by QAD Inc.

ProgressProg10_TG_2015_04.pdf/biw/mdf

QAD Inc.
100 Innovation Place
Santa Barbara, California 93108
Phone (805) 566-6000
http://www.qad.com

Contents

QAD Introduction to Progress Programming Change Summary

The following table summarizes significant differences between this document and previous versions.

Date/Version	Description	Reference
April 2015/Progress 10	Rebranded for QAD Progress 10	---
April 2014/Progress 10	Numerous minor revisions	---
	Removed obsolete information regarding browses	---
	Added note to prevent environment login or performance issues	page 4
	Revised guide to state that revision level is only passed to mfdtitle.i in SE	page 158
October 2013/Progress 10	Minor revisions	---
October 2012/Progress 10	Revised Virtual Environment Information	page 3
August 2012/Progress 10	Edit for consistency and clarity; complete technical review; verify and update sample programs; add appendix with exercise answers	---
May 2012/Progress 10	Initial version of updated and reorganized guide	---

About This Course

Lesson 1. Overview

Lesson 1 includes the following topics:

- Course description
- Course agenda
- Additional resources
- Introduction to using sample programs provided for class activities

Course Description

QAD designed this course for programmers working with QAD non-component-based code. The intent of this course is to introduce basic database and programming concepts as they apply to QAD applications. Additionally, basic Progress programming syntax is covered to include custom report writing or changes to existing reporting capability. A working view of the data structures is also presented.

The course includes hands-on programming to understand key concepts, as well as sample program snippets for later reference. The sample programs are already available in the classroom training environment.

Each topic in the course follows the same format, beginning with an overview of the concepts and commands to be discussed, followed by a detailed explanation and a hands-on exercise. Sample programs representing solutions for the exercises are provided in an appendix to the guide as well as in a folder in the training environment.

Course Objectives

By the end of this class, students will know how to:

- Use basic Progress programming principles and the QAD data definitions
- Use the Progress Editor
- Create Progress procedures
- Calculate fields and report totals
- Access data from multiple tables
- Use standard QAD include files
- Understand more advanced Progress functions including arrays, temp-tables, scoping, and ProDataSets

Course Benefits

- A basic foundation for programming non-component-based code used in QAD products
- Ability to more fully utilize the features provided by QAD applications

Audience

This course is intended for:

- Beginning programmers who plan to customize reports in QAD applications or who need to know how to write simple queries
- Project managers who want to understand the fundamentals of developing non-component based QAD applications
- Members of QAD implementation teams

Prerequisites

Students should have a general understanding of QAD Enterprise Applications and an interest in how it works. Prior programming or database administration experience is suggested, but not required.

QAD Enterprise Applications Version

The course is intended to be taught with the Progress Version 10. Most basic principles of Progress programming are independent of the particular version of QAD Enterprise Application being used. However, significant differences in the database structure of QAD Enterprise Edition and QAD Standard Edition do affect some points of the training.

The training assumes that you are using the latest QAD Enterprise Edition version of the product suite, that includes the enhanced Enterprise Financials. But the concepts covered apply regardless of the version.

Virtual Environment Information

A virtual environment is only available during QAD classroom training. For information on how to register for a class, go to:

http://learning.qad.com

The hands-on exercises in this book should be used with the environment specified by your instructor. A special simplified database has been designed for use with this class, to make it easier to write simple inquiries. Unlike the standard application training database, the database used for the Progress training contains only a small number of tables with a small number of records.

Progress Editor

The navigation instructions in this class assume that you are using the GUI version of the Progress Editor. If you are using the character-based version of the Editor, you will need to adjust these instructions. The screen appearance and shortcut keys are different in the two versions.

Domains

Beginning with QAD Standard Edition (previously known as MFG/PRO eB2.1), QAD introduced the concept of domains, which represent separate business areas. To support this concept, the domain field was added to most tables in the QAD database to support the segregation of data by domain. In the QAD code base, a global variable is used to determine the user's current domain so that the correct records are returned. A standard include file is used to set this variable correctly.

In this class, the domain is being set manually to the value 10USA, one of the domains that exists in the training database. This approach keeps the code in the sample exercises simple and easy to understand.

Important If you attempt to use these exercises in a database other than the one the book is designed to work with, the exercises will fail if domain 10USA does not exist.

Special Environment Considerations

Significant architectural differences are found between QAD Enterprise Applications Standard Edition and Enterprise Edition. However, this class explores very basic Progress programming principles that are the same in both.

You should note the following special considerations about the training environment:

- Since the training database includes only a few tables, some of the examples discussed in the class reference tables that do not exist in this small database. These sample procedures are for reference and cannot be executed in the training environment.
- The class only applies to the operational code base of QAD Enterprise Edition, not the Enterprise Financials, which are managed with a different architecture.

Note To prevent training environment login or performance issues, be sure to close the Progress Editor at the end of each session.

Connecting to the Training Database

To connect to this database, follow these steps:

1. Double-click the Progress OE10 Training icon on the training desktop.
2. This will start the Progress Editor with the following small program displayed:

```
/* APPEND CLASS WORKING DIRECTORY TO BEGINNING OF PROPATH */
propath = "c:\_OE10\code,c:\_OE10\Exercises,c:\_OE10\examples,c:\_OE10\labs,"
        + propath.

/* CONNECT TO CLASS DATABASE */
connect c:\_OE10\db\train -1.

/* WE'RE DONE */
message "Connected." view-as alert-box.
```

3. Choose Compile|Run from the Editor menu to execute this program and connect to the training database.
4. When connected, choose File|Close to close the connection program.

5 Sample programs for use during the class exercises are located in the following directory of the virtual environment:

```
C:\_OE10\Exercises
```

Exercise Solutions

To supplement the exercises, a folder of exercise solutions is also provided with executable programs:

```
C:\_OE10\Exercises\ExerciseAnswers
```

You can also find the answers to non-program questions as well as the program code samples in Appendix A, “Exercise Answers,” on page 295.

Course Agenda

This class is divided into three major sections, introduced by an overview. Each section includes a number of lessons.

- Lesson 1: Overview
- Section 1: Getting Started and Creating Simple Reports
  - Lesson 2: Introduction to Progress
  - Lesson 3: Using the Progress Editor
  - Lesson 4: Creating simple Progress procedures
  - Lesson 5: Formatting, sorting, and report totals
  - Lesson 6: Selecting specific records
- Section 2: Writing More Complex Reports
  - Lesson 7: Using FIND, REPEAT, and defined variables
  - Lesson 8: Accessing multiple tables
  - Lesson 9: Frame control
  - Lesson 10: Using include files
  - Lesson 11: Arrays, temp-tables, buffers, and other useful functions
- Section 3: Advanced Topics
  - Lesson 12: Scoping
  - Lesson 13: ProDataSets
  - Lesson 14: Triggers and events (optional)
  - Lesson 15: Queries and handles (optional)

This class is intended to be completed in three days. Some topics are optional so that the class can be completed in two days if necessary.

Additional Resources

Progress Resources

The OpenEdge Progress programming language is a product of Progress Software Corporation. Extensive documentation on all Progress products can be found on the company Web site:

http://www.progress.com/

Documentation in PDF format for OpenEdge 10 is found here:

http://communities.progress.com/pcom/docs/DOC-16074/

Progress Software Corporation also provides Web-based training opportunities.

QAD Resources

If you encounter questions on QAD software that are not addressed in this book, several resources are available. The QAD corporate Web site provides product and company overviews. From the main site, you can access the QAD Learning or Support site and the QAD Document Library. Access to some portions of these sites depends on having a registered account.

http://www.qad.com/

QAD Learning Center

To view available training courses, locations, and materials, use the QAD Learning Center. Choose Education under the Services tab to access this resource. In the Learning Center, you can reserve a learning environment if you want to perform self-study and follow a training guide on your own.

QAD Document Library

To access release notes, user guides, training guides, technical references, and installation and conversion guides by product and release, visit the QAD Document Library. Choose Document Library under the Support tab. In the QAD Document Library, you can view HTML pages online, print specific pages, or download a PDF of an entire book.

To find a resource, you can use the navigation tree on the left or use a powerful cross-document search, which finds all documents with your search terms and lets you refine the search by book type, product suite or module, and date published.

Of particular interest for this class are the two technical references: *Database Definitions* and *Entity Diagrams*. These two references provide programmers with the information they need to understand relationships between tables and the kind of data stored in the tables. They are both referenced in this class.

QAD Support

Support also offers an array of tools depending on your company’s maintenance agreement with QAD. These include the Knowledgebase and QAD Forums, where you can post questions and search for topics of interest. To access these, choose Visit Online Support Center under the Support tab.

QAD Training

QAD supports an extensive set of reporting tools that can be used to generate both simple and complex reports, without having to write custom queries. You can learn more about these tools in these classes:

- QAD Reporting Framework
- .NET UI Administration

QAD Development Standards

QAD has developed an extensive set of reference information about preferred coding practices, which it makes available to users on the QAD Support site. After accessing the Support site, choose Tools, then General Reference to find the link to the Development Standards.

Using Class Materials

Your class materials consist of this book, which is yours to keep. We encourage you to take notes in the book. The QAD Enterprise Applications *Data Definitions* and *Entity Relationship* manuals are also available for reference. While your instructor will lecture on the main topics covered in the course, you should read the explanation in this guide as well. You may also find it useful to refer to this guide when you are back at work.

Most of the topics are illustrated with exercises. The data you enter when you are learning the system cannot harm the training database. Some of the topics build on information presented earlier, so it is important to complete the exercises in order.

Conventions for Progress Syntax Used in the Class

To highlight Progress syntax, the following conventions are used in the examples in this course:

- Progress keywords are shown in upper case for emphasis in the code. However, Progress is not case sensitive. The QAD standard in source code is to use all lower case, with a few exceptions.
- Table and field names are shown in lower case.
- Character strings are surrounded by quotes (" ").
- Square Brackets ([]) indicate optional items.
- Ellipses (…) indicate an item may be repeated.

Sample Programs

Sample programs like the one shown in “Sample Inquiry” on page 9 are provided with the training environment. You will learn how to build a query similar to this sample, which illustrates the various commands introduced in the course. The sample program code structure includes the following elements:

- Header
- Define variables, parameters, buffers, temp-tables...
- Define forms

- Procedure body
- Exit point
- Clean up
- Internal procedures and functions

Note Not all sample programs can be executed in the training environment because they reference tables that are not included.

Data for Use with Sample Programs

Some of the class exercises require input of a valid item or customer in order to display records. Others require the student to write a program that finds data. Since most data in the database is stored by domain, a valid domain name is also required.

For these exercises, the following data should be used:

- Domain (*_domain): 10USA (where * represents the table name prefix in the exercise, such as so_domain)
- Item (pt_part): 01010
- Customer (cm_addr or so_cust): 10C1001

The sample in the following section illustrates the use of the 10USA domain.

Modifying Sample Programs

In many exercises, you are asked to modify lines in the supplied sample programs. In the instructions, the From code is provided in the sample program. The To code is the code that you create or modify. This is illustrated in the following example.

```
From:                pt_desc1
                     pt_desc2
To:                  pt_desc1 + pt_desc2 FORMAT "x(40)"
```

1 Locate the lines in the sample procedure described in the From condition.

```
FOR EACH pt_mstr WHERE pt_domain = "10USA" NO-LOCK:
      DISPLAY        pt_part
                     pt_desc1
                     pt_desc2
                     pt_um       NO-LABEL
                     pt_price    FORMAT "-9,999.99".
END.
```

2 Change the lines described in the From condition to correspond to the To condition

```
FOR EACH pt_mstr WHERE pt_domain = "10USA" NO-LOCK:
     DISPLAY         pt_part
                     pt_desc1 + pt_desc2 FORMAT "x(40)"
                     pt_um NO-LABEL
                     pt_price FORMAT "-9,999.99".
END.
```

Sample Inquiry

```
/* sample-inquiry.p QAD Enterprise Applications Inquiry Template Program */

{us/mf/mfdtitle.i "a"}

DEFINE VARIABLE cust  LIKE cph_cust               NO-UNDO.
DEFINE VARIABLE cust1 LIKE cph_cust  LABEL "To"   NO-UNDO.
DEFINE VARIABLE part  LIKE pt_part                NO-UNDO.
DEFINE VARIABLE part1 LIKE pt_part   LABEL "To"   NO-UNDO.

/* Form Definitions */
FORM
   SKIP(1)
   cust    COLON 20
   cust1   COLON 50
   part    COLON 20
   part1   COLON 50
WITH FRAME a WIDTH 80 SIDE-LABELS TITLE "Customer Sales Inquiry ".

FORM
   cph_cust
   cph_part    FORMAT "x(8)"
   pt_desc1
   cph_sale(6) column-label "June!Sales"
   cph_cost(6) column-label "June!Cost"
WITH FRAME custpart DOWN width 80 title " June Sales ".

REPEAT:
   IF cust1 = hi_char THEN cust1 = "".
   IF part1 = hi_char THEN part1 = "".

   UPDATE
      cust
      cust1
      part
      part1
   WITH FRAME a.

   /* OUTPUT DESTINATION SELECTION */
   {us/gp/gpselout.i &printType          = "terminal"
             &printWidth               = 80
             &pagedFlag                = " "
             &stream                   = " "
             &appendToFile             = " "
             &streamedOutputToTerminal = " "
             &withBatchOption          = "no"
             &displayStatementType     = 1
             &withCancelMessage        = "no"
             &pageBottomMargin         = 6
             &withEmail                = "yes"
             &withWinprint             = "yes"
             &defineVariables          = "yes"}

   IF cust1 = "" THEN cust1 = hi_char.
   IF part1 = "" THEN part1 = hi_char.

   FOR EACH cph_hist NO-LOCK
      WHERE cph_domain = global_domain
      AND  (cph_cust  >= cust AND cph_cust <= cust1)
      AND  (cph_part  >= part AND cph_part <= part1)
   BREAK-BY cph_cust:

      DISPLAY
         cph_cust
         cph_part
         cph_sale(6)
         cph_cost(6)
       WITH FRAME custpart.

       FIND pt_mstr NO-LOCK
          WHERE pt_domain = global_domain
          AND   pt_part   = cph_part
```

```
       NO-ERROR.

       IF AVAILABLE pt_mstr THEN
          DISPLAY
             pt_desc1
          WITH FRAME custpart.
       ELSE
          DISPLAY
             "" @ pt_desc1
          WITH FRAME custpart.

      DOWN 1 WITH FRAME custpart.

      /* ALLOW PROGRAM INTERRUPTION */
      {us/mf/mfrpchk.i}
   END.   /* FOR EACH cph_hist */
END.   /* repeat */
```

Chapter 1

Getting Started and Creating Simple Reports

Agenda for Section 1

This section of the class consists of five lessons:

- Lesson 2: Introduction to Progress
- Lesson 3: Using the Progress Editor
- Lesson 4: Creating Simple Procedures
- Lesson 5: Formatting, Sorting, and Report Totals
- Lesson 6: Selecting Specific Records

Each lesson includes goals, concepts, and hands-on exercises.

Lesson 2: Introduction to Progress

In this lesson, you are introduced to Progress and the application database. Progress forms the core of your software application. It controls the way in which information is stored and retrieved, and provides the tools you need to create your own reports and inquiries.

Upon completion of this lesson, you will be familiar with the following topics:

- Progress components
- Database concepts
- The Data Dictionary
- Table and field naming conventions
- Progress field definitions
- Indexes
- QAD database structures and layout

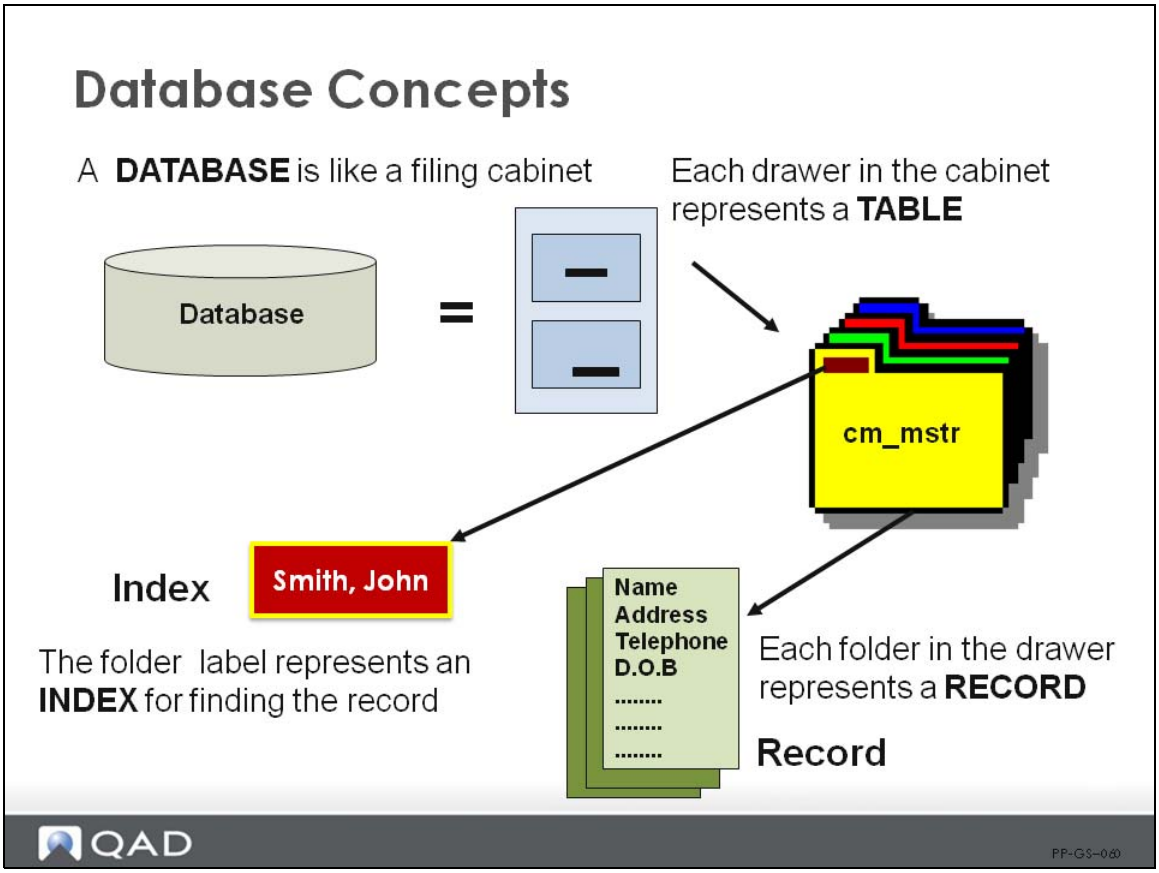
Progress Components

Progress consists of the following components:

- The Database Manager manages all interactions with information in the database to ensure the security and integrity of your data.
- The Data Dictionary describes the information stored in your database. This includes what data is in the database, the structure of the data, and how it will be accessed.
- The Progress language is what you use to write procedures that retrieve and display information from your database. Progress is a 4GL or fourth-generation language, which represents a class of programming languages closer to human languages than typical high-level programming languages. Progress is also described as an ABL, advanced business language.
- The Frame Manager controls the way your information is displayed. It provides default output formats for displaying information.
- The Editor is what you use to write and edit Progress language procedures.

Each component is discussed in more detail.

Database Concepts



Database. A database is a place to store information, like a file cabinet. The cabinet contains files on multiple subjects such as employees or projects. The cabinet contains file folders of information on those subjects.

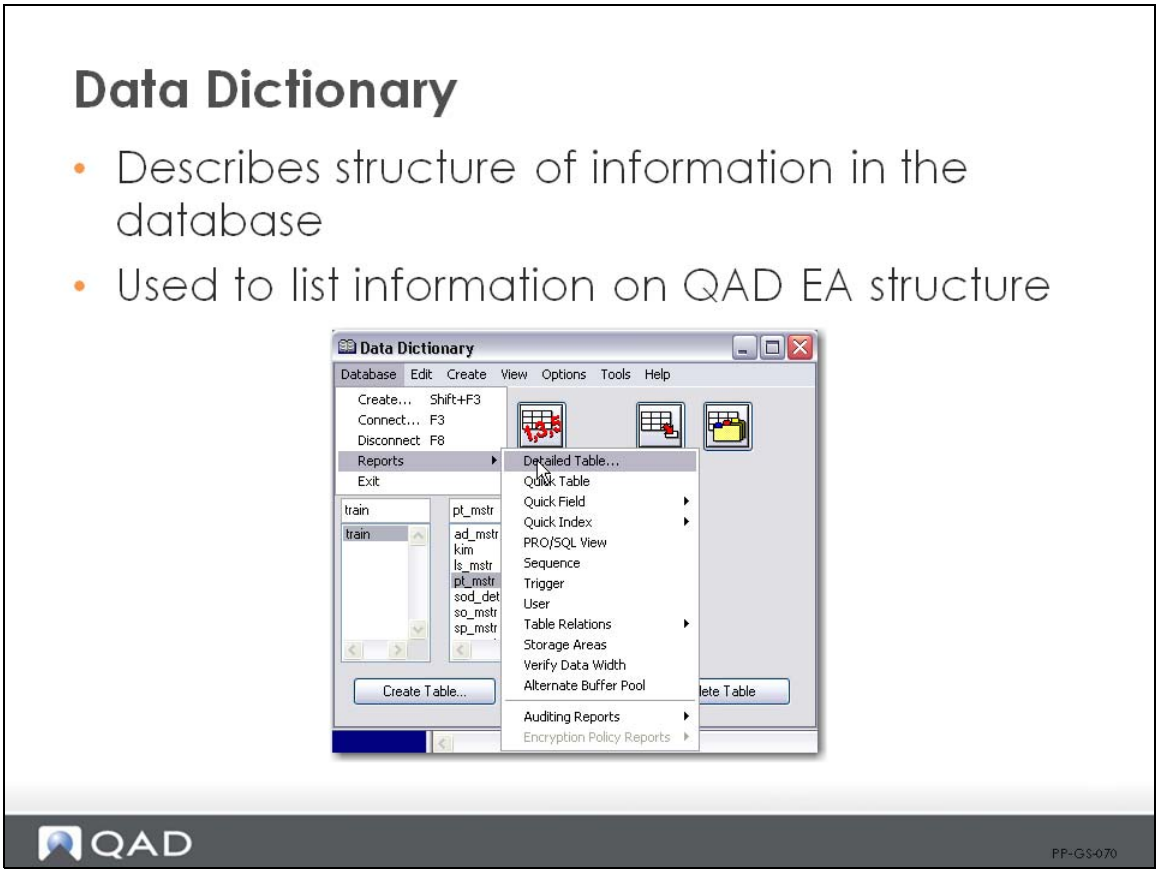
Table. A table is a group of related information, similar to a file drawer full of similar files, such as employee files. A file contains many file folders and a table contains many records. Each table in a database is unique.

Record. A record is a complete set of information in one specific folder such as for John Doe. A record is one of the file folders in the file drawer. Records contain many pieces of information about each unique entity.

Field. Each piece of information within a folder or record is called a field. Fields contain the actual data values. Multiple fields make up a record within a table. The employee name is a field, as are the employee address and the employee title.

Index. An index is a device used to locate a specific record, similar to the way you use a labeled tab on a file folder. To find the record for John Doe, you scan all of the file folder tabs. You usually keep your file folders in alphabetical order in the drawer, so that you only have to look through a few folders to locate the one John Doe. Progress uses an index in the same way, to quickly locate the records you want. You can also use indexes to retrieve records in a certain order, such as in item number order.

Data Dictionary



The Data Dictionary is the tool you use to define and manage all the tables and fields in your application database. You can also use it to view information that is already defined.

Access to the Progress Editor can be restricted through security. How security is set up varies depending on your version of QAD EA.

Exercise 2-1: Enter Progress Editor

Note In this class, you access the Editor by double-clicking the Progress OE10 Training icon on the desktop of the training environment.

To enter the Progress Editor in a QAD Enterprise Edition database:

1. From menu search, type `mgeditor.p`.
2. The Progress Editor displays.

To enter the Progress Editor in a QAD Standard Edition database:

1. From the QAD EA main menu option line, press End as you would to exit.
2. The system prompts you to confirm exit. Instead of typing N or Y, type P when you are prompted.

   Y quits the application.

N returns to the menu option line.

3 Press Enter or Return key.

Exercise 2-2: Enter Data Dictionary

1 From the Progress Editor, do one of the following:

   a Select the Tools menu option, then Data Dictionary.

   Or

   b Type dict in the Progress Editor screen and then press F2 (GUI) or F1 (character).

2 A listing of connected databases and the tables in the currently highlighted database displays.

3 Select the table in the list that you want to view.

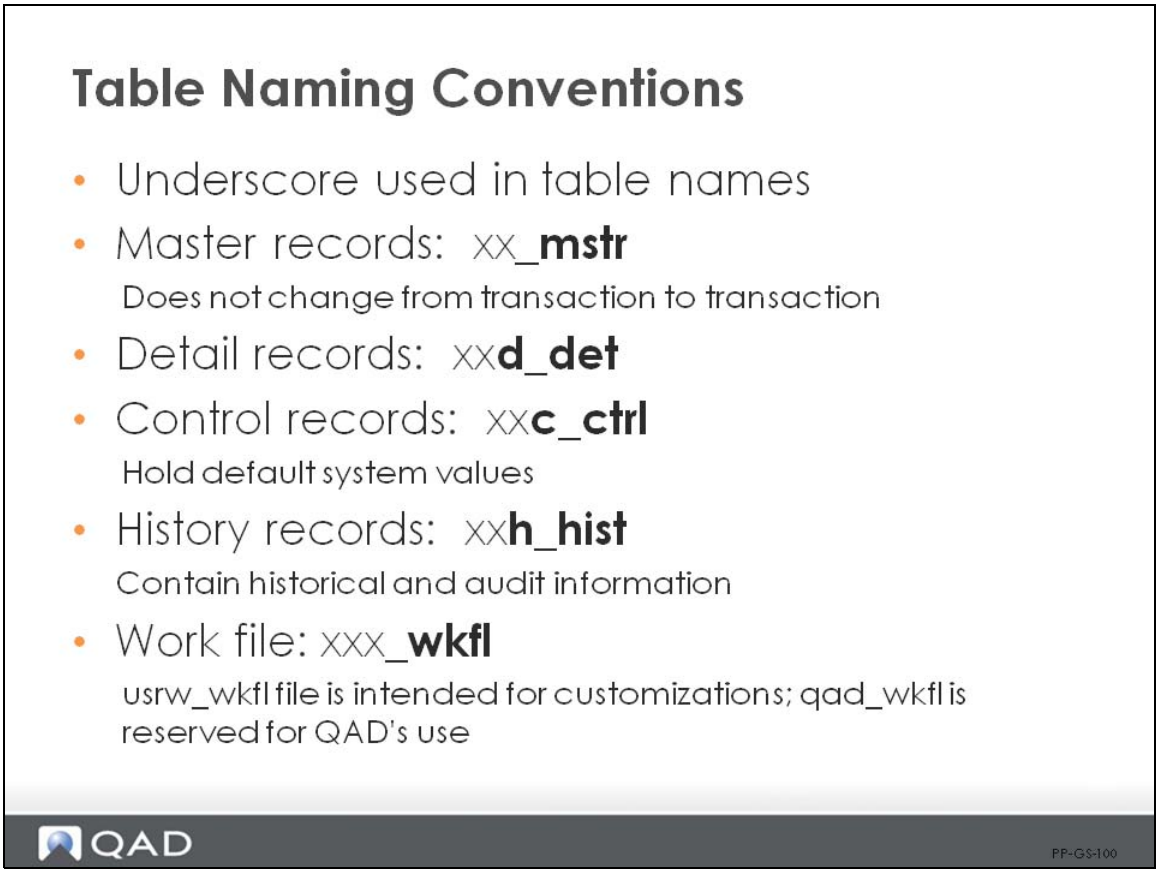
4 Select the Database menu option and then select Reports.

5 Select Detailed Table from the submenu.

6 Select Terminal for Report Options.

7 When the output displays, scroll through the information to see what is included: field names, formats, initial values, indexes, and so forth.

Table Naming Conventions



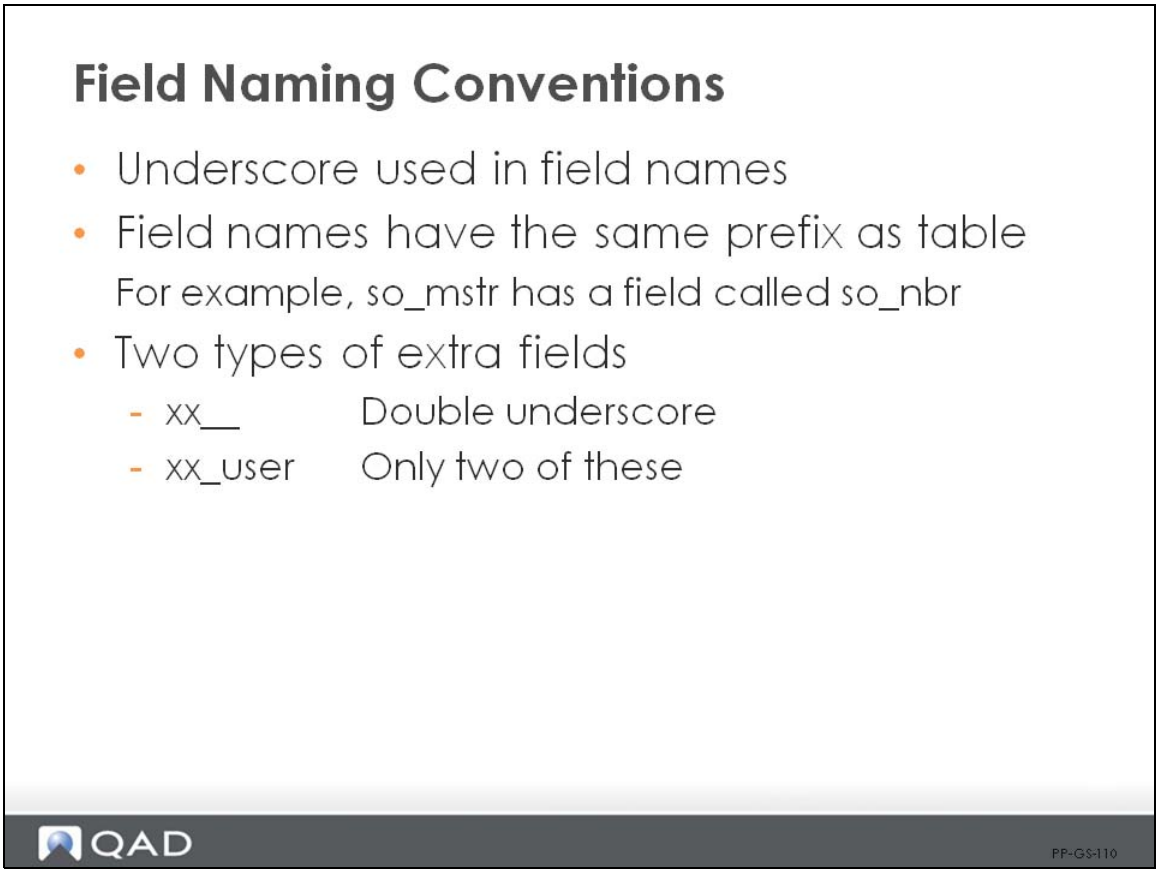
The following naming conventions are standard for operational tables in the QAD EA database.

- Underscores are used in table and field names, never hyphens.
- Master tables are named with a prefix (xx) followed by _mstr.
- The detail table associated with a master table is named `xxd_det`. Master tables generally contain static data that does not change from transaction to transaction; detail tables contain data that can change from transaction to transaction.
- Control tables are named `xxc_ctrl`. These tables contain settings that regulate how information is processed such as default values (base currency = USD), rules, parameters, or priorities that affect the entire system or specific modules. Security is often defined for control tables.
- History tables are named `xxh_hist`. They contain historical and audit information. The `ih_hist`, `ibh_hist`, and `idh_hist` tables contain invoice history.
- Internal software work files are named `xxx_wkfl`. The `usrw_wkfl` is intended for customizations.

Important The `qad_wkfl` is used by the QAD research and development staff and should never be used for customizations.

Note In QAD EE, the database tables that support component-based programs follow a different naming convention, as do the tables for the Warehouse Management System.

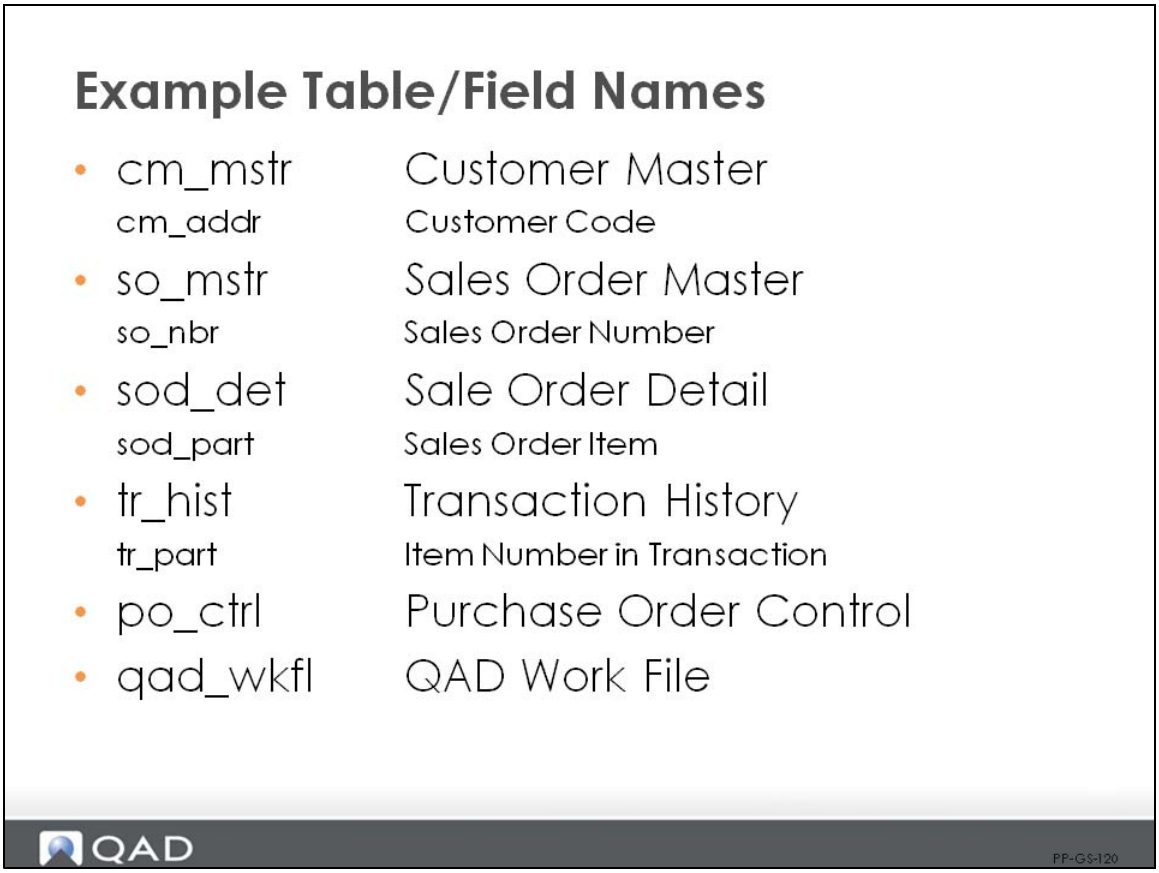
Field Naming Conventions



Several field naming conventions have been followed in the QAD EA database.

- Underscores are used in table and field names, never hyphens.
- Field names have the same prefix as the associated table name. All fields in the xx_mstr table begin with xx_; for example, the pt_part field in the pt_mstr.
- Each table includes two types of extra fields (xx represents the associated table prefix):
  - Double-underscore fields named xx__. Double-underscore fields of each data type exist in the tables, such as pt__chr01, pt__dec01, pt__log01. These fields are reserved for QAD use, and should not be used when developing custom codes.
  - User fields xx_user1 and xx_user2 are available for the development of custom programs, should the developer need to store extra data in the database.

Example Table/Field Names



This slide illustrates various examples of field and table names in the database.

Exercise 2-3: Naming Conventions

Using the Progress Data Dictionary, accessed as described in the previous exercise, or the QAD EA *Data Definitions* manual, complete the following exercises.

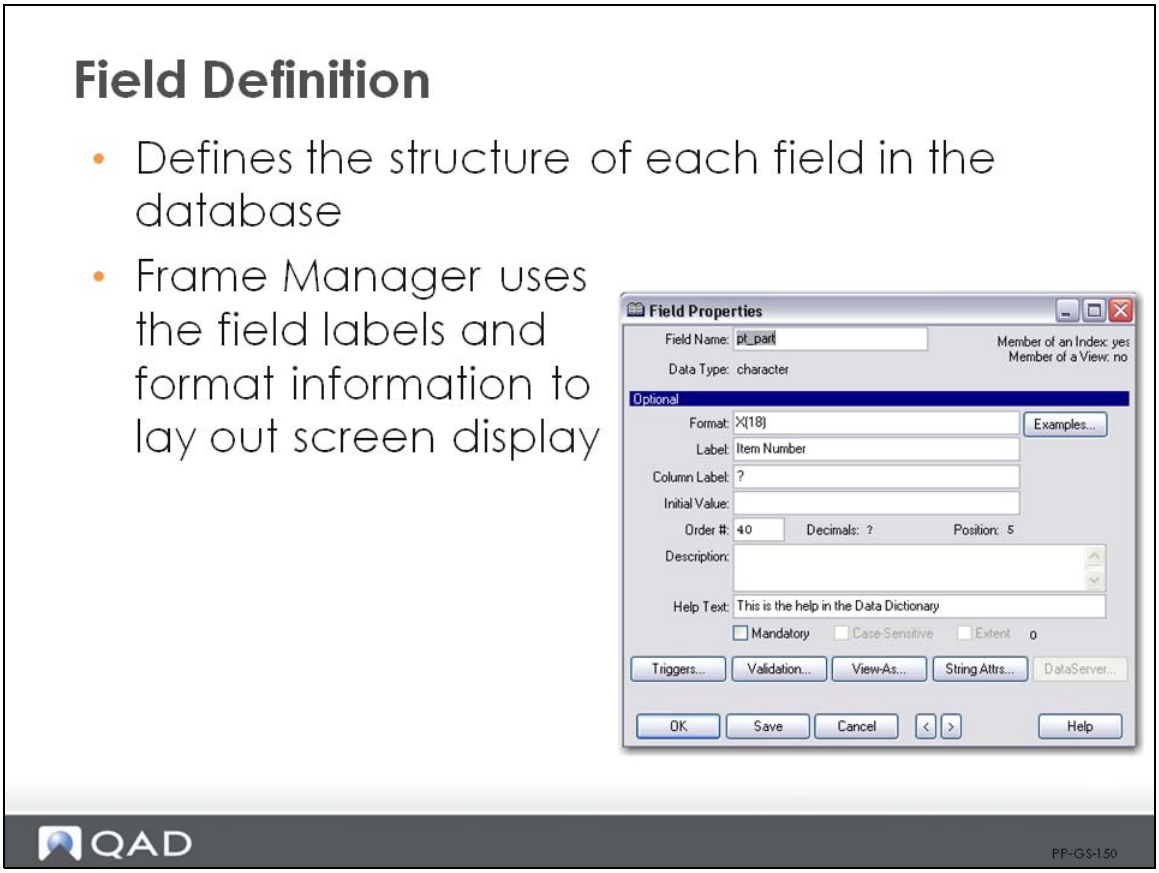
1 What is the database name of the following tables?

- Customer Master
- Sales Order Master
- Sales Order Control
- Inventory Transaction History
- Item Master

2 What is the database name of the following fields in the Item Master table?

- Item Number
- Description
- Revision
- Routing Code
- Group

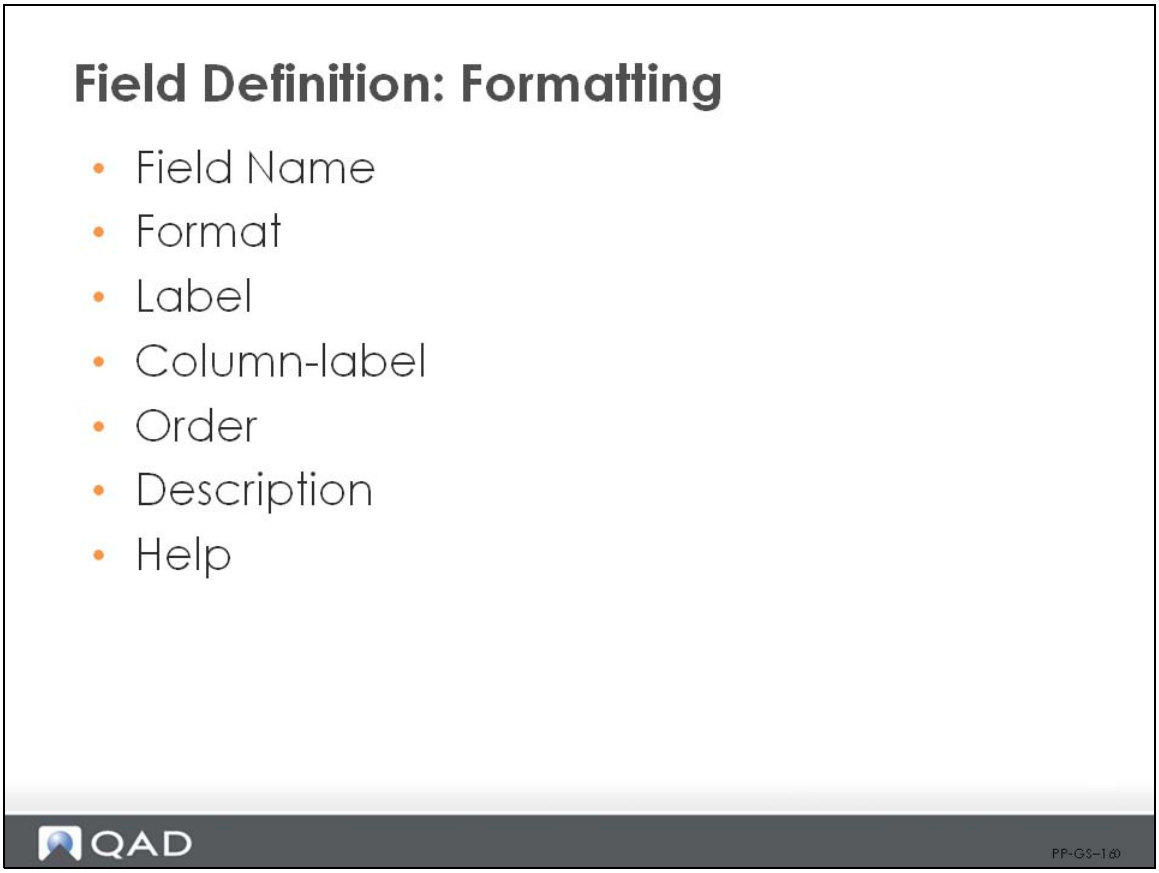
Field Definition



The Data Dictionary defines the structure of each field in the database. The Frame Manager uses the field labels and format information defined in the Data Dictionary to lay out screen displays.

Important The Database Manager uses the valexp information to validate data as it is entered in the user interface. The valexp has limitations and is not used by QAD. Instead, validation is managed in the code, not the schema.

Field Definition: Formatting



Screen formatting definitions include:

Field Name. Names can be up to 32 characters long and can consist of alphabetic characters, digits, and the characters $, &, #, %, –, and _. Field names must begin with A–Z or a–z and cannot be a Progress keyword.

Format. Describes the size and display characteristics of the field. Formats for decimal fields can include minus sign (–), period (.), or currency sign ($). Leading zeros are suppressed by using the greater than (>) sign. Trailing zeros are suppressed by using the less than (<) sign.

Note Format controls how the field displays on the screen; not the actual size of the data stored in the database.

Label. Determines the default label used when the field is displayed.

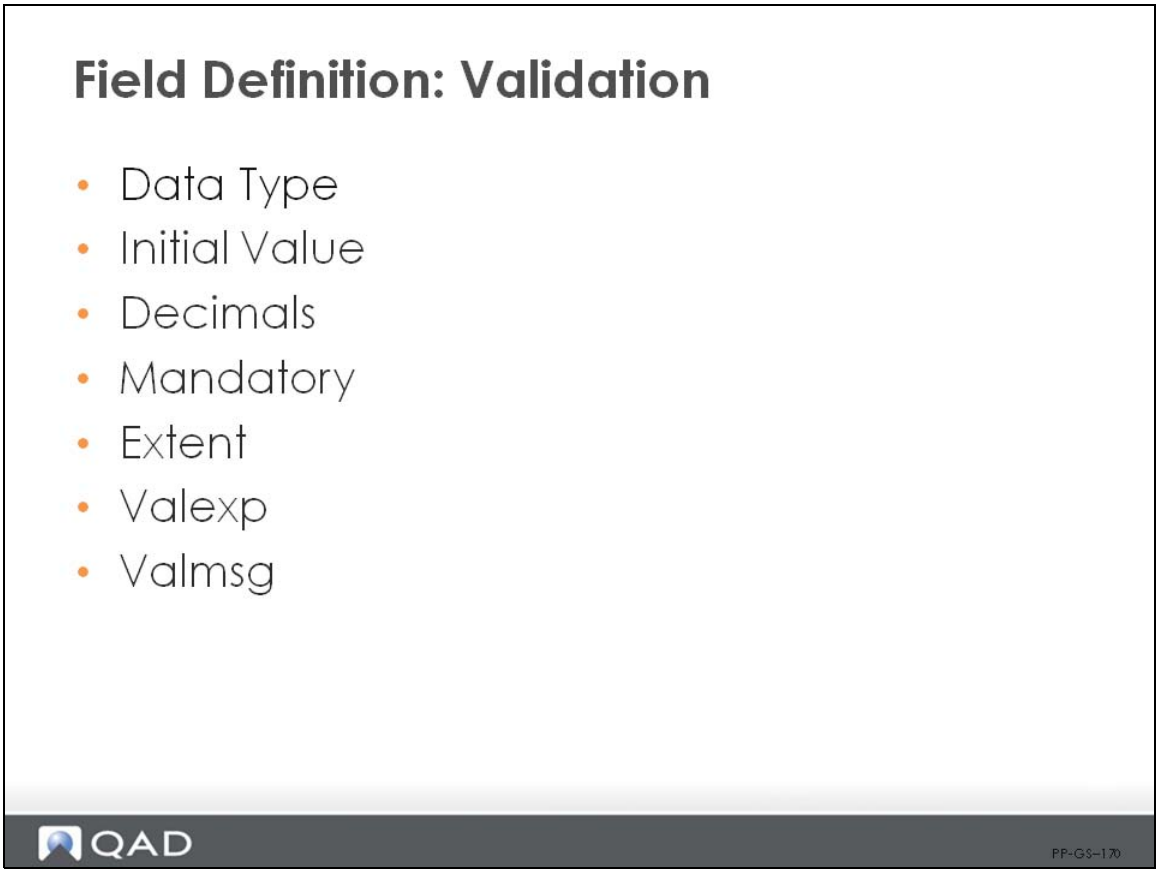
Column-label. The default label used when the field is displayed in columnar format. Most QAD EA reports and inquiries are in columnar format.

Order. Determines the default order in which Progress display the fields. This is also the order in which the fields are exported during a database dump/load.

Description. Provides a more detailed explanation of the information that the field holds.

Help. Text describing the field to be displayed to users. QAD does not use this text. Field and program help in QAD applications comes from records loaded into a help database and retrieved using a standard program.

Field Definition: Validation



Validation information includes:

Data Type. Verifies the type of data entered. Common data types are character, date, decimal, integer, and logical.

Initial Value. The default initial value for the field. This is the only field definition that can be changed without requiring that programs be recompiled for changes to take effect.

Decimals. Maximum number of decimals stored in the field. The number of decimals stored can differ from the number defined in the format. Most decimal fields in QAD EA store up to 10 decimals.

Mandatory. Indicates if input in the field is required.

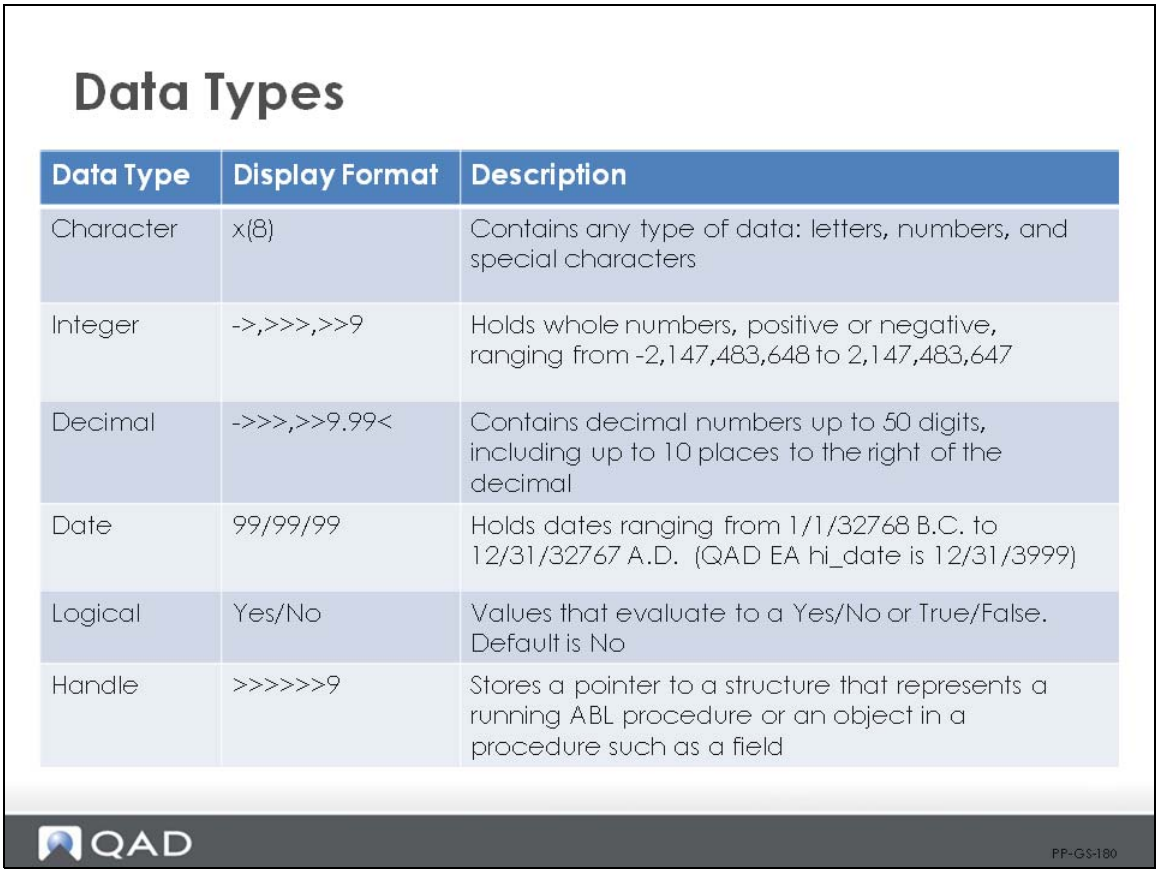
Extent. For array fields, specifies the number of elements in the array. An array contains a collection of elements, each selected by one or more indexes that can be computed at run time. See “Array Processing” on page 178.

Valexp. Contains any validation information for the field. Data entered for this field is tested against this validation expression and is accepted if it passes. The validation expression must evaluate to True/False. In QAD EA, generalized codes and field security validation are defined for some operational tables using this format, but its application is very limited.

Valmsg. The message displayed when invalid data is entered.

Note Valexp and valmsg are not used in QAD coding.

Data Types



As mentioned in the previous slide, Progress supports the six main data types listed here: character, integer, decimal, date, logical, and handle.

Progress OpenEdge 10 introduced a few new, more advanced types for 64-bit management, managing large amounts of data, and more sophisticated date and time tracking. You can learn more about them in the Progress documentation.

INT64. A64-bit integer data type, similar to the standard integer (32-bit).

BLOB. A binary large object (BLOB) supports storage of large binary data (up to 1 GB) that is typically managed by external programs such as PDFs and word processing files.

CLOB. A character large object (CLOB) is distinguished from a BLOB in that it contains only character data (up to 1 GB) and has a code page associated with it. CLOBs are manipulated by copying them to a LONGCHAR variable.

LONGCHAR. The LONGCHAR data type is used for managing character data that is not limited in size to 32K, but rather is limited in size by system resources.

DATETIME. The DATETIME data type consists of two parts, date and time. The unit of time is milliseconds from midnight.

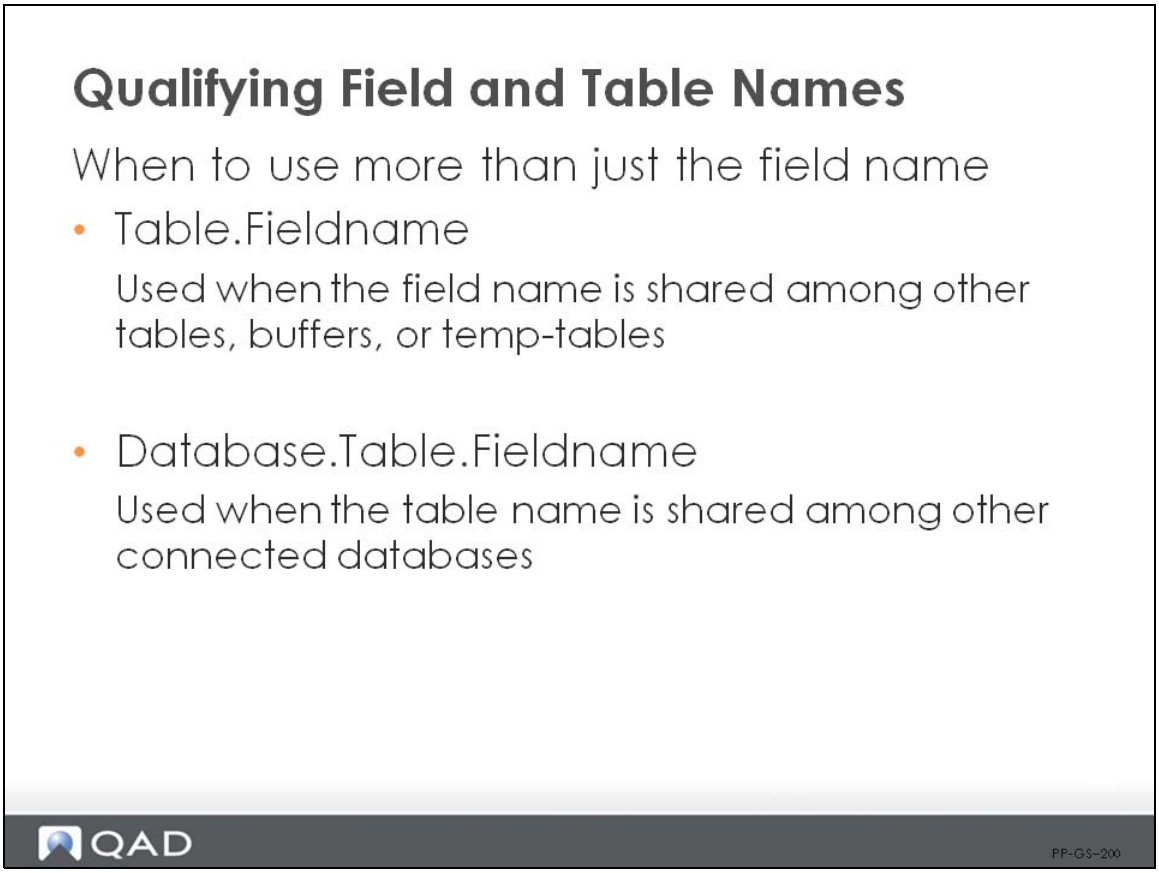
DATETIME-TZ. The DATETIME-TZ data type consists of three parts: date and time (as for DATETIME), and an integer representing the time zone offset from Coordinated Universal Time (UTC) in minutes.

Exercise 2-4: Field Definitions

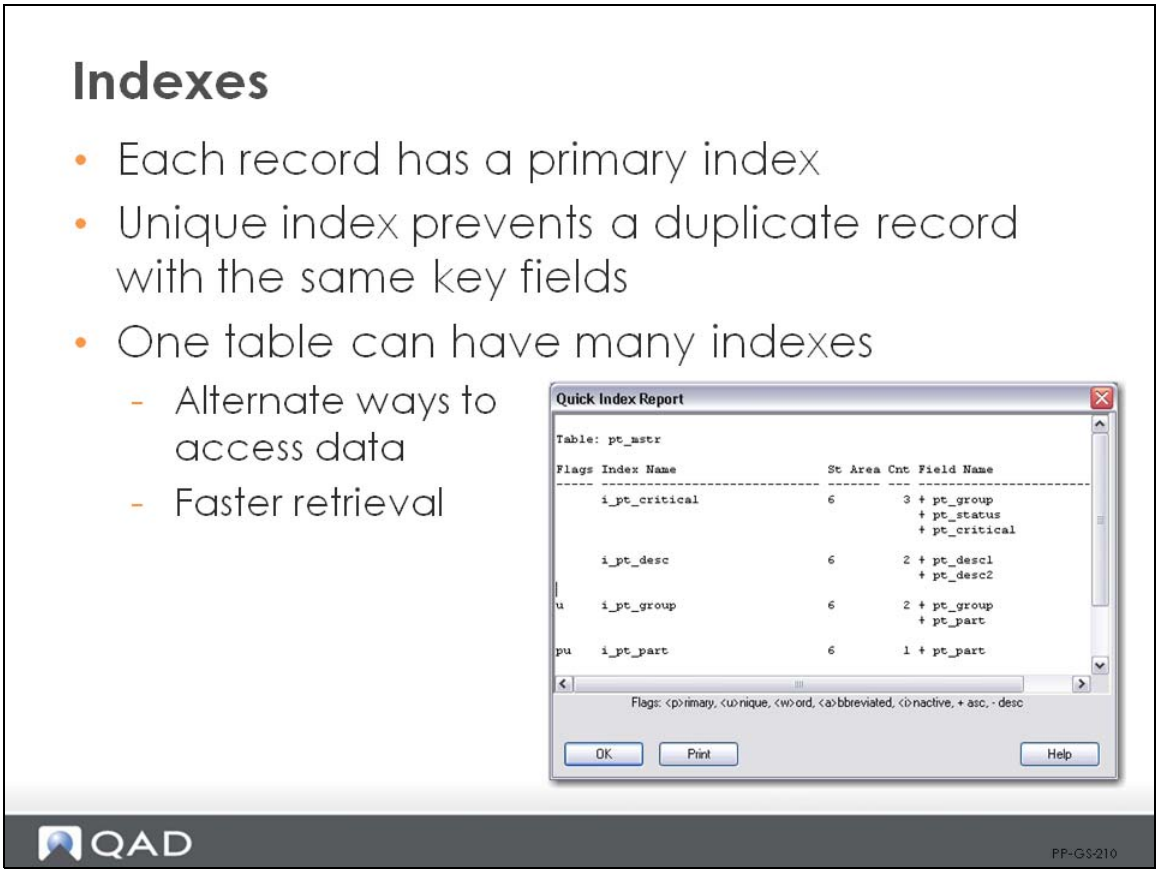
Use the field definitions from the Progress data dictionary, or the QAD EA *Data Definitions* manual, to determine the following information about the fields in the Item Master (pt_mstr):

1. How big can an item number be?
2. Can an item number only contain numbers?
3. How many decimals are allowed in the price?
4. Is there any restriction on the value for the item's product line?
5. What values can the Inspection Required field have?
6. What is the initial value of Issue Policy?

Qualifying Field and Table Names



Indexes



An index is like a labeled tab on a file folder. When you want to look up something in your files, you first select the right filing cabinet (table) and then use the index tabs to select the correct file folder (record). This is much faster than removing the folders one at time and scanning the contents to see if you have the folder you want.

Progress looks up information in the same way, accessing specific records by looking at the indexes rather than looking at every record individually.

Each record is defined with a primary index (for example, item number). A unique index indicates that only one file folder or record has that index. The item number is defined as a unique index. This prevents a duplicate item from being added to the database.

Records can be indexed in ascending or descending order. By default, records appear on reports and inquiries in the same order they appear in the primary index.

Progress lets you use alternate indexes (for example, numbers instead of letters). These provide alternate ways to access your data and help you get to it faster.

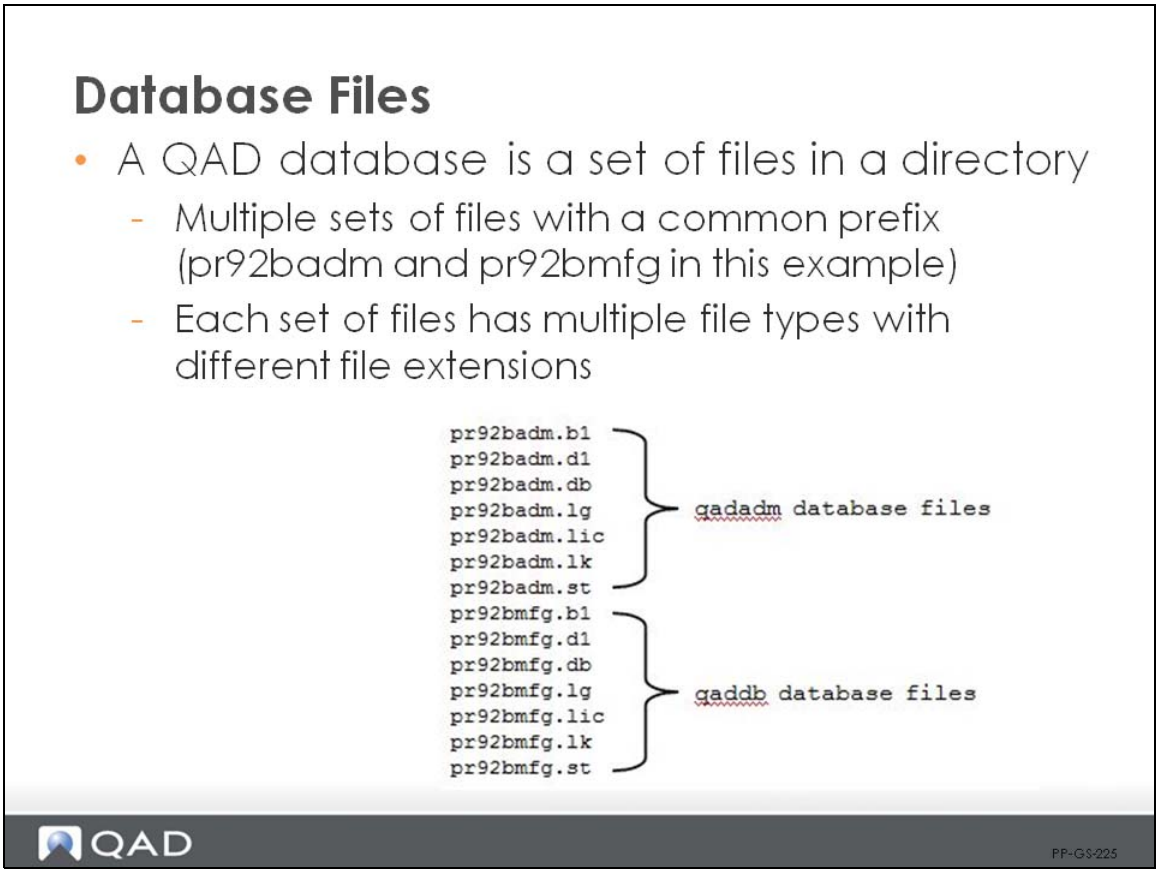
Exercise 2-5: Indexes

1 In what order are Sales Order Master records displayed by default?

Hint. What would sales orders logically be ordered by?

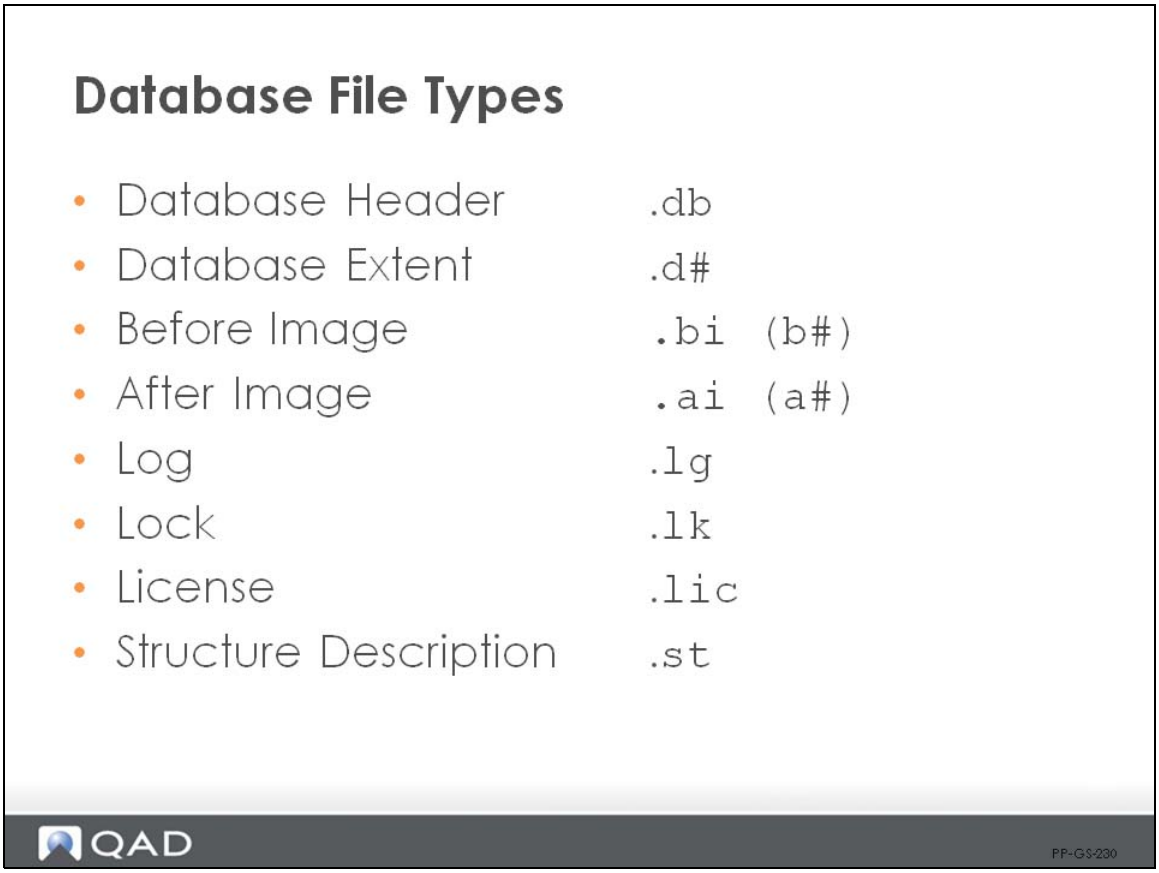
2 In what order are the Item Master records displayed by default?

Database Files



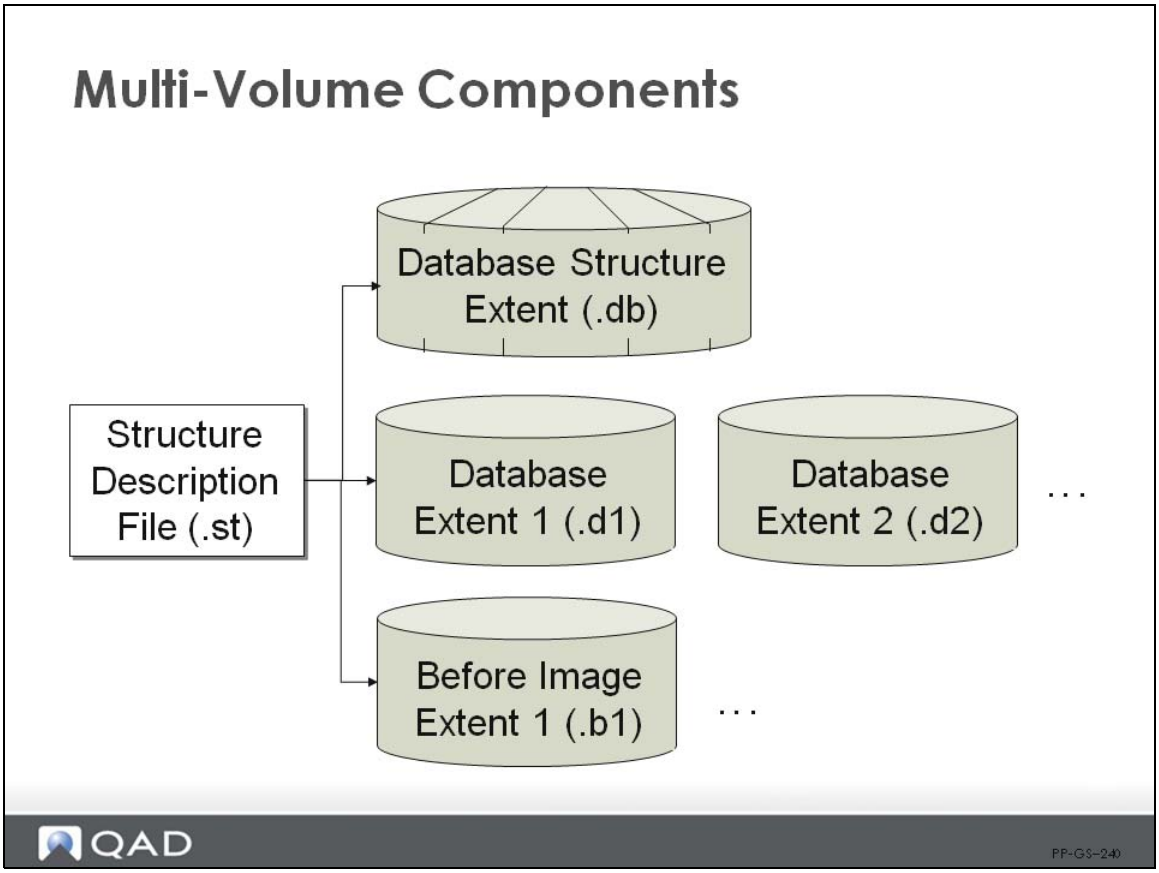
A QAD database consists of files in an operating system directory. All of the files that are part of one database have the same prefix. A complete database is a set of three individual databases: qadadm (administrative data), qaddb (core application data), qadhlp (help records).

Database File Types



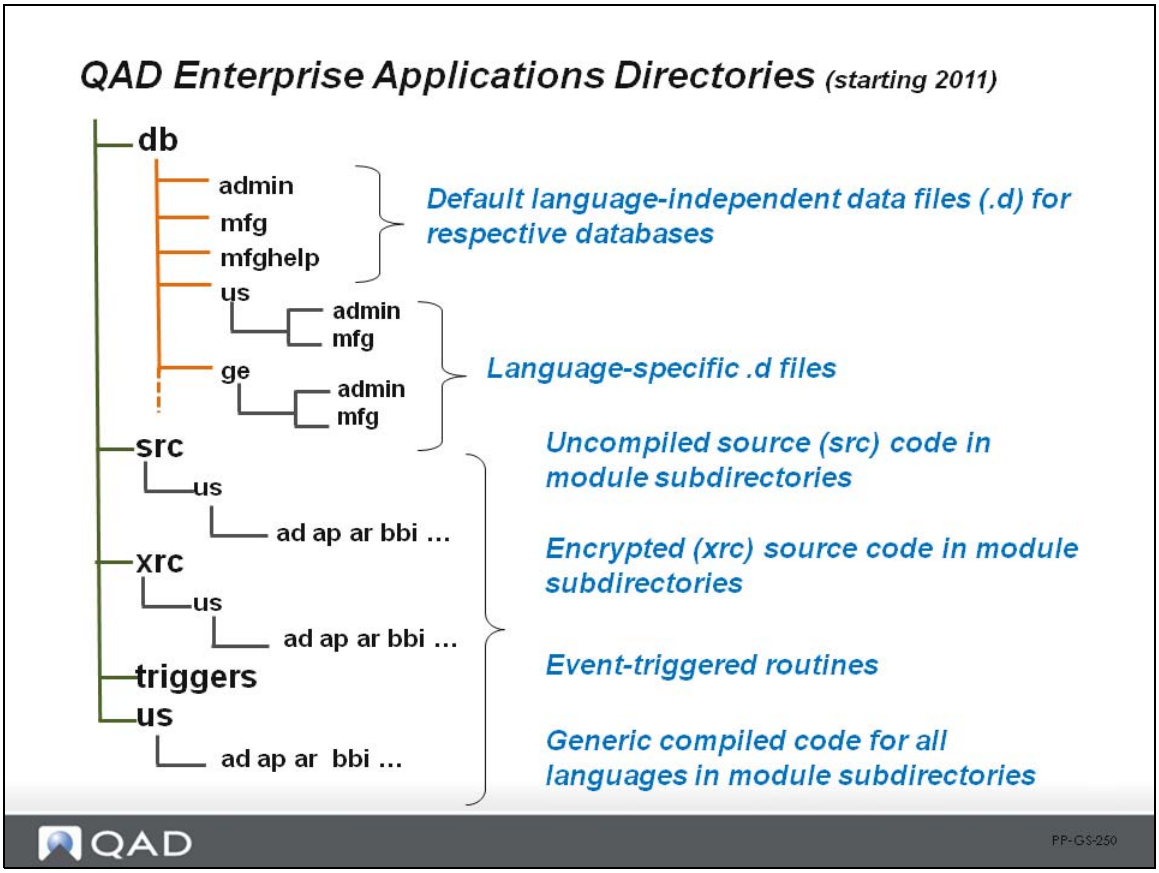
File Type	Used for...
.d	Data file loaded into one or more tables in the database
.db	Single volume database file or road map to multi-volume database files and information
.d#	Database extent number (dbname.d5)
.bi/.b#	The Before Image file or extent number file; maintains snapshot to support rollback of non-committed transaction
.ai/.a#	The After Image file or extent number file for transaction rollback
.lg	Reporting log of database activity
.lk	The existence of this file indicates the database is being used in update mode
.lic	Keeps track of number of user logins
.st	A map for creating a database

Multi-Volume Components



The structure description file (`.st`) is the instructional road map for putting the physical files onto the disks during database creation.

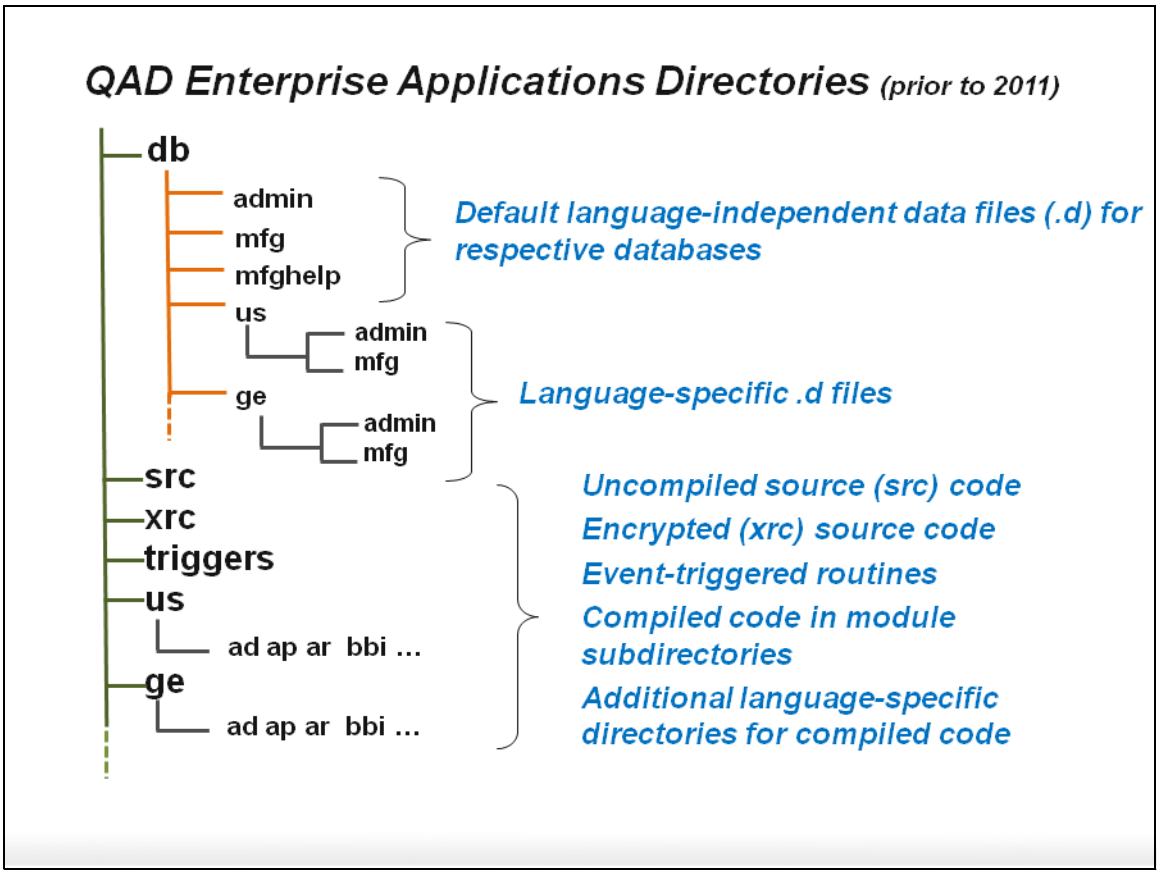
QAD Enterprise Applications Directories



This slide illustrates the layout of files in a QAD EA installation beginning with QAD EA 2011. The structure differs in two ways from earlier installations:

- Language-specific source directories are no longer used; all compiled code is under the `us` directory.
- Compiled code is now located in two-letter module code subdirectories, just like the source and encrypted code.

Directory	Description
`db`	This directory contains files related to building the product database, which actually consists of a set of databases. In addition to general data files, language-specific data files are supplied in two-letter language code directories.
`src/us`	This directory contains unencrypted source code in two-letter subdirectories corresponding to two-letter product module codes. For example, the `ad` directory has code related to addresses and `ap` has code related to Accounts Payable. All customers receive unencrypted source for reports and inquiries; full source code must be specifically licensed.
`xrc/us`	This directory contains encrypted source code mirroring the same structure as the src directory.
`triggers`	This directory contains trigger programs (.t) that execute based on database events (such as read, write, or delete).
`bbi`	This directory contains include files used in browses and other programs that are compiled when needed.
`us`	This directory contains all of the compiled code in the identical two-letter directories as src and xrc.



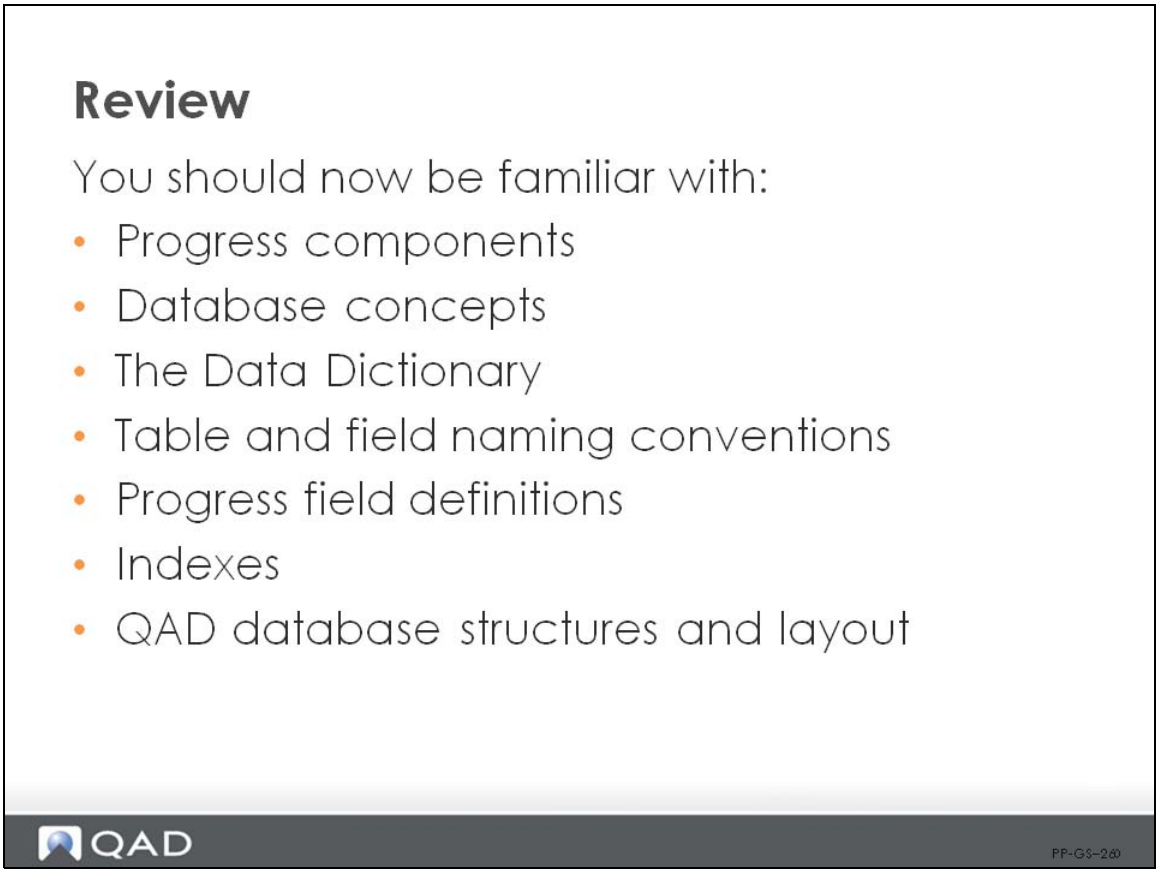
This slide illustrates the layout of files in a QAD EA installation prior to QAD EA 2011.

Important This code has an important implication when referencing other files in a program. With the new directory structure all references to include files and subprocedures must be prefixed with `us/<2_letter_subdirectory>/`

The include files listed here are found in the `bbi` subdirectory, not in a directory that matches the first two letters of the include file name.

attribut.i	gpappend.i	gplabel.i	lookup.i	pxmsglib.i
base.i	gpbrwtt.i	gplblfmt.i	menu.i	query.i
bprint.i	gpdefvar.i	gpmkrcta.i	mf1.i	stack.i
browse.i	gpfield.i	gpmkrect.i	mfdecgui.i	uibsetup.i
brprhd.i	gpfieldv.i	gpmnsave.i	mfdeclre.i	wbadmweb.i
buttnbar.i	gpfildel.i	gpmsgdf.i	mfdecweb.i	wbbr01.i
containr.i	gpfile.i	gpmsgob1.i	mfmsg.i	wbgp02.i
contents.bbi	gpfilev.i	gpmsgob2.i	mfphead2.i	wbgp03.i
cxcustom.i	gpfilfld.i	gpreturn.i	mfphead.i	wbgp05.i
dataf.i	gpinvoke.i	gprptdef.i	mfquoteo.i	wbgpms01.i
dataio.i	gpiswrap.i	gprun1.i	mfreset.i	wbpp01.i
dfattrib.i		gprun.i	mgdomain.i	whdeclre.i
encrypt.bbi		gpstrip.i	pxgblmgr.i	window.i
eudeclre.i		initobj.i	pxmsg.i	wocapl.i

Review


Review
You should now be familiar with:
• Progress components
• Database concepts
• The Data Dictionary
• Table and field naming conventions
• Progress field definitions
• Indexes
• QAD database structures and layout
QAD
PP-GS-260

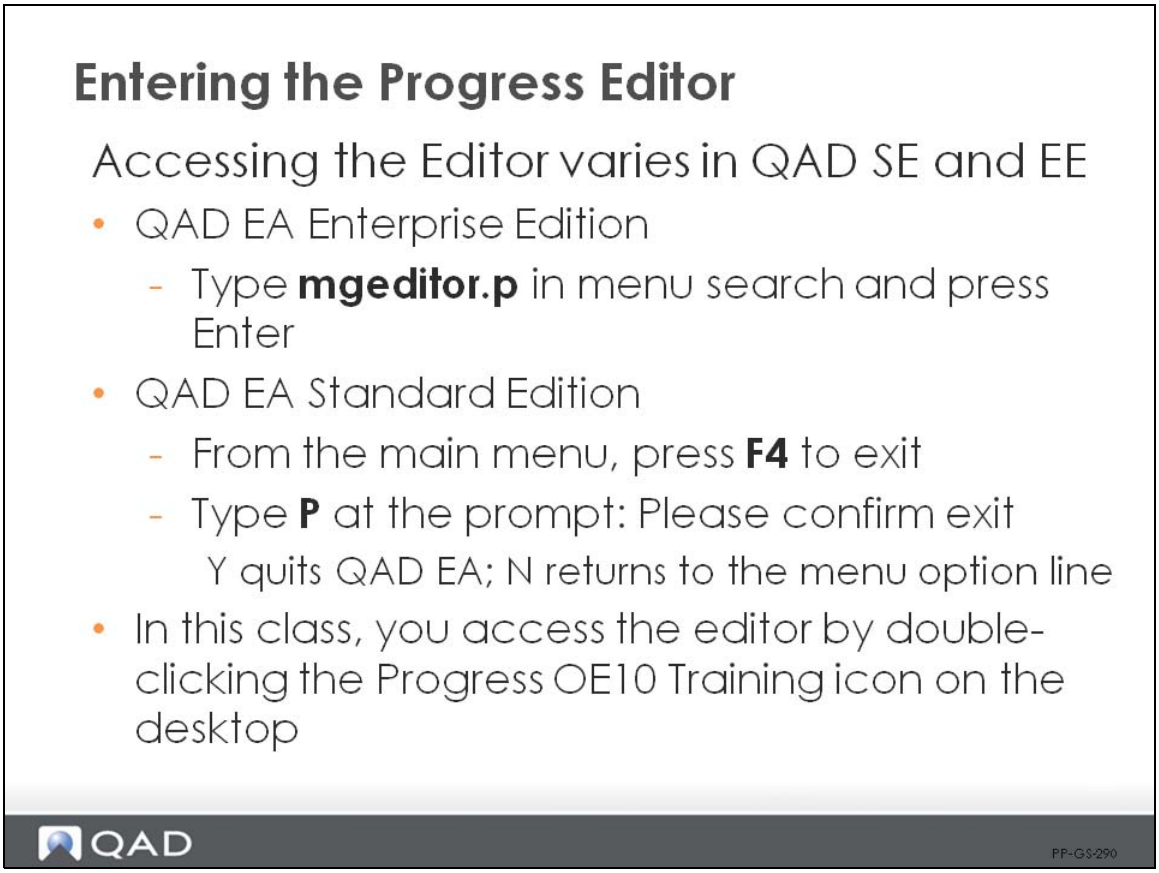
Lesson 3: Using the Progress Editor

One of the major Progress components is the Progress Editor. You use this editor to write and modify your Progress procedures.

Upon completion of this lesson, you will be familiar with the following topics:

- Entering the Progress Editor
- Saving and retrieving a Progress file
- Reviewing error messages
- Copying/pasting/deleting text
- Using the Editing menu
- Executing Progress procedures
- Leaving the Progress Editor

Entering the Progress Editor



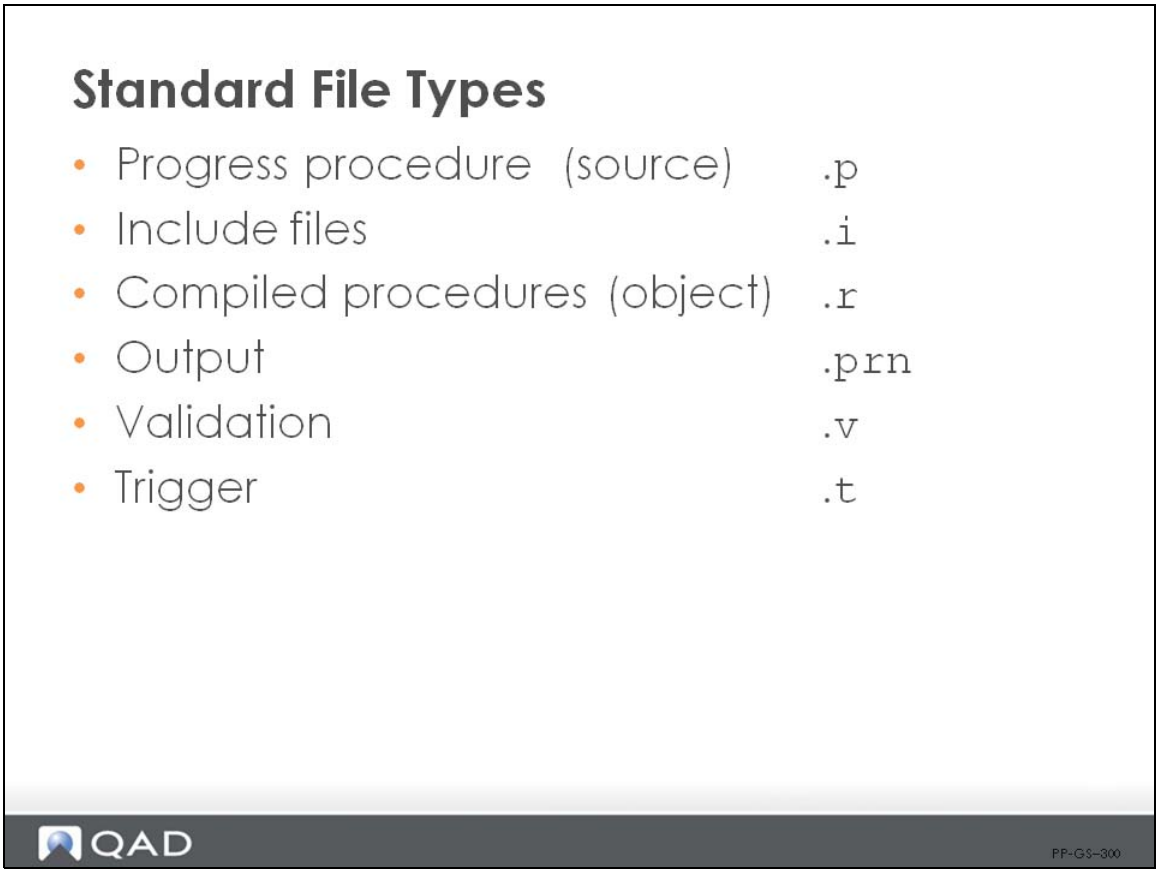
The Progress character editor is a full-screen editor without many of the features of a standard Windows editor such as Notepad. This means that you cannot use the arrow keys or the mouse to position yourself on the screen. The entry area automatically scrolls as you enter more lines. This is the editor you would use if Progress is on a Linux/UNIX platform.

The Progress GUI editor is a full-screen editor with the same features as a standard Windows editor such as Notepad. This is the editor you will use in class.

Press Enter or Return to advance to the next line.

Keyboards vary and how each of the keys is mapped in Progress varies from terminal to terminal.

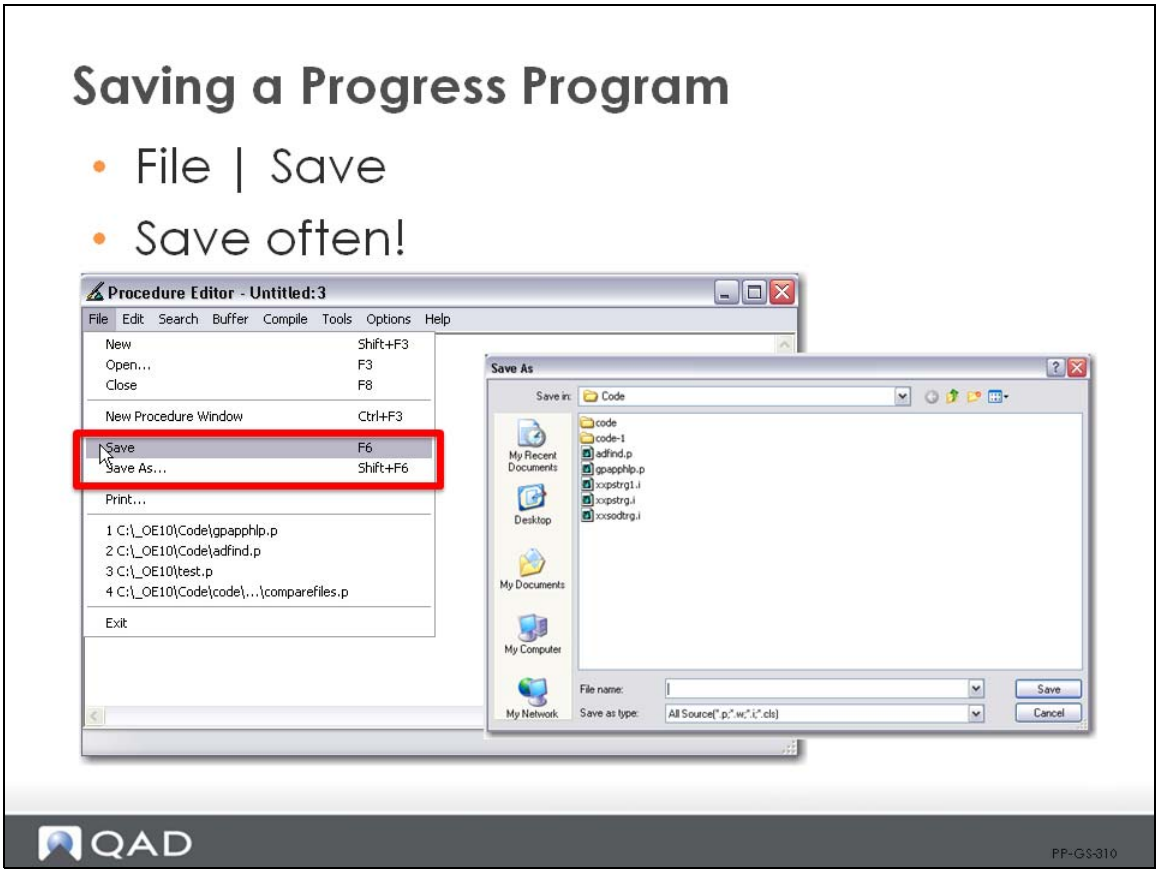
Standard File Types



The various types of files used in Progress programming are distinguished by different file endings. Each type is covered in more detail later in the course. For now, you will create and use Progress procedure (.p) files.

File Type	Description
.p	Uncompiled source code
.i	Include files, which are incomplete code segments used in multiple other files
.r	Compiled code
.prn	Report output file, text format
.v	Field testing and validation (not used by QAD); validation files are a type of include file
.t	Uncompiled source code files associated with database tables through the data dictionary and invoked when a table event occurs such as create, write, or delete

Saving a Progress Program

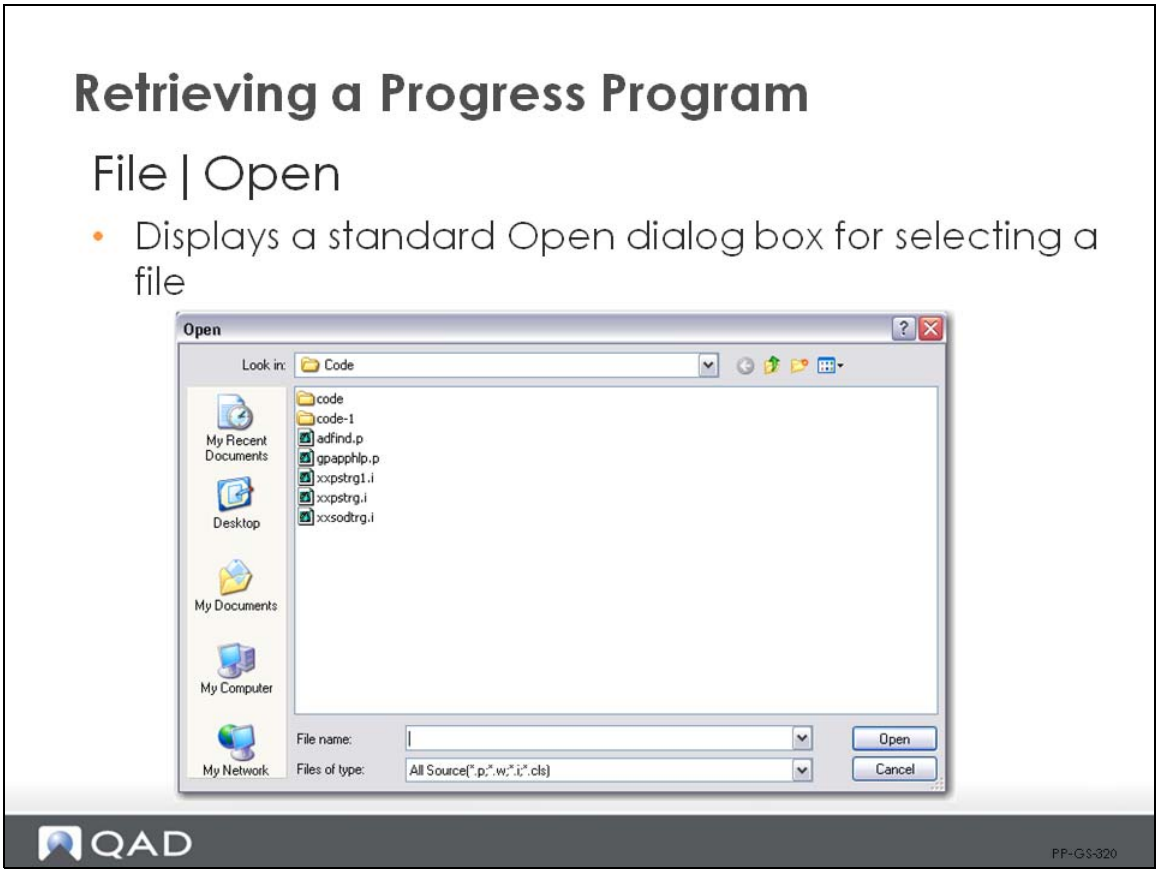


The Editing menu is displayed along the top of the screen. Each option has a pull-down menu that you can access using the relevant accelerator key. To save a file or program in Progress:

- Select File|Save to save using the current file name.
- Select File|Save as to save the file with a different name.

A Save dialog box appears as shown above. Enter the file name at the prompt. Remember to include the .p extension and press Return.

Retrieving a Progress Program



To Retrieve a file or program in Progress, select File|Open. The Open dialog box appears as shown above. Type in the file name at the prompt, remembering to include the .p extension, and press Return or select the file from the list.

Exercise 3-1: Using the Editor

1 Use the Editor to type the following procedure.

```
FOR EACH pt_mstr WHERE pt_domain = "10USA" NO-LOCK:
   DISPLAY pt_prt ppt_desc1.
END.
```

2 Save this procedure.

Hint. Use the File menu and select Save. Use your first name as the program file name, and make sure the file extension is .p, the normal extension for a Progress procedure.

3 Clear the Screen by selecting the File menu option and then Close.

4 Retrieve the same program. Use the File menu and then Open.

5 Modify the procedure as follows:

From: `pt_prt`
To: `pt_part`

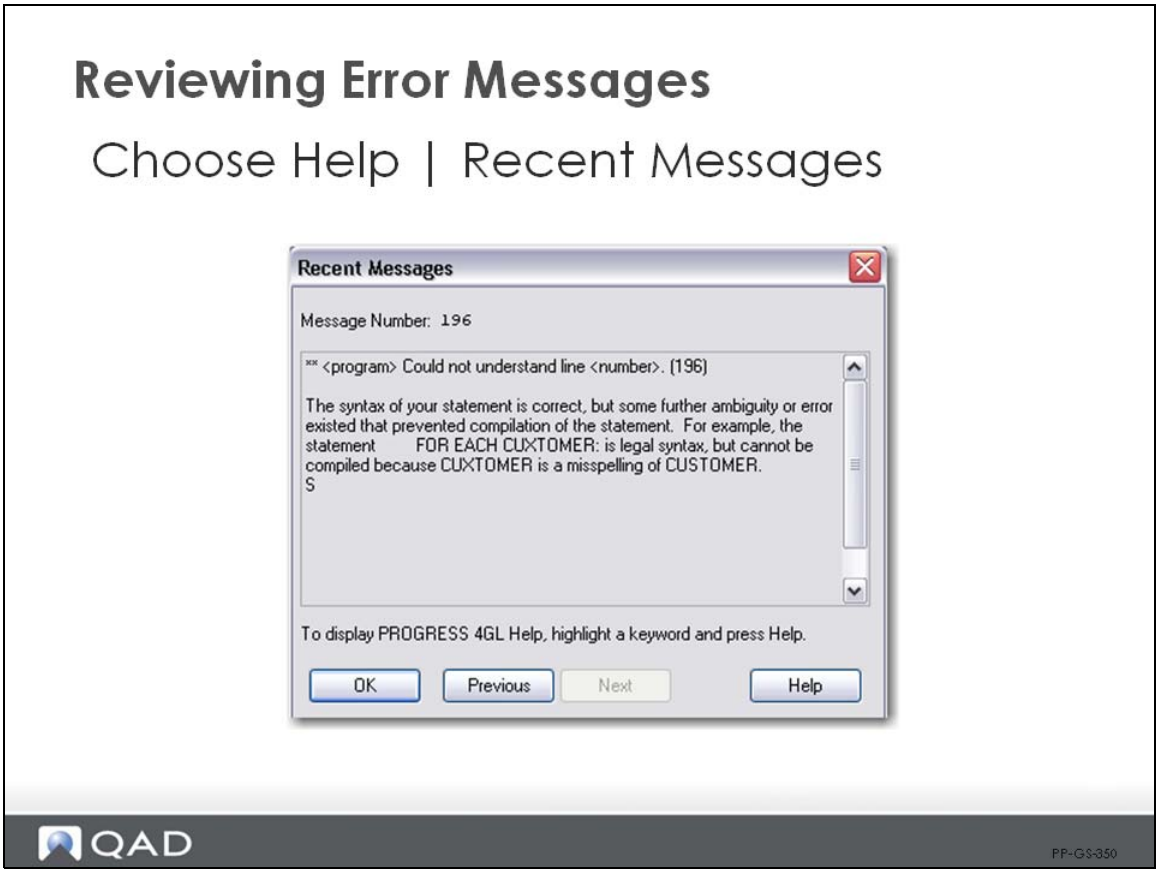
6 Clear the screen. **Do not save the program**. Select **No** from the Save option dialog.

Hint. Attempt to clear the screen by selecting the File menu option and then Close.

7 Retrieve same program again. Was the modification saved?

Hint. Retrieve the same program back onto the screen. Ensure that the last modification was not saved.

Reviewing Error Messages



When a program is compiled, Progress checks for errors in the code. If it finds an error, an error message and error number are displayed. You can view errors that occurred in the Progress session by selecting Help|Recent Messages. You can also view Progress errors by accessing Help|Messages and identifying the message number.

Exercise 3-2: Error Messages

1 Retrieve the program created in the previous exercise.

File|Open

2 Run the program by choosing Compile|Run (F2 in GUI, F1 in character). This should produce an error.

3 Modify the procedure as shown here and run.

From: `ppt_desc1.`
To: `pt_desc1.`

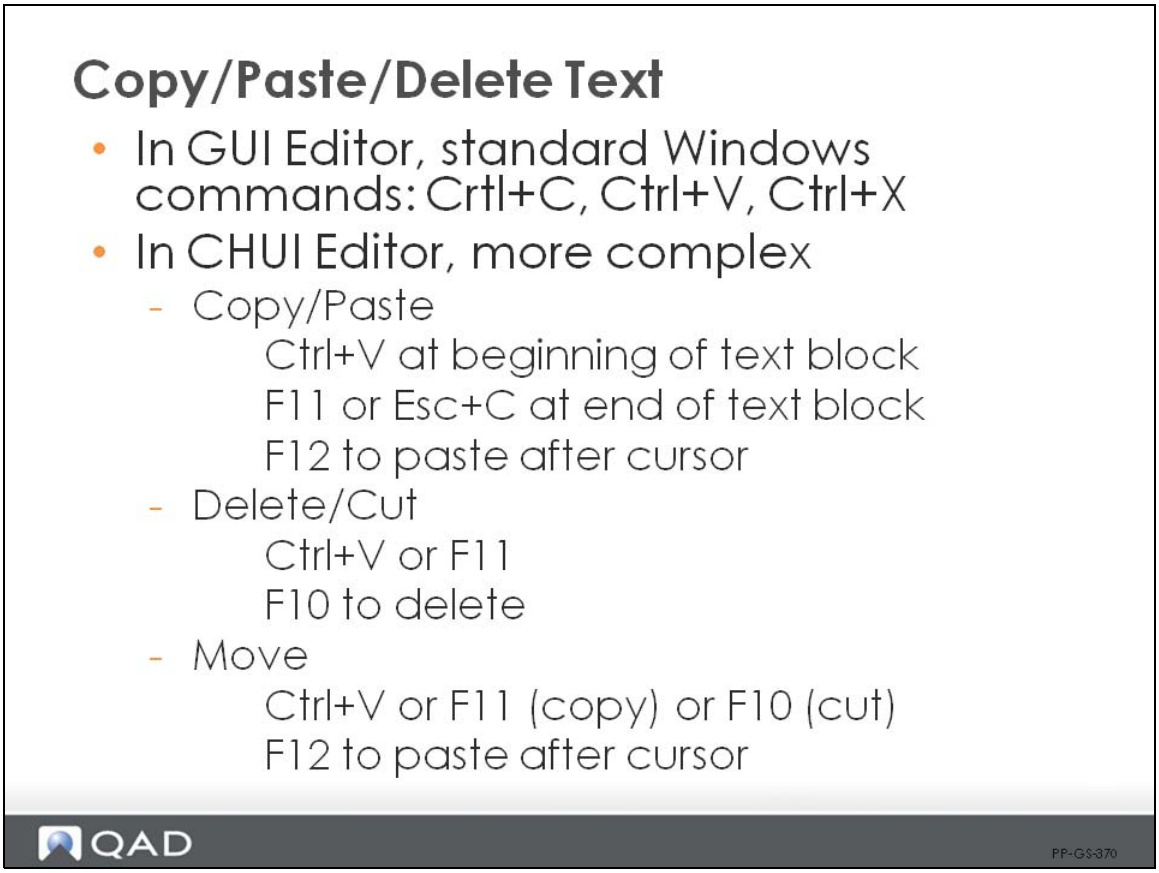
4 Review the messages by selecting Help|Recent Messages.

5 While in the Progress Help, explore the Messages option and the information it provides.

6 Correct any error in your program and rerun it.

7 Save your program under its current name by using File|Save.

Copy/Paste/Delete Text



Manipulating Text in the GUI Editor

If you are using the GUI Editor, you can use standard Windows editing commands for manipulating text:

- To copy, highlight the text with the mouse and press Ctrl+C.
- To paste, highlight the text with the mouse and press Ctrl+V.
- To delete, highlight the text with the mouse and press Ctrl+X.

Manipulating Text in the Character Editor

If you are using the character-based Editor, use the following guidelines for text operations.

Note Terminal key mapping may affect the way keys work on your particular computer.

Copy

1. Go to the beginning of the block of text that you want to copy and press Ctrl+V. This indicates that the text to copy starts here.
2. Go to the end of the block of text and press ESC+C to copy the block. The text is not highlighted but it is copied.

3 Go to where you want to paste the text and press F12. The block of text is inserted after that position.

Copy Portion as a New File

1 To copy a portion of code and save it as a new file, first select the block of code you want to copy, as described in Copy.

2 Open a new file by selecting File|New.

3 Select Edit|Paste to retrieve the block from memory.

4 Save the copied code into a new file name by selecting File|Save As.

Delete

1 To delete a portion of code, go to the beginning of the block of text you want to delete and press Ctrl+V.

2 To delete the block of code, move your cursor to the end of the block and press F10.

3 To delete a whole line of code, press Ctrl+D anywhere in that line.

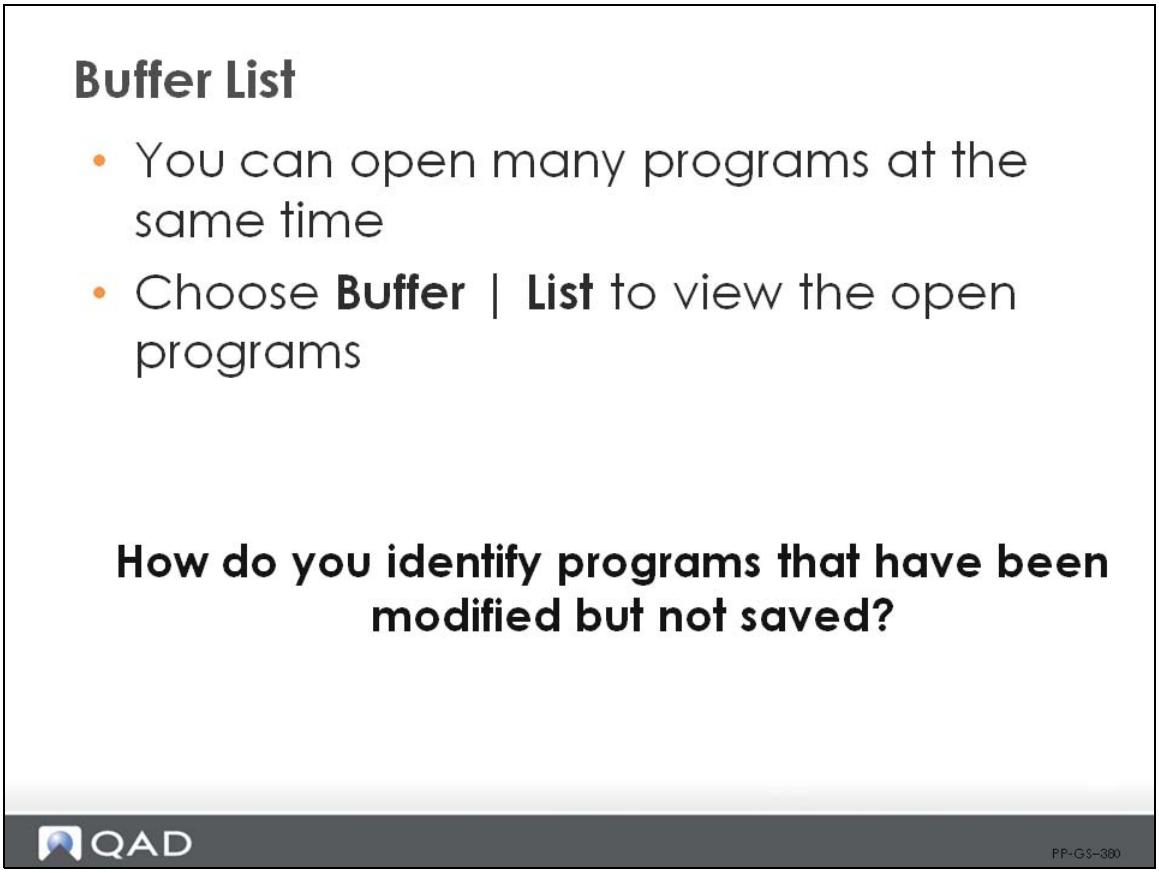
Move

1 Go to the beginning of the block of text that you want to move and press Ctrl+V. This indicates that the text to be moved starts here.

2 Go to the end of the block of text and press F10 to cut the block.

3 Go to where you want to paste the text and press F12. The block of text is inserted after that position.

Buffer List

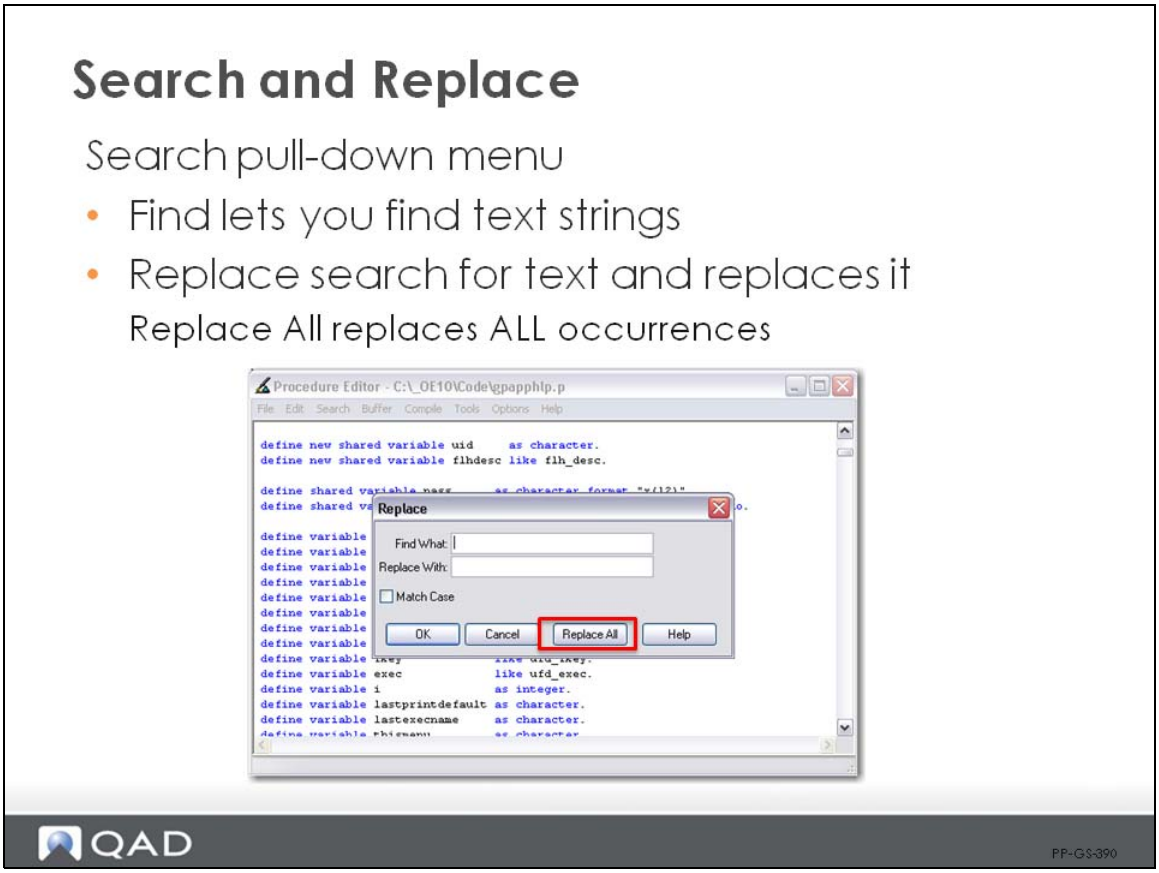


The Progress Editor creates buffers when you use the New or Open option. A buffer is a temporary area where data is stored while you view and modify it.

To view a list of the open programs for your current session, choose Buffer|List. The open buffers are listed in the Buffer List dialog box in the order that you opened them.

To display information about the current buffer, choose Buffer|Information. The Buffer Information dialog box shows the file name; read and write access for the file; the number of lines, columns, and bytes; and the modified status of the buffer.

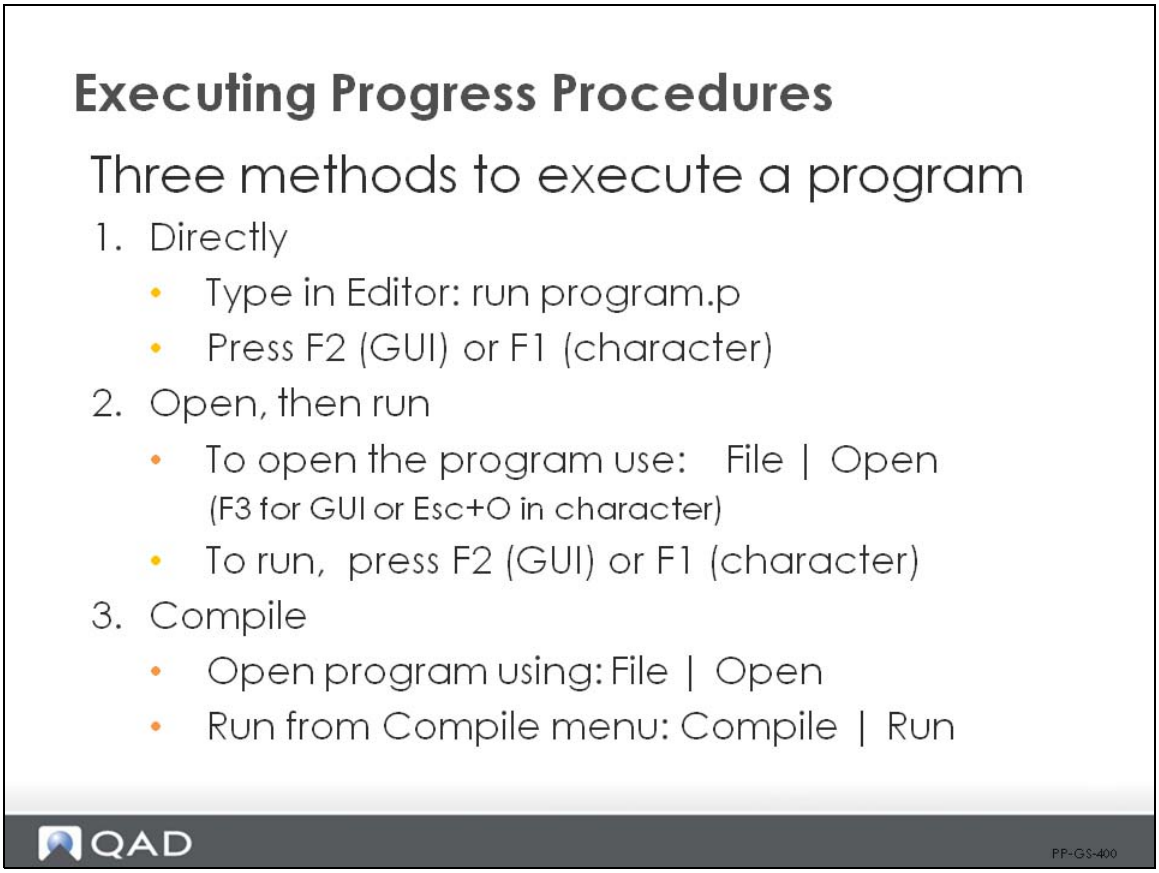
Search and Replace



The Search pull-down menu offers several text search options. Find, Find Next, and Find Previous lets you find text strings. Replace lets you search for text and replace it or skip it. Goto Line places the cursor on a specified line number.

Selecting **Replace All** replaces all occurrences of the text above and below the current position in the text.

Executing Progress Procedures

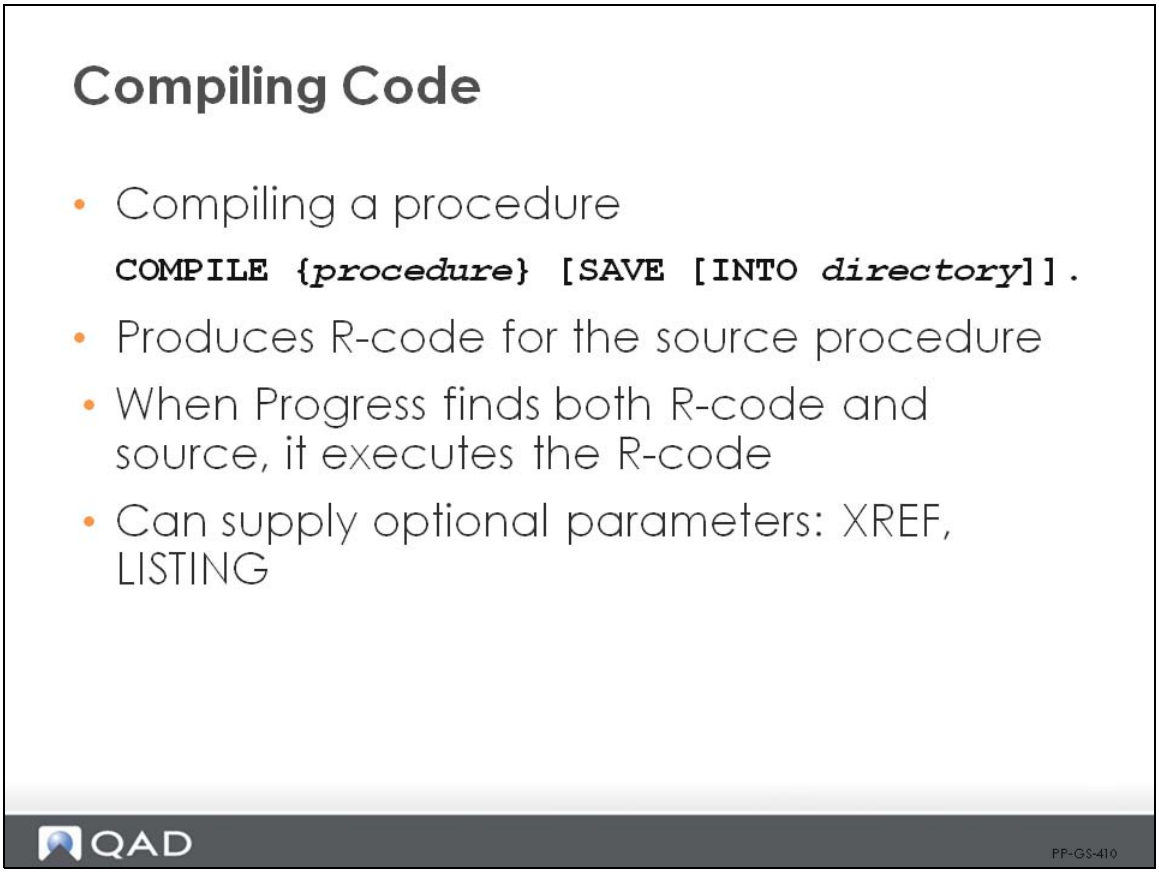


When you execute a procedure, Progress compiles the procedure lines; that is, it checks for typing errors or invalid syntax. If you have made errors, Progress displays an error message, and puts the cursor on the line containing the error. You should correct the error and rerun the procedure.

When Progress encounters an error, you can review a more detailed explanation of that error in Help|Recent Messages.

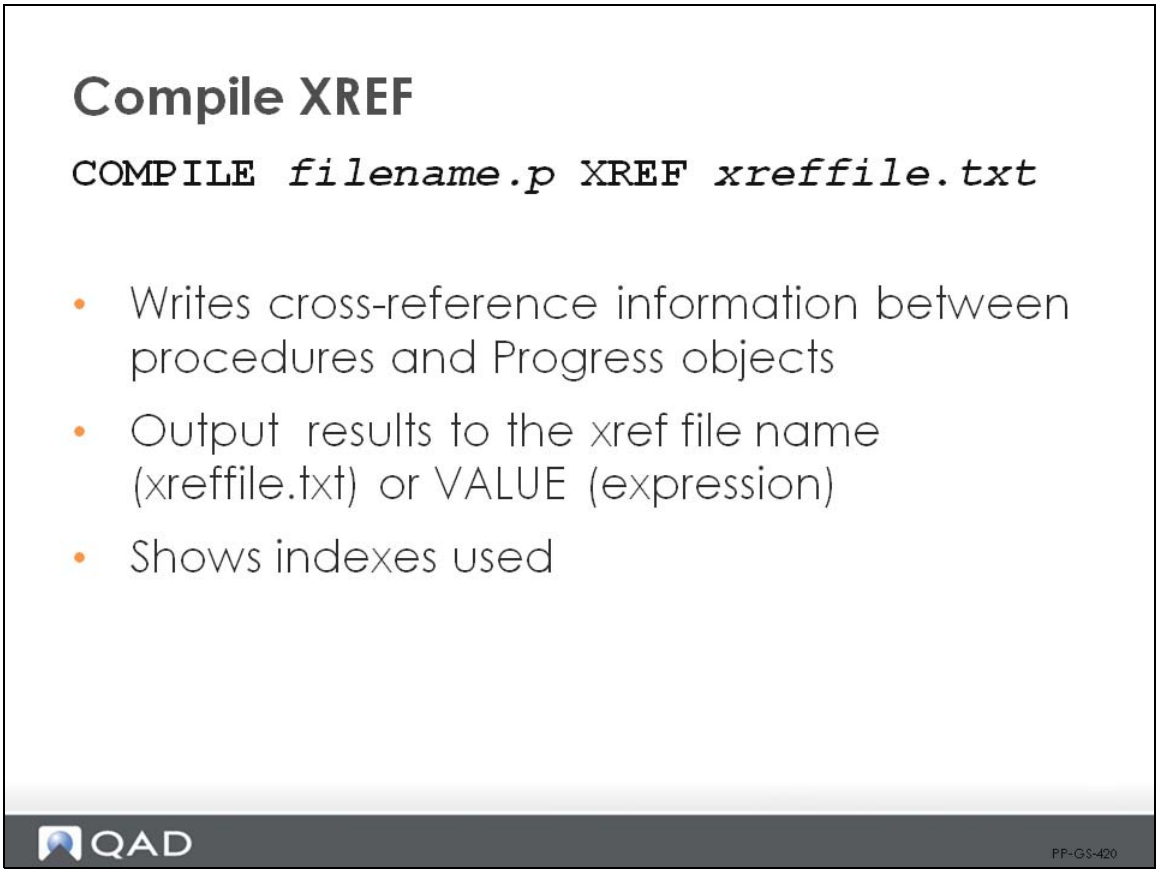
You can check the code syntax before you run the code by selecting Compile|Check Syntax.

Compiling Code



To compile a procedure that you have saved, you use the COMPILE statement, which has a number of options. For example, you can specify where the compiled code is saved and whether a cross-reference file is generated to help you debug your application.

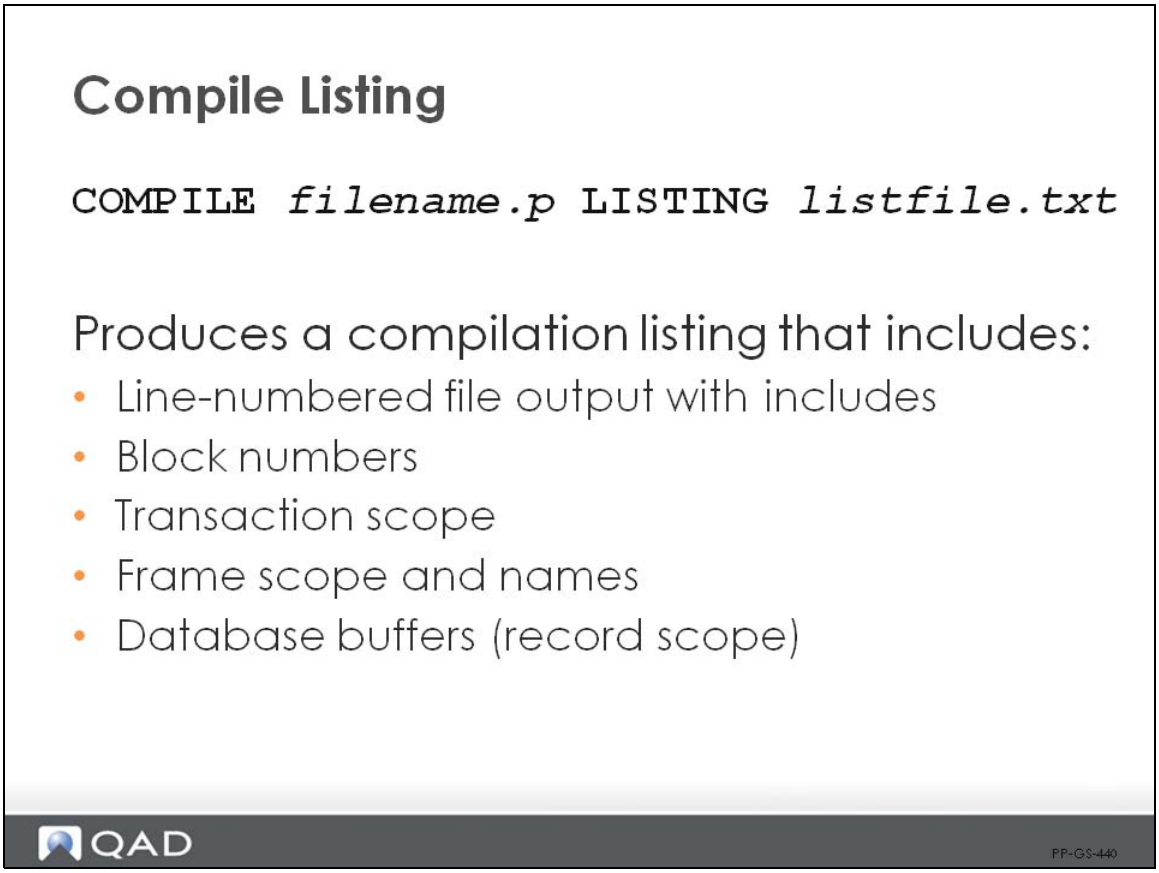
Compile XREF



The following illustrates sample output of the XREF option. The output includes two search strings, highlighted in bold text, that illustrate the index usage. The first SEARCH of the pt_mstr table uses WHOLE-INDEX, meaning that every record in the pt_mstr table will be read. This indicates a poorly formatted WHERE clause leading to a potential performance problem. The second SEARCH line for the pl_mstr table shows that the pl_prod_line index is being used.

```
1 COMPILE c:\_OE10\code\xref.p
1 CPINTERNAL ISO8859-1
1 CPSTREAM ISO8859-1
1 STRING "pt_mstr" 7 NONE UNTRANSLATABLE
1 SEARCH train1.pt_mstr pt_part WHOLE-INDEX
4 STRING "pl_mstr" 7 NONE UNTRANSLATABLE
4 ACCESS train1.pl_mstr pl_domain
4 ACCESS train1.pt_mstr pt_domain
4 ACCESS train1.pl_mstr pl_prod_line
4 ACCESS train1.pt_mstr pt_prod_line
4 SEARCH train1.pl_mstr pl_prod_line
5 ACCESS train1.pt_mstr pt_part
5 ACCESS train1.pt_mstr pt_desc1
5 ACCESS train1.pl_mstr pl_prod_line
5 ACCESS train1.pt_mstr pt_um
5 ACCESS train1.pt_mstr pt_price
5 STRING "x(18)" 5 NONE TRANSLATABLE  FORMAT
5 STRING "x(24)" 5 NONE TRANSLATABLE  FORMAT
5 STRING "x(4)" 4 NONE TRANSLATABLE  FORMAT
12 STRING "Item Number" 11 LEFT TRANSLATABLE
12 STRING "Description" 11 LEFT TRANSLATABLE
12 STRING "Line" 4 LEFT TRANSLATABLE
12 STRING "pt_part" 7 NONE UNTRANSLATABLE
12 STRING "pt_desc1" 8 NONE UNTRANSLATABLE
12 STRING "pl_prod_line" 12 NONE UNTRANSLATABLE
```

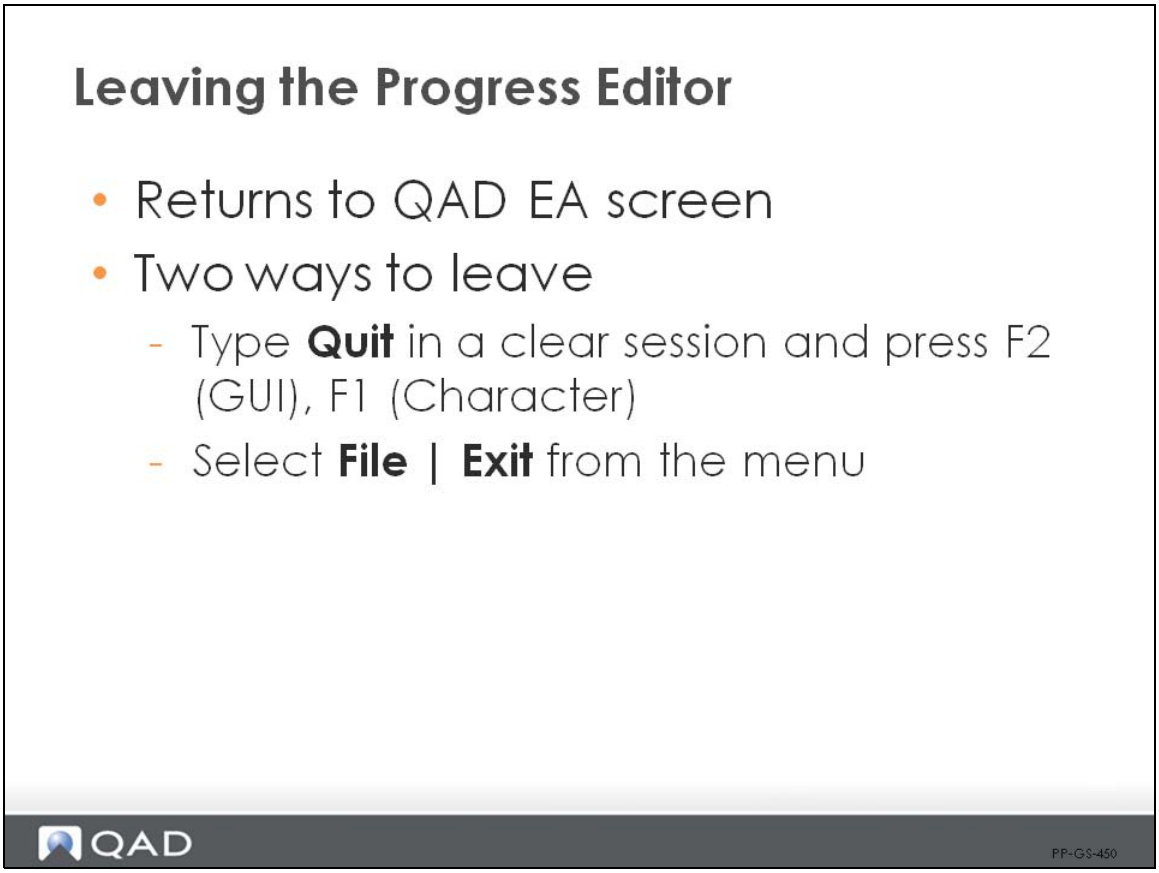
Compile Listing



This option produces a listing of the compilation that includes the following information:

- The number of the block where each statement belongs
- The name of the source file you are compiling
- The date and time at the start of the compilation
- The line number of each line in the source file
- The complete text of all include files and the name of any subprocedures
- The buffer and frame scopes to procedures, internal procedures, and trigger blocks

Leaving the Progress Editor



When the Progress editor is closed, the QAD EA menu screen is displayed. You can exit in two ways.

Option 1:

1 Select File|New to get a clear editor buffer.

2 Type quit.

3 Press F2 (GUI) or F1 (character).

Option 2:

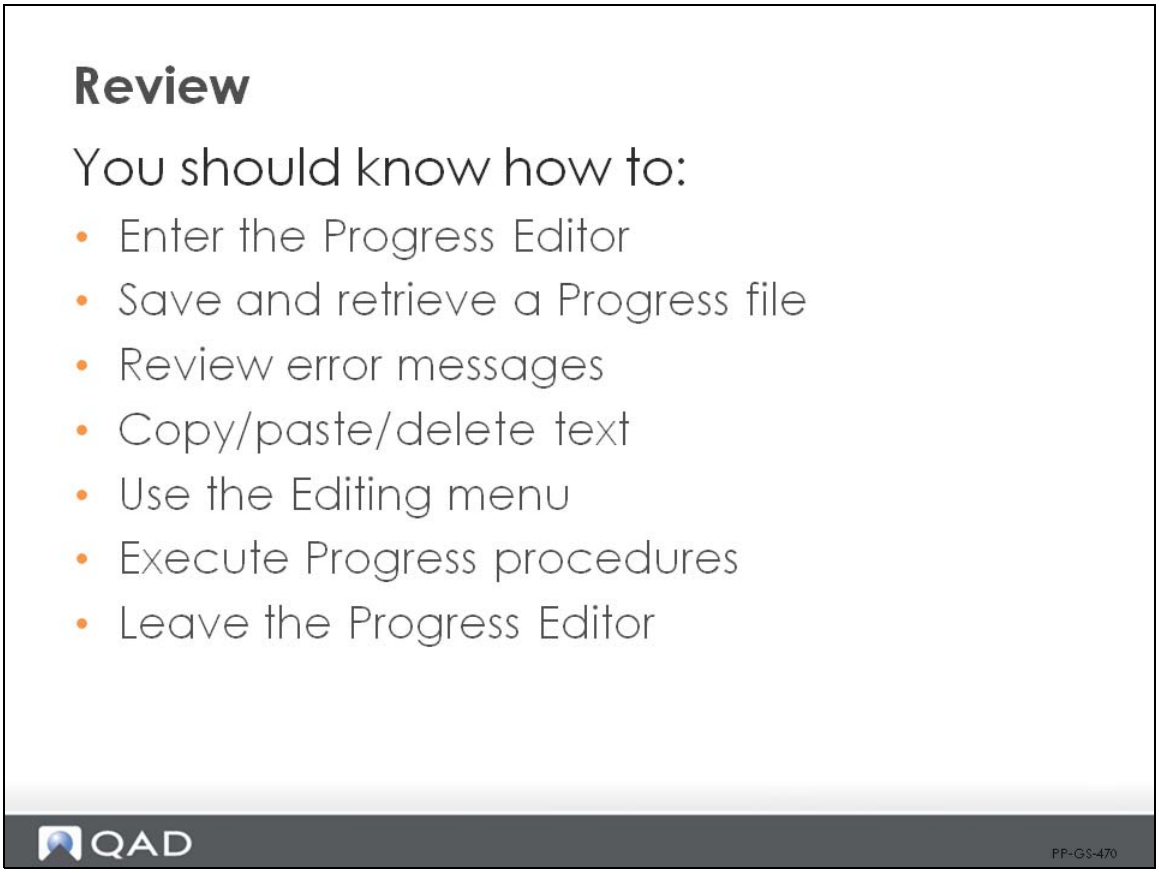
Select File|Exit from the menu.

Exercise 3-3: Leaving

1 Make sure that all of your programs are saved, then quit the Progress session using one of the two methods.

2 Start a new Progress Editor session.

Review



In this lesson, you learned how to get in and out of the Progress Editor and how to use the Progress pull-down menus.

You will perform both of these actions throughout the rest of the lessons, so make sure you feel comfortable doing them.

Lesson 4: Creating Simple Procedures

In this lesson, you will learn how to create simple Progress procedures to access and display information from the database.

Upon completion of this lesson, you will be familiar with the following topics:

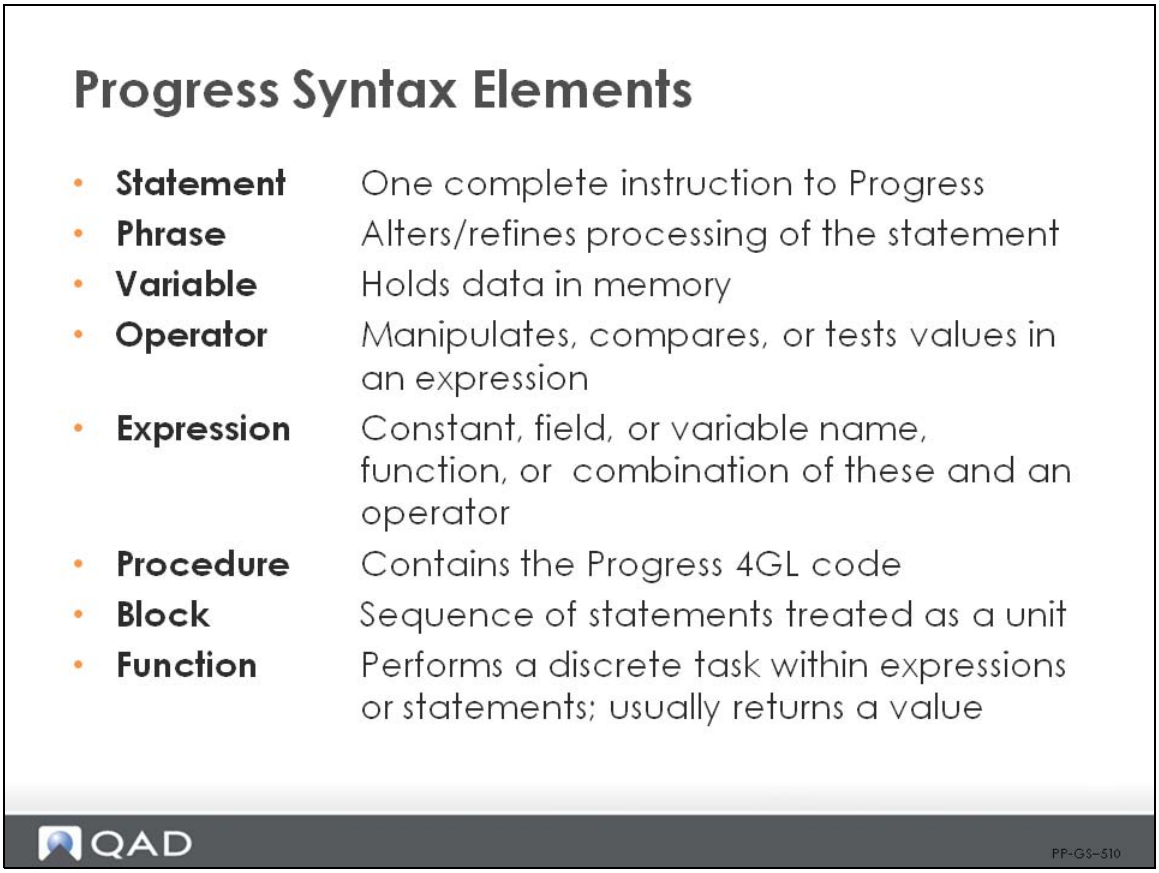
- Using DISPLAY statements to display expressions
- Using FOR EACH statements to retrieve records for a table and NO-LOCK to read records without locking them
- Using the MESSAGE statement to display messages
- Using the PAUSE statement
- Using the OUTPUT statement to direct reports to a printer, terminal, or file and manage multiple streams
- Using the EXPORT, PUT, and IMPORT statements

Conventions for Progress Syntax in Class

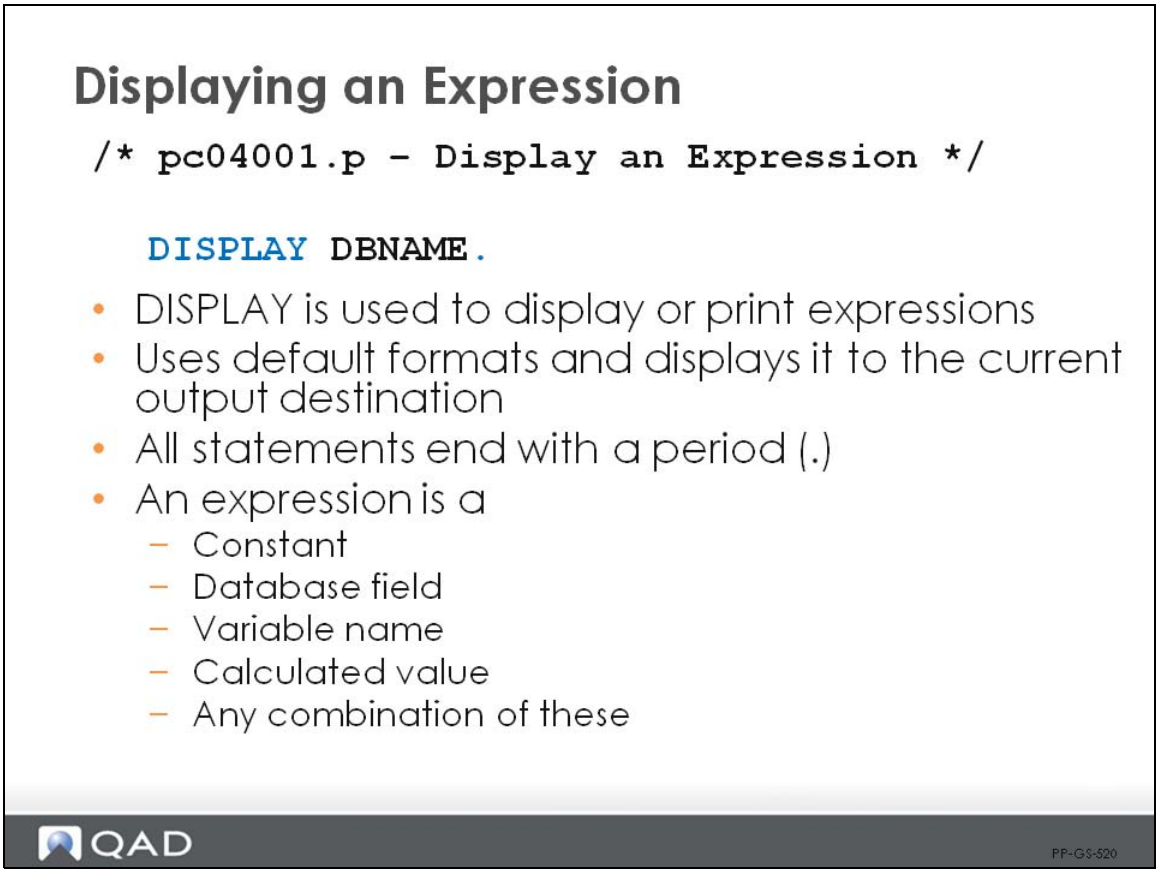
To highlight Progress syntax, the following conventions are used in the examples in this course:

- Progress keywords are shown in upper case for emphasis in the code. However, Progress is not case sensitive. QAD standard practice is to use all lower case when writing source code.
- Table and field names are shown in lower case.
- Character strings are surrounded by quotes (" ").
- Square Brackets ([]) indicate optional items.
- Ellipses (…) indicate an item may be repeated.

Progress Syntax Elements



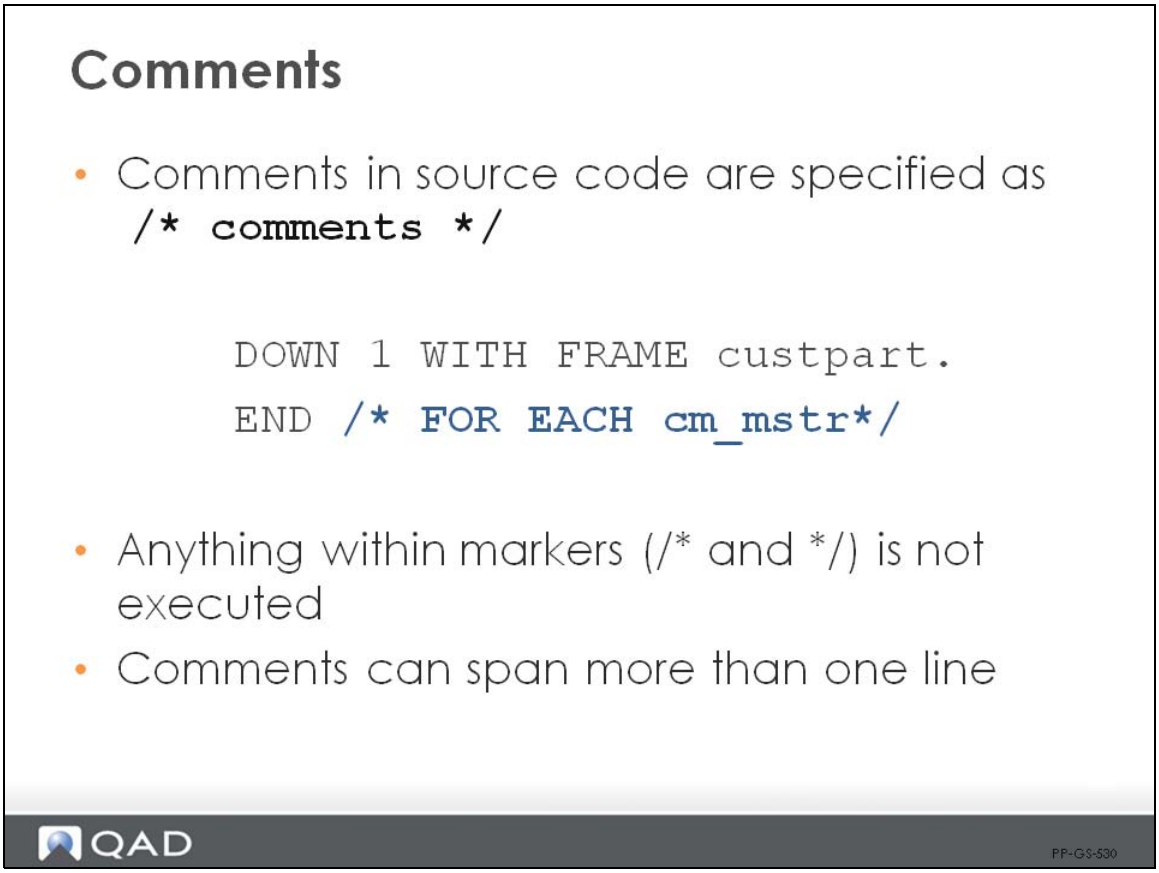
Displaying an Expression



This is a simple program displaying the database name. An expression is a constant, field name, variable name, calculated value, or any combination of these.

In QAD EA, the most common use for the DISPLAY statement is to display/print database fields or calculated values. Data types for expressions can be character, date, decimal, integer, logical, or handle. See “Data Types” on page 23 for data types and default formats.

Comments



Comment lines in Progress programs begin with /* and end with */. Comments can span several lines.

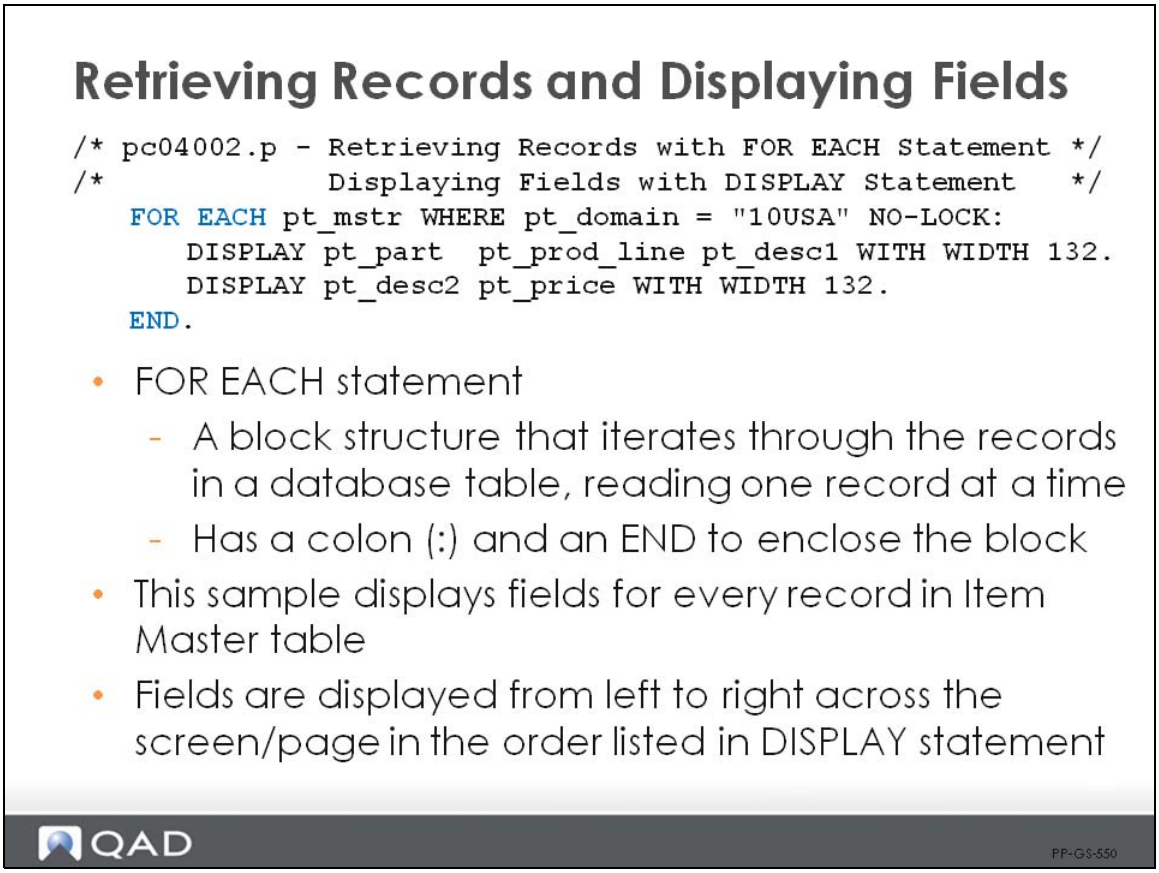
Exercise 4-1: Display

Type each of the following strings in the Progress Editor and then choose Compile|Run to see the result.

1. DISPLAY Progress.
2. DISPLAY TODAY.
3. DISPLAY "Simple Arithmetic:" 5 + 4 / 2.

Note the period at the end of each statement. The case of the statement does not affect the results.

Retrieving Records and Displaying Fields



This program displays several fields for every record in the Item Master. Records are retrieved in the order of the primary index pt_part.

- The FOR EACH statement is a block structure that iterates through the records in a database table and reads one record at a time.
- The DISPLAY statements inside the FOR EACH block are executed once for each record in pt_mstr.
- Progress places a box around the display and controls the paging. The fields are displayed from left to right across the screen/page in the order they are listed in the DISPLAY statement. Progress wraps fields that do not fit on the first line around to the next line.
- Field labels and other display characteristics are taken from the Data Dictionary field definitions when each field is displayed.
- All statements end with a period (.), except for DO, FOR EACH, and REPEAT block statements, which end with a colon (:).
- One or more statements can appear within a DO, FOR EACH, or REPEAT block. They are executed on each iteration of the block.
- An END statement is required following a DO, FOR EACH, or REPEAT statement. The END statement indicates the end of all statements that are to be processed within that block.

Note When multiple records are retrieved, you can use F4 (character) or Esc (GUI) to terminate the display before the program completes.

Exercise 4-2: Retrieve and Display

Retrieve `pc04002.p`, modify it as follows, and then choose Compile|Run to see the effect.

1. First change:

   From:
   ```
   DISPLAY pt_desc2 pt_price WITH WIDTH 132.
   ```
   To:
   ```
   DISPLAY pt_desc2 WITH WIDTH 132.
   ```

2. Second change:

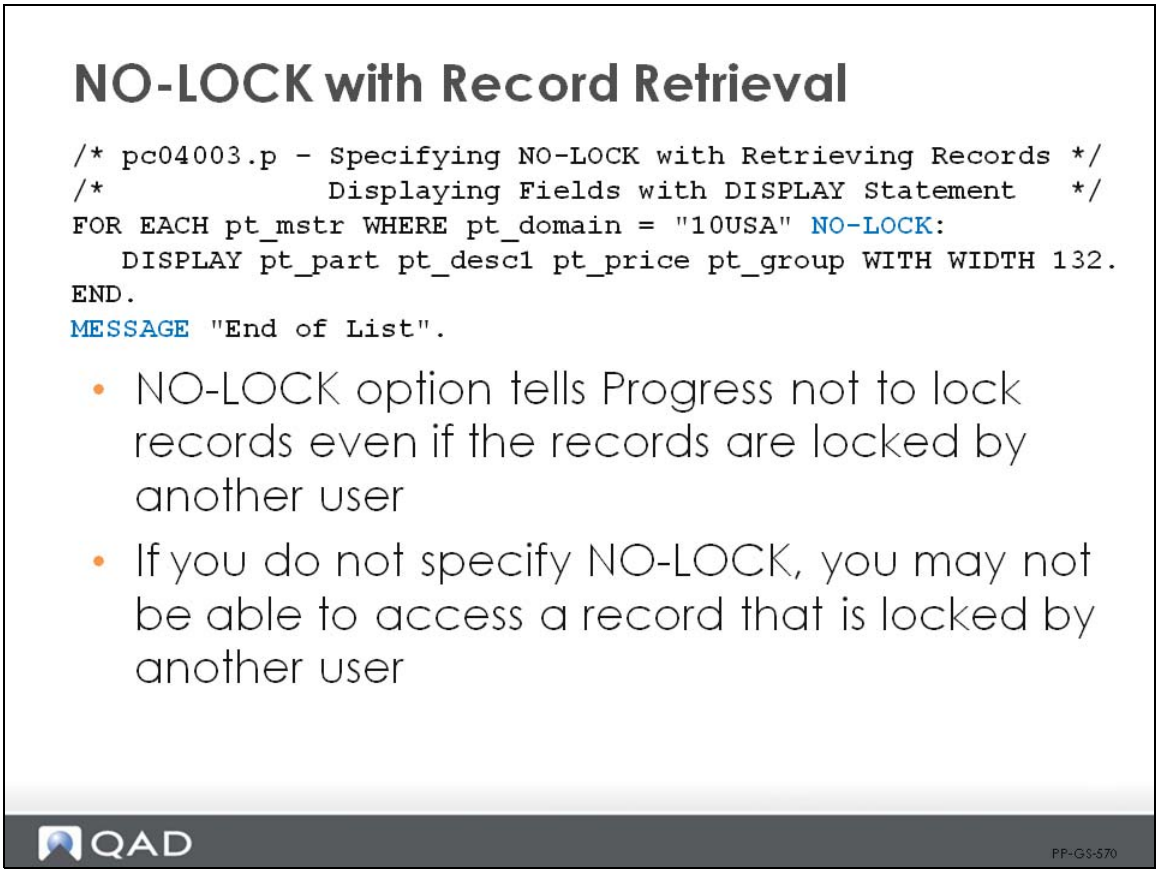
   From:
   ```
   DISPLAY pt_part pt_prod_line pt_desc1 WITH WIDTH 132.
   DISPLAY pt_desc2 WITH WIDTH 132.
   ```
   To:
   ```
   DISPLAY pt_part pt_prod_line pt_desc1 pt_desc2 WITH WIDTH 132.
   ```

3. Third change:

   From:
   ```
   DISPLAY pt_part pt_prod_line pt_desc1 pt_desc2 WITH WIDTH 132.
   ```
   To:
   ```
   DISPLAY pt_part pt_desc1 pt_price + (pt_price *.10) pt_group
     WITH WIDTH 132.
   ```

Note the period at the end of each statement.

NO-LOCK with Record Retrieval



The NO-LOCK option in the record phrase tells Progress not to lock records even if another user has the record locked. By specifying NO-LOCK, you ensure that the procedure does not unintentionally update the database. Progress does not permit a procedure to update the database when NO-LOCK was specified. You also avoid unnecessary record contention in a multi-user environment.

Progress does not require that NO-LOCK be specified; your code will compile and pass a syntax check without it. However, QAD code is always written with EXCLUSIVE-LOCK or NO-LOCK to avoid issues that otherwise occur in an Oracle environment. You must remember to explicitly specify EXCLUSIVE-LOCK or NO-LOCK.

If you do not specify NO-LOCK, you may not be able to access a record that is locked by another user. Your procedure will wait until the other user finishes with that record. A record read with NO-LOCK can be changed by another user. Other users can read, change, or delete a record you have read with NO-LOCK.

MESSAGE Statement

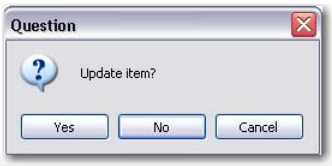
QAD does not use the MESSAGE statement for displaying messages. Instead, an include file (`pxmsg.i`) is used in conjunction with message text stored in the msg_mstr table. This approach supports the reuse of messages, consistency, and translation requirements.

ALERT-BOX Phrase

ALERT-BOX Phrase

- Graphical message box with buttons
  - INFO, QUESTION, WARNING, ERROR types
  - YES-NO, YES-NO-CANCEL, OK, OK-CANCEL, RETRY-CANCEL button sets
- Alert-box messages can only update logical variables

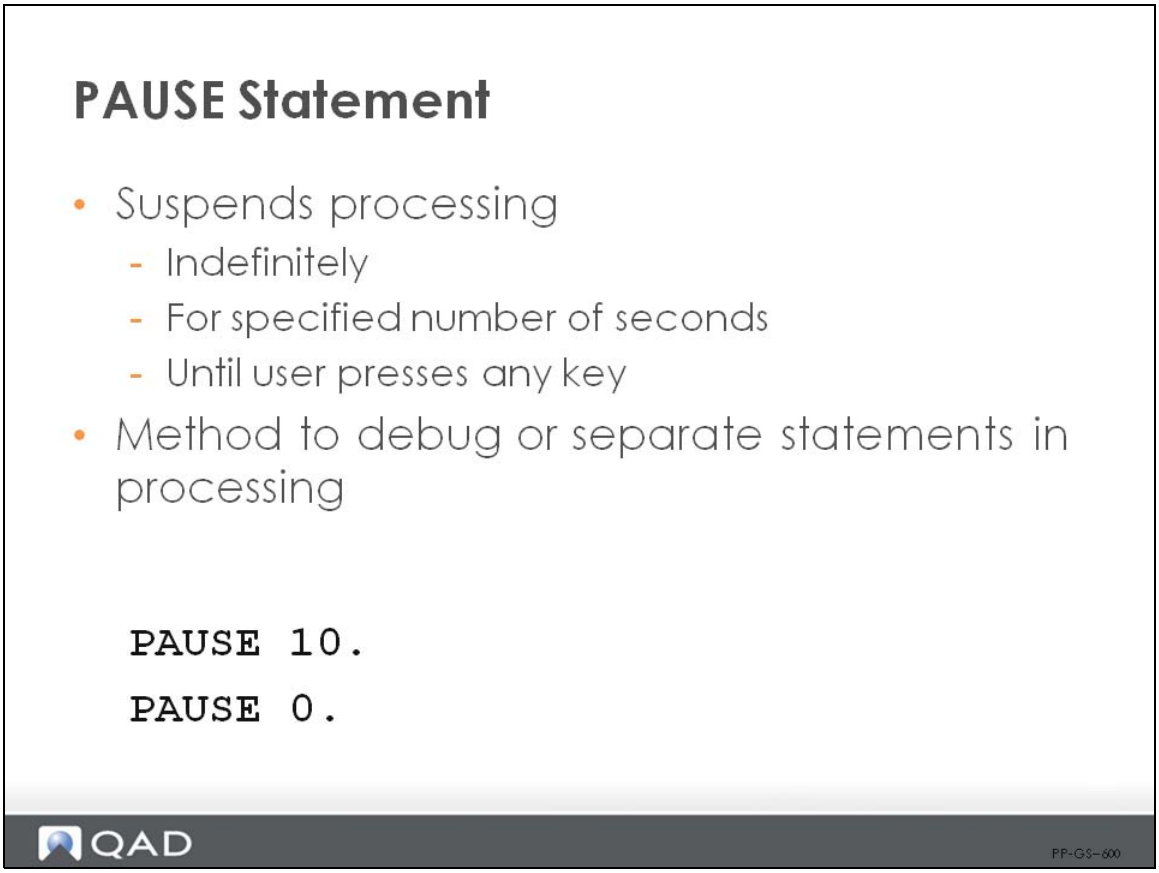
```
MESSAGE
   "Update item?"
   VIEW-AS ALERT-BOX QUESTION BUTTON YES-NO-CANCEL
    UPDATE x AS LOGICAL.
```



QAD

PP-GS-590

PAUSE Statement



PAUSE 10 halts processing for 10 seconds.

If a condition in the code caused a “press spacebar to continue” message that you did not want, you can use the PAUSE 0 statement to prevent that message from happening.

Note Using the PAUSE statement in programs that run under the QAD .NET UI can cause display problems. Instead, you can use a special include file `gpwait.i` to manage the delay.

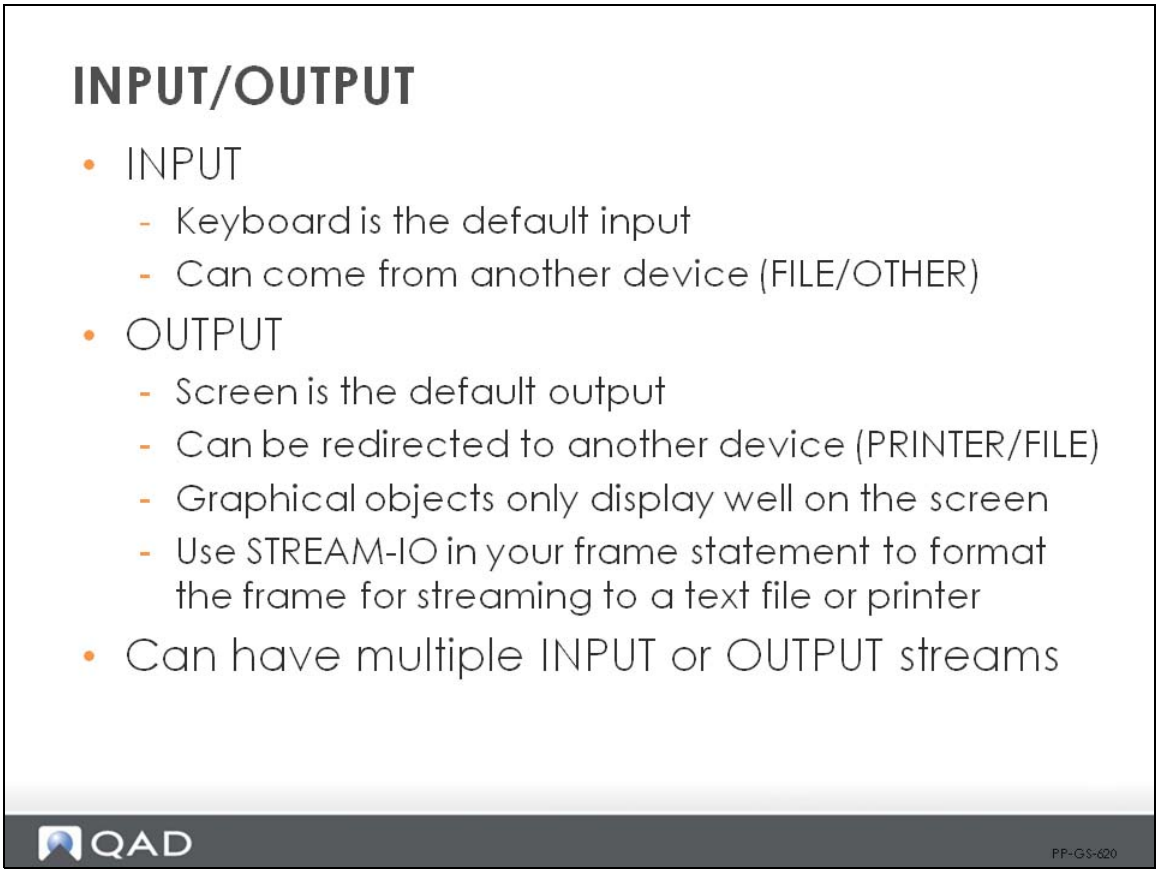
Exercise 4-3: Display Customer Data

Write a procedure that displays the following:

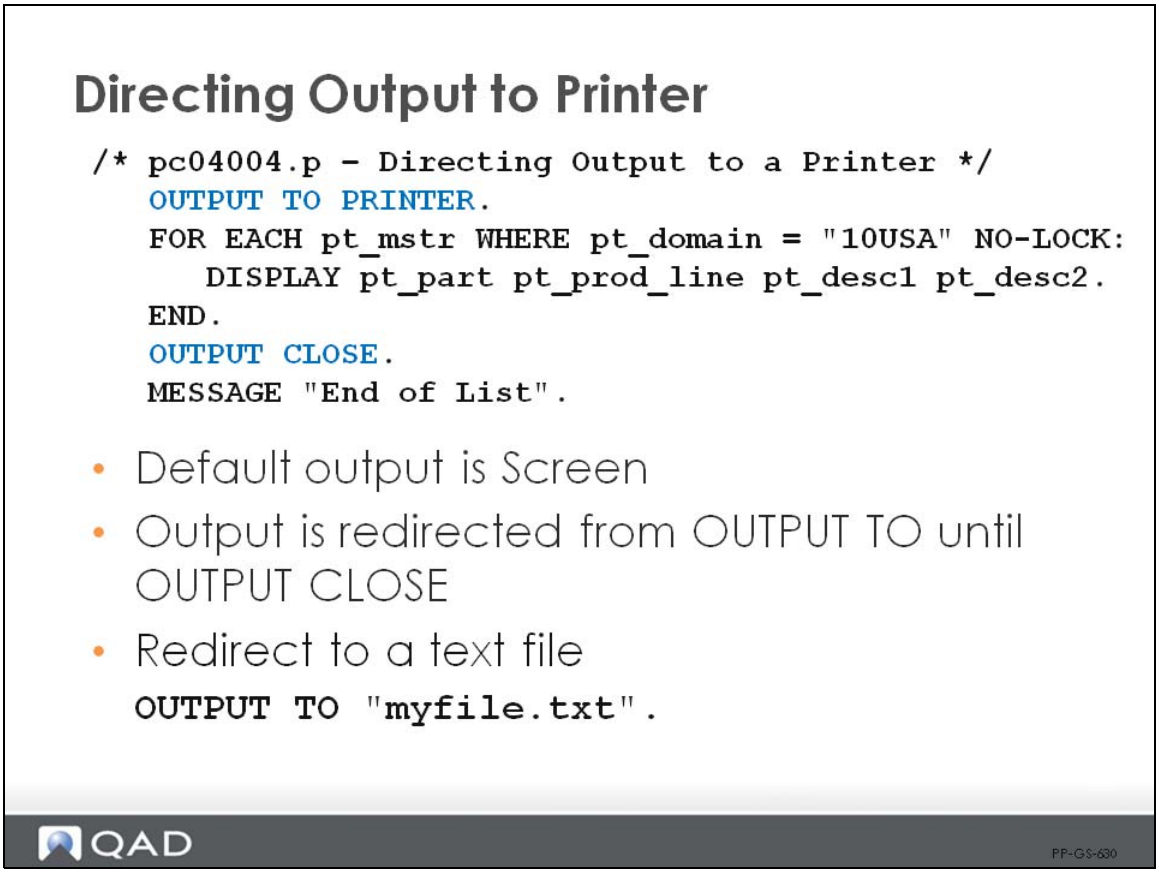
- Customer number (address)
- Customer sort name
- Current balance
- Region
- Last sale date

Hint. Use table cm_mstr and set cm_domain to 10USA.

INPUT/OUTPUT



Directing Output to Printer



By default, a procedure's output is directed to the terminal. The OUTPUT TO PRINTER statement redirects output to the default system printer. Output remains redirected until Progress encounters an OUTPUT CLOSE statement or the procedure ends.

You can redirect output to a printer other than the default system printer by entering the name or ID of the device or file (for example, term1, lpt2, yourfile.txt).

The default number of lines per page is 56. To suppress page breaks, specify PAGE-SIZE 0.

When output is directed to a printer, the output is automatically paged. To make output to a text file paged, add the PAGED keyword after the text file name in the OUTPUT TO statement (for example, OUTPUT TO myfile.txt PAGED).

The default when redirecting output to a file is that the file is overridden each time. To keep the current contents and add more lines to the bottom, use the APPEND keyword with OUTPUT TO (for example, OUTPUT TO myfile.txt APPEND).

Exercise 4-4: Output

1 Modify procedure `pc04004.p`:

From: `OUTPUT TO PRINTER.`
To: `OUTPUT TO myfile.txt.`

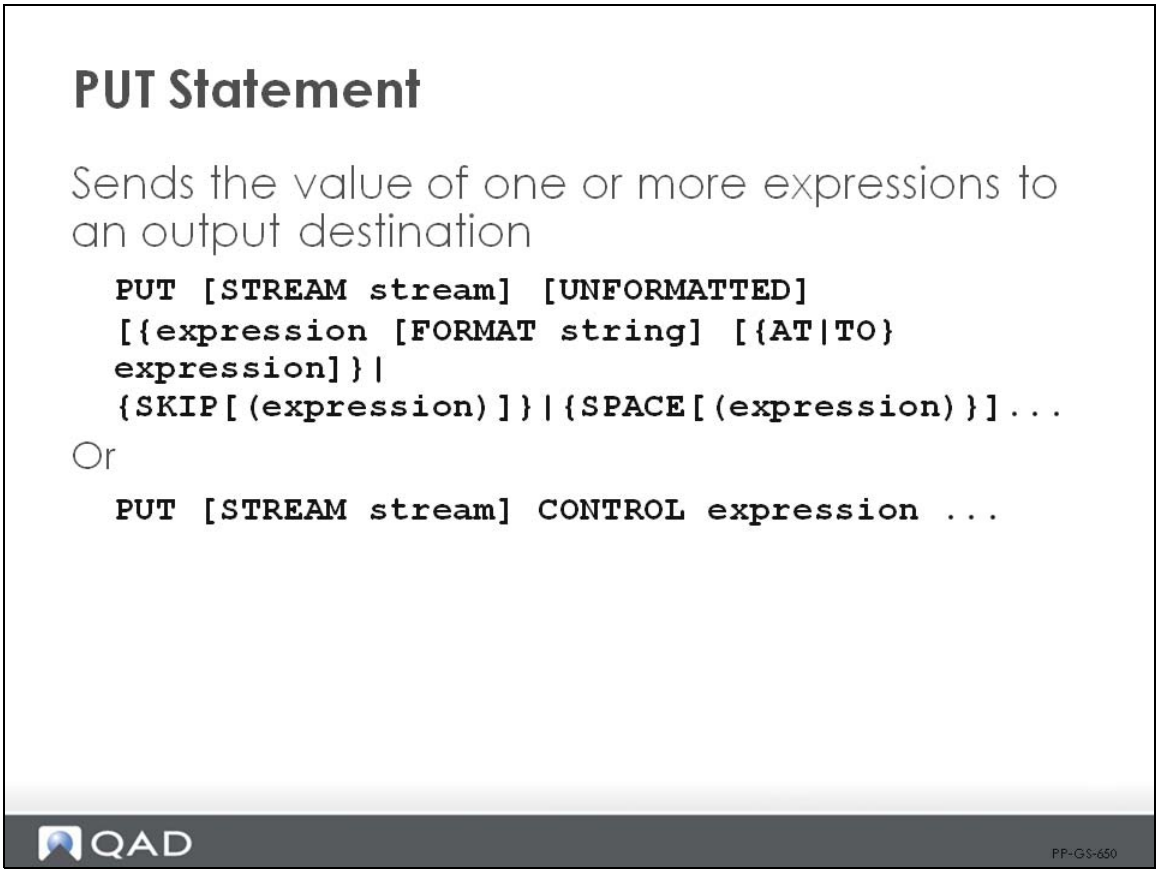
2 Retrieve the data file that you directed the print output to.

   a Select Open from the File pull-down menu.

   b Enter the file name `myfile.txt`.

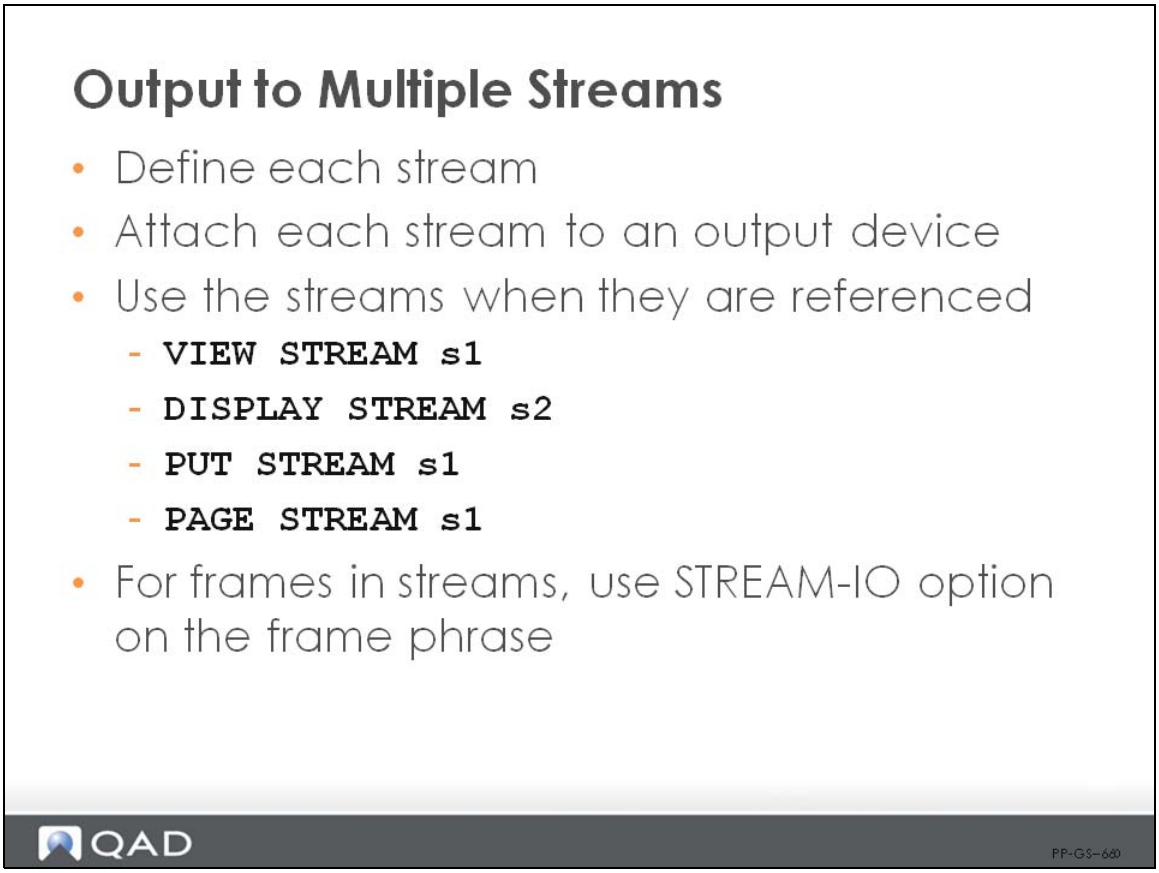
3 Review the file.

PUT Statement



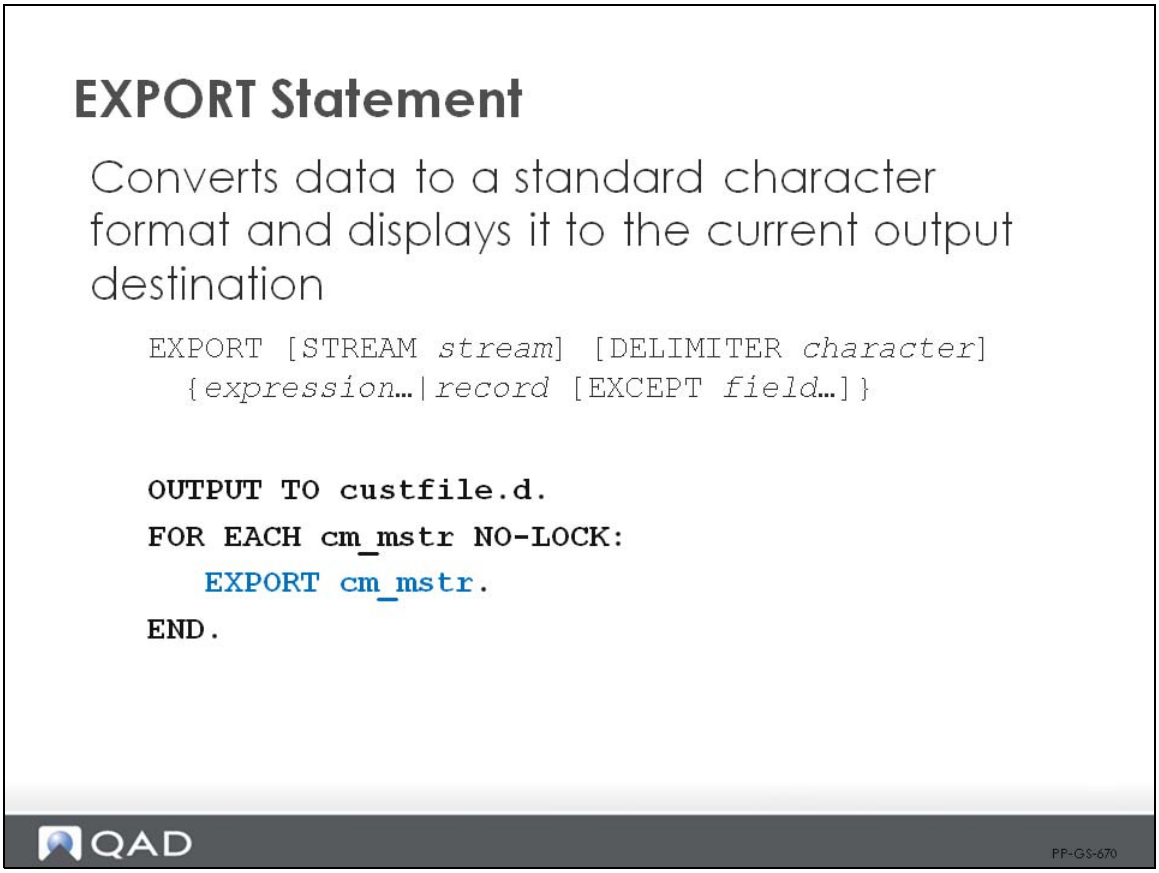
The PUT statement sends the value of one or more expressions to an output destination other than the terminal.

Output to Multiple Streams



STREAM-IO specifies that all output from the compiled procedure or class is formatted for output to a file or printer. This means that all font specifications are ignored and all frames are treated as text, producing platform-independent output appropriate for printing.

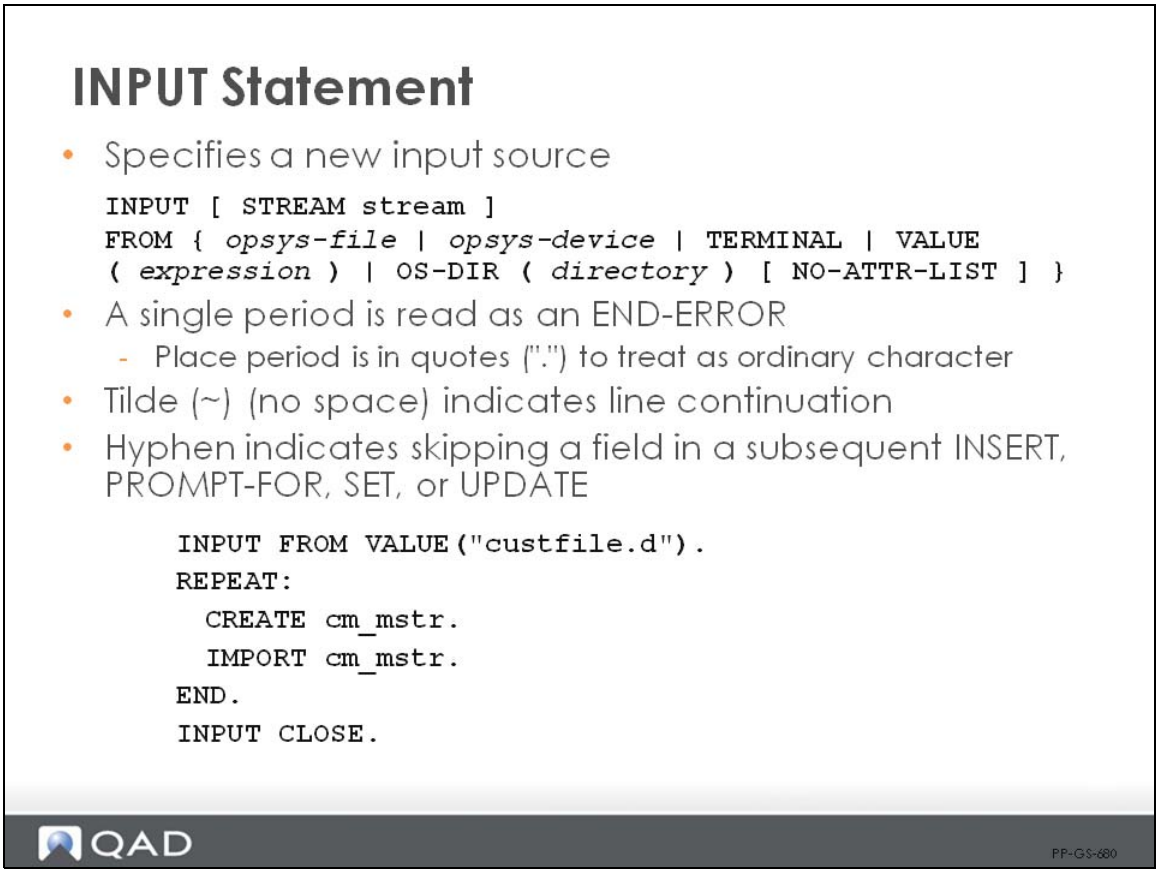
EXPORT Statement



The EXPORT statement converts data to a standard character format and displays it to the current output destination (except when the current output destination is the screen) or to a named output stream. You can use data exported to a file in standard format as input to other procedures.

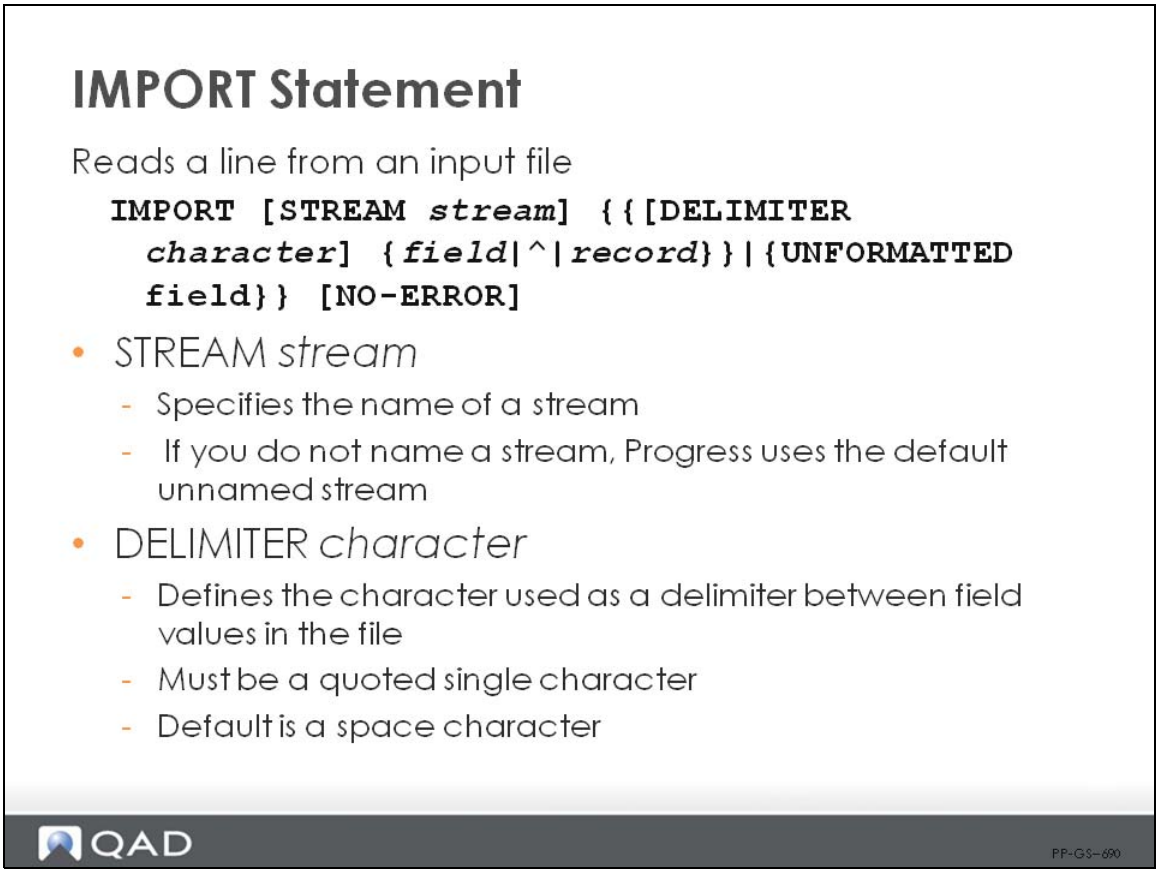
When exporting a database table, the data appears as a comma-separated list for reach record (or whatever is specified using the DELIMITER phrase). The contents are listed in the sequence designated by the Order attribute on each field in the Data Dictionary.

INPUT Statement



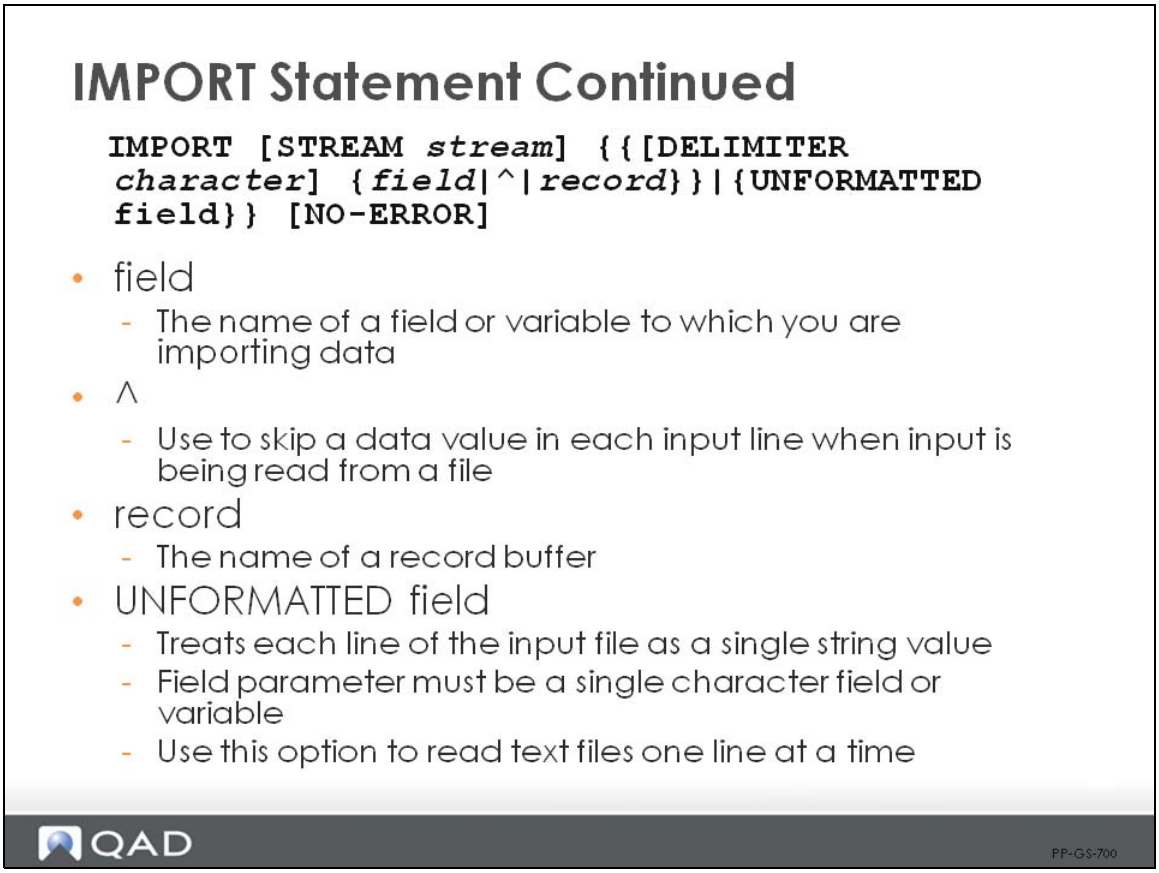
The INPUT FROM statement specifies the new input source for a stream. In this example, the data exported in the previous example is input from the file `custfile.d`.

IMPORT Statement

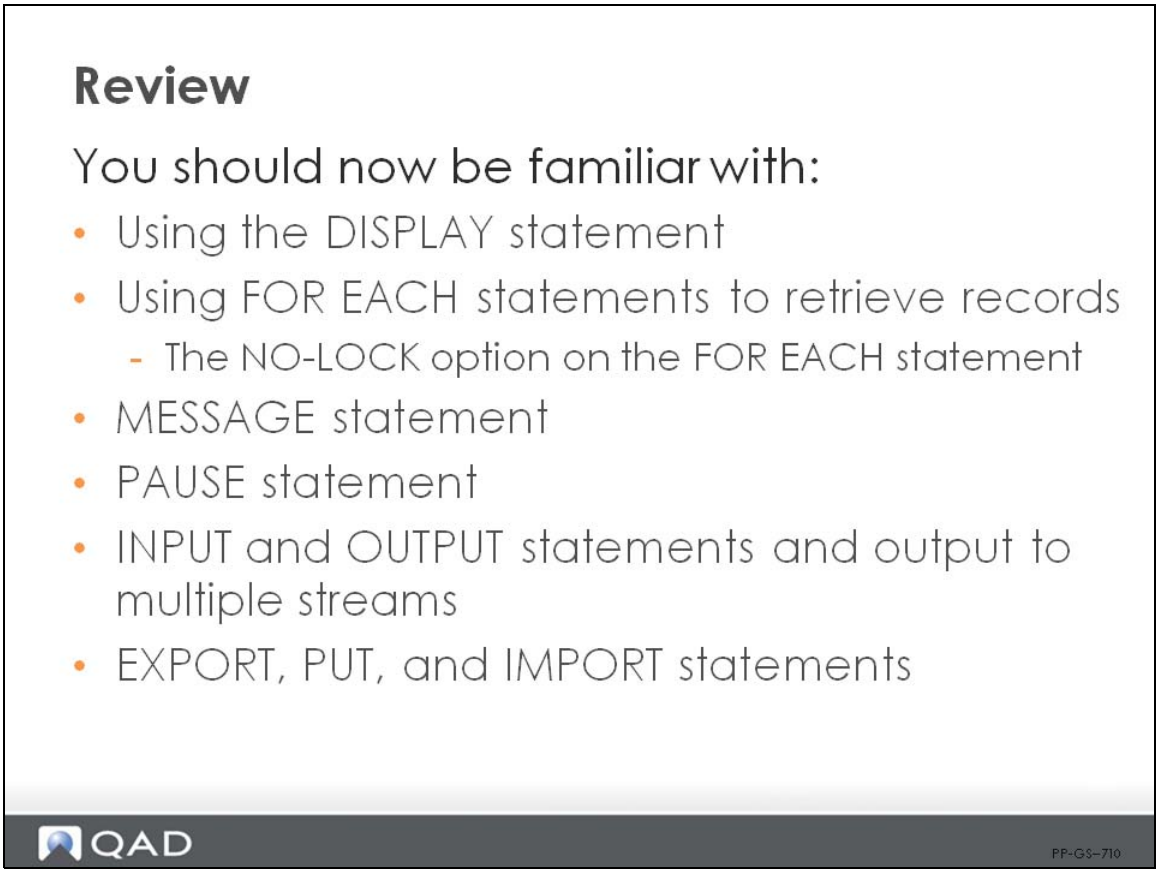


The IMPORT statement reads a line from an input file that might have been created by EXPORT as shown in the example on the previous slide.

IMPORT Statement 2



Review



In this lesson, you learned how to write basic Progress procedures. The simplest procedure consisted of the DISPLAY statement with an expression, which can be a constant, field name, variable name, or any combination of these.

You should be familiar with the expressions and five Progress commands: DISPLAY, FOR EACH, END, MESSAGE, and OUTPUT.

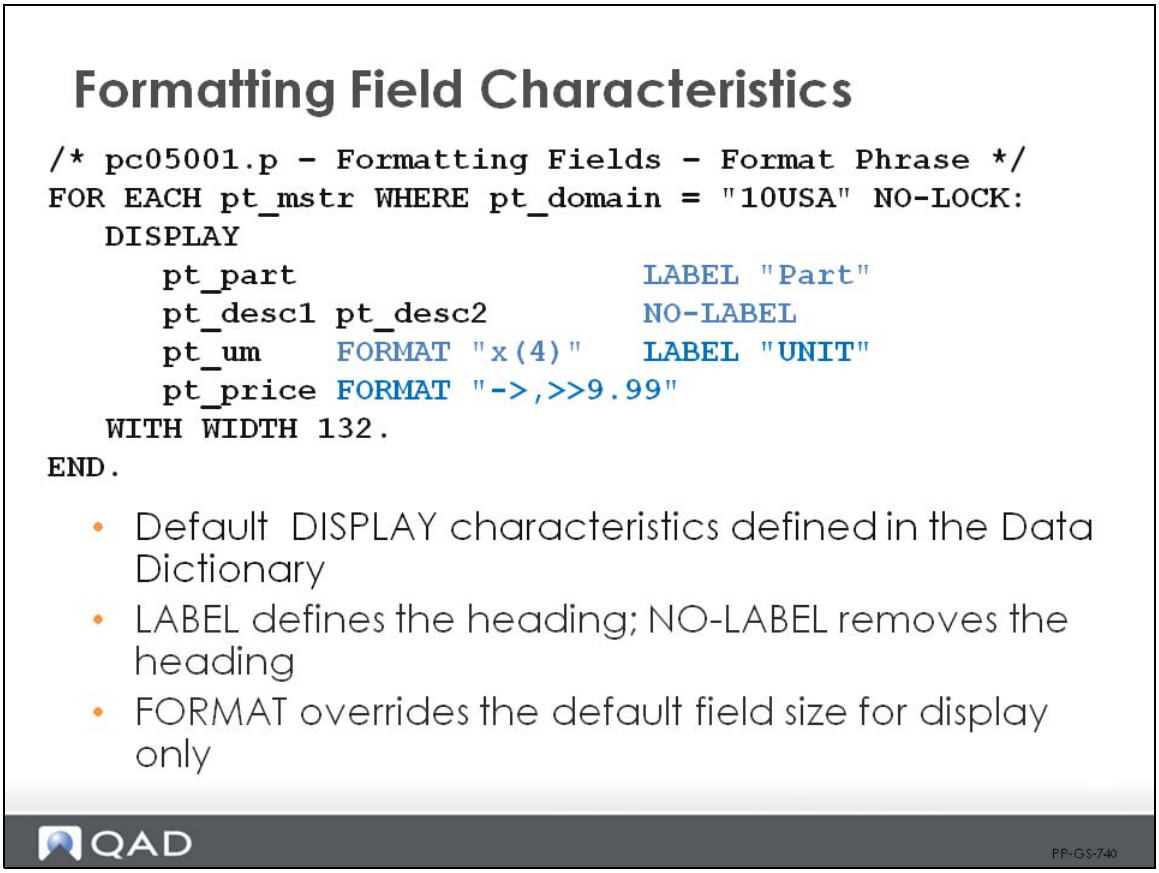
- The DISPLAY statement is used to display an expression.
- The FOR EACH statement is used to retrieve records from a file and the END statement must be used at the end of a FOR EACH block. The NO-LOCK option on the FOR EACH statement should be used in the record phrase to avoid record locking.
- The MESSAGE statement is used to display to the message lines as the bottom of the screen.
- The PAUSE statement halts processing for specified number of seconds.
- The OUTPUT statement is used to direct reports to the default system printer and manage output to multiple streams.
- The PUT statement allows you to write unformatted data to a screen or file.
- The EXPORT statement exports data to a file.
- The IMPORT statement imports data from a file.

Lesson 5: Formatting, Sorting, and Report Totals

Upon completion of this lesson, you will be familiar with the following topics:

- Using the DISPLAY statement FORMAT phrase to modify the way fields are displayed
- Using the FRAME phrase to modify framing characteristics
- Sorting records using the BY option of the FOR EACH statement
- Generating report totals and subtotals with the aggregate phrase within the DISPLAY statement

Formatting Field Characteristics



You can override the default display characteristics defined for each expression with the FORMAT phrase. The default display characteristics are defined in the Data Dictionary.

The LABEL option of the FORMAT phrase defines the heading used on the field. It overrides the Data Dictionary label definition for this procedure only. Correspondingly, you can use NO-LABEL to suppress the display of the label defined in the Data Dictionary.

You can also use the FORMAT phrase to control the size of the data displayed. This is frequently used to truncate the field for display purposes. Note the following:

- FORMAT controls only how the data is displayed, not the actual size of the stored data. For example, a format of x(4) does not restrict the user from storing more than four characters in a field. It only limits the display of that field to the first four characters.
- The size of the data does not control the display. Defining a field to contain 24 characters does not automatically guarantee that all 24 characters will always be displayed. The content displayed is governed by the FORMAT statement, in the Data Dictionary, or as specified in the program using the field.
- LABEL (or COLUMN-LABEL) controls the width of the column and its title, but does not influence the size of the data displayed in the column. The column details are controlled by the FORMAT statement. A column can have a heading that is four characters and the actual data is 24 characters, but if the FORMAT is x(8) only eight characters of data are displayed.

Exercise 5-1: Field Format

1 Open `pc05001.p` and run it without making any changes. Note the unit of measure for part 01011 says EACH.

2 Make the following change and run again:

From: `pt_um  FORMAT "x(4)"  LABEL "Unit"`

To: `pt_um  LABEL "Unit"`

Why is the Unit column four characters wide but part 01011 only shows EA instead of EACH?

3 Now make this change and run to see the effect:

From: `pt_um  LABEL "Unit"`

To: `pt_um  NO-LABEL`

Why does the unit of measure column still show only two characters?

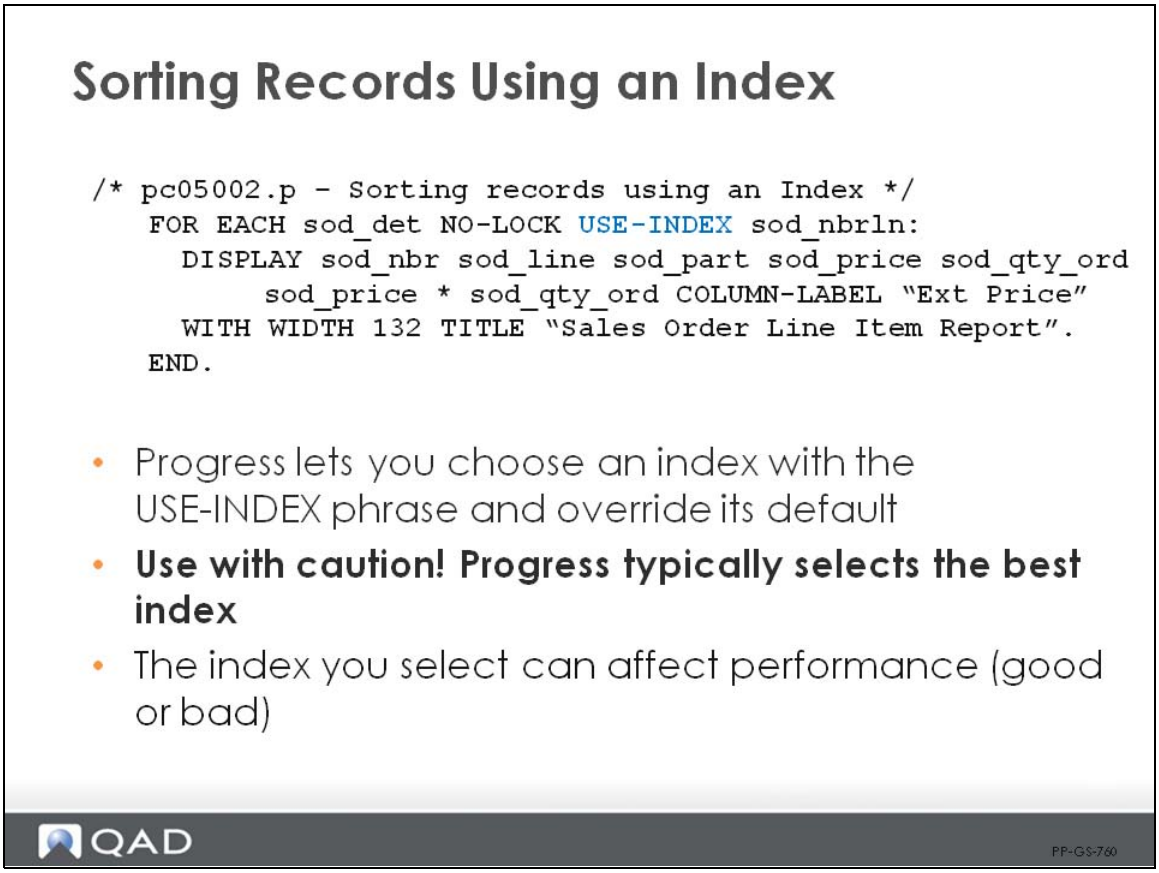
4 And finally, make this change and notice the effect:

From: `pt_desc1 pt_desc2  NO-LABEL`

To: `pt_desc1 + pt_desc2`

What happened to the Description column? Why?

Sorting Records Using an Index



Progress allows you to specify the index to use when reading records from a table with the FOR EACH statement. However, in general, you should avoid this practice since Progress is better at determining the best index to use.

Important Significant performance degradation can result from specifying the wrong index.

For specifying the sort order, use the BY phrase.

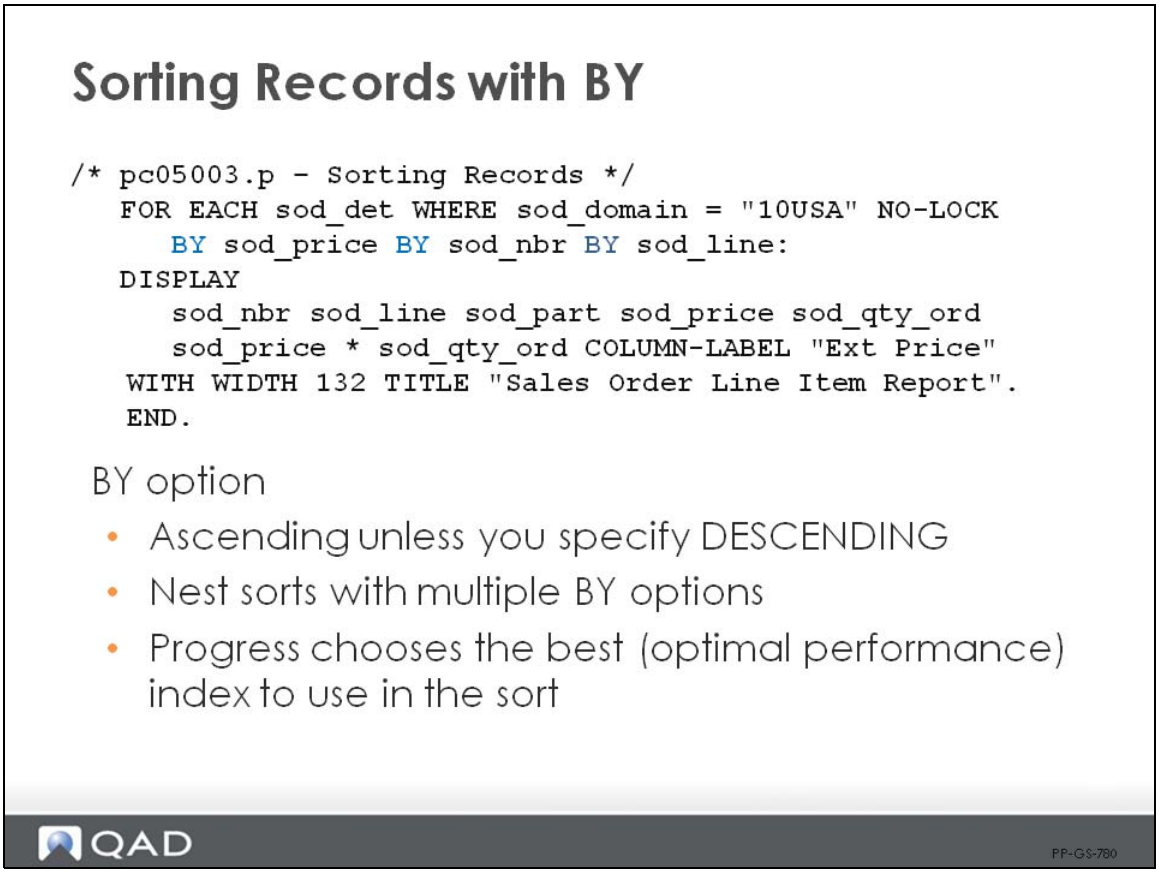
Exercise 5-2: Sorting Records

1 Write a procedure to report location detail records (ld_det) sorted by item number, then site, and then location.

2 Write a procedure to report inventory transactions (tr_hist) sorted by transaction type.

Hint. In the procedure, use `ld_domain = "10USA"` and `tr_domain = "10USA"`

Sorting Records with BY



The FOR EACH statement retrieves records ordered by the primary index unless you specify other sorting criteria. You can sort on any field or expression. The sort is ASCENDING unless you specify DESCENDING as part of the BY option. Nested sorts can be done using multiple BY options. Records are sorted first by the field specified in the first BY, then sorted by the field specified in the second BY, and so on.

Fields with character data type sort in character sequence even when they contain numeric values. For example, when the numbers 1, 2, and 11 are stored in a character type field, they are displayed 1, 11, 2 when sorted since the fields are sorted character by character. To avoid this problem, assign the numbers preceded with a 0 (zero). They would look like 01, 02, 11.

When BY is used, Progress chooses the best index to assist with the sorting.

Exercise 5-3: Sort using BY

1 Retrieve `pc05003.p` and modify:

From:
```
BY sod_price BY sod_line
```
To:
```
BY sod_price * sod_qty_ord
DESCENDING BY sod_part
```

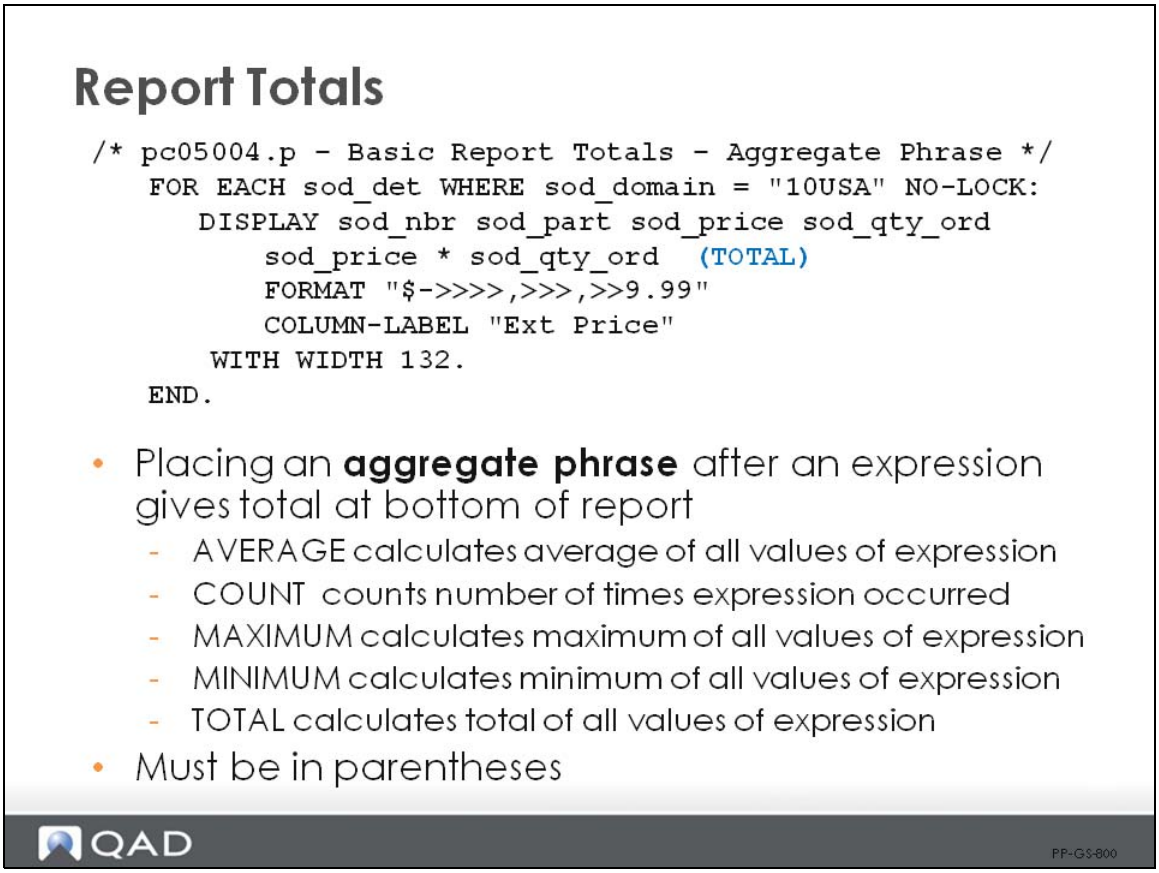
2 Write a procedure to sort the Sales Order Master table by customer.

Hint. Use so_cust and sod_doman = "10USA".

Display the sold-to customer code, sales order number, order date, and due date.

3 Modify the above procedure to sort the sales orders by order date and then by the due date.
Hint. Use so_ord_date and so_due_date.

Report Totals



Most reports are incomplete without totals. The example above totals the dollar value of all pending sales orders. The report has far more value with this total than without it. Totals are produced by using an aggregate phrase after an expression in a DISPLAY statement.

Placing the option TOTAL after an expression or variable name displays a running total at the bottom of the report. The aggregate phrase must be placed within parentheses () and must directly follow the field to be totaled. Totals can be calculated for any number of fields.

An aggregate phrase identifies one or more aggregate values to calculate for a field. This example shows the TOTAL option. More than one aggregate value can be calculated for any given expression. These include:

- AVERAGE calculates the average of all of the values of an expression.
- COUNT counts the number of times the expression occurred.
- MAXIMUM calculates the maximum of all of the values of the expression.
- MINIMUM calculates minimum of all values for the expression.
- TOTAL calculates the total of all of the values of the expression.

Exercise 5-4: Report Totals

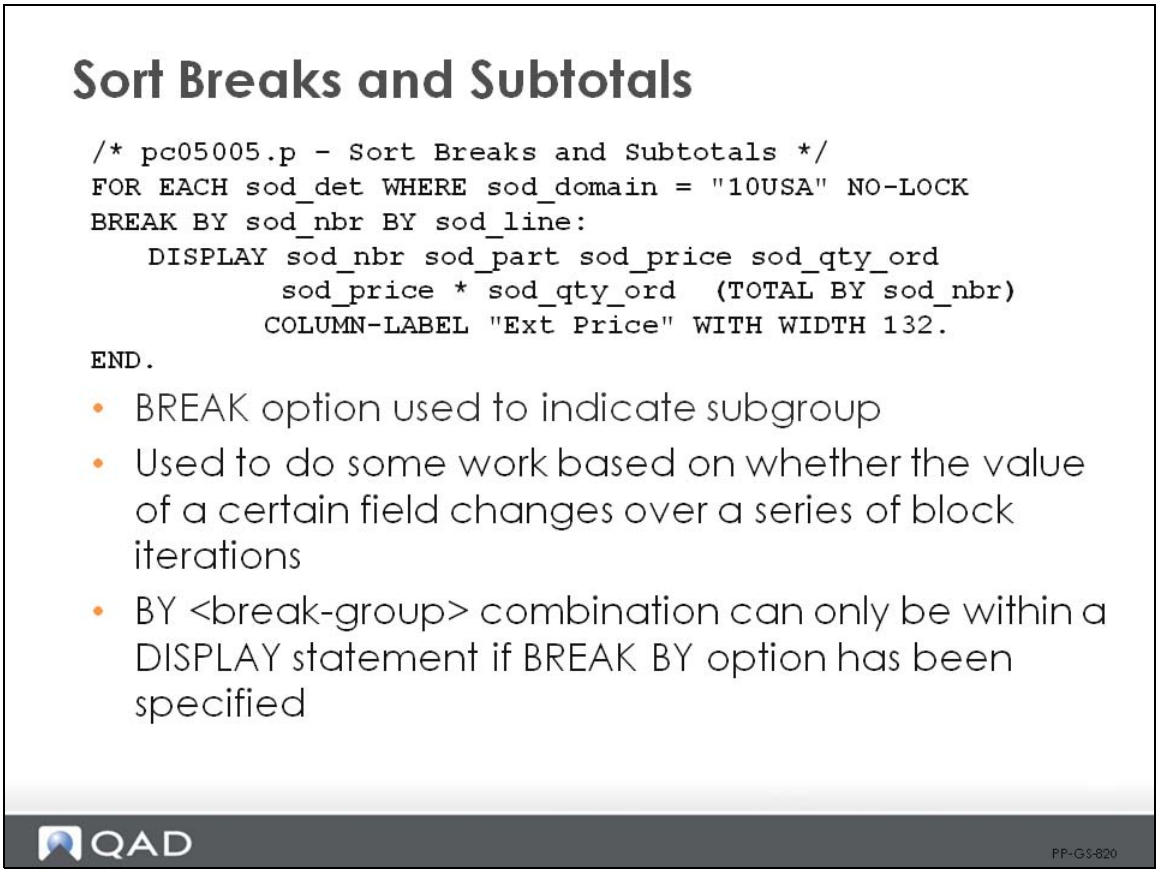
1 Write a procedure to report sales order detail records (sod_det) by item number (sod_part), and give a total of the quantity shipped (sod_qty_ship).

Hint. Use sod_domain = "10USA" in the WHERE clause.

2 Display the following:

- sales order line
- item number
- quantity ordered
- quantity shipped

Sort Breaks and Subtotals



Often, calculating statistics for the total report is not enough. You need to see subtotals and other calculations for a group of items. This example calculates total backlog value subtotaled by order.

The BREAK option is used with the FOR EACH statement to indicate that you want to do something when the value of a certain field changes. Using the TOTAL aggregate phrase option with a BY break group creates report totals and subtotals by the break group. This also permits the use of the options SUB-AVERAGE, SUB-MAXIMUM, SUB-COUNT, and so on. You can only use the BY <break-group> combination in an aggregate phrase within a DISPLAY statement when the BREAK BY option is specified in the FOR EACH statement.

Example:

```
FOR EACH table WHERE expression BREAK BY field:
    IF FIRST-OF(field) THEN…
    IF LAST-OF(field) THEN…
END.
```

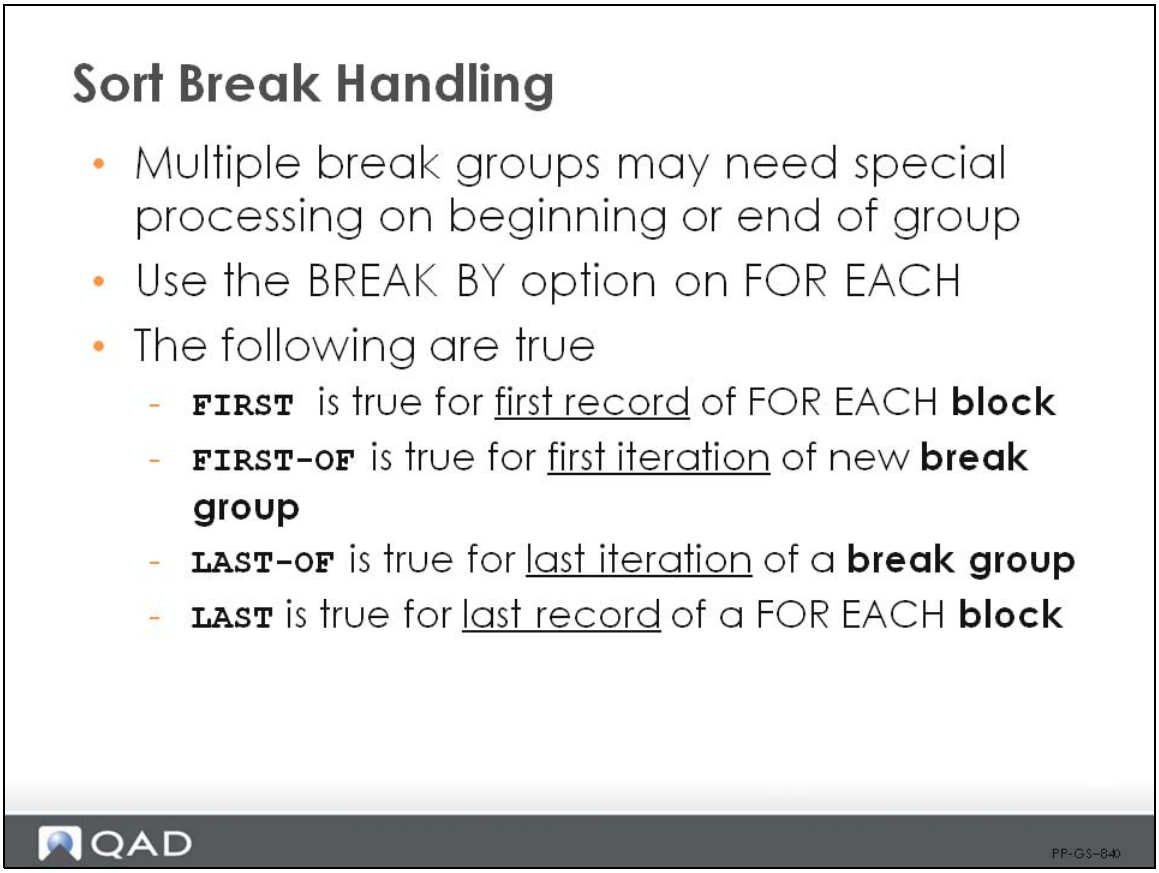
Exercise 5-5: Sort Breaks and Subtotals

1 Modify the procedure `pc05005.p` to subtotal the quantity ordered for each sales order.

2 Write a procedure to report location detail records (ld_det) by site and location.

- Display item number, site, location, quantity on hand, and quantity allocated
- Display subtotal of quantity on hand for each location
  Hint. Use ld_domain = "10USA" in the WHERE clause.

Sort Break Handling



When producing a report with multiple break groups by using the BREAK BY option on a FOR EACH statement, you may want to have special processing at the beginning or end of a break group. The FIRST, FIRST-OF, LAST, and LAST-OF functions are used to test records and their relationship to a break group.

Note You can only use FIRST, FIRST-OF, LAST, and LAST-OF when break groups have been specified.

- FIRST is true for the first iteration of a FOR EACH ... BREAK block.
- FIRST-OF is true for the first iteration of a new break group.
- LAST is true for the first iteration of a FOR EACH ... BREAK block.
- LAST-OF is true for the last iteration of a break group.
- Use FIRST and/or FIRST-OF to create special headings at the beginning of a break group.

If you have variables for storing totals, use LAST-OF and/or LAST to determine when to display these variables.

Exercise 5-6: Sort Break

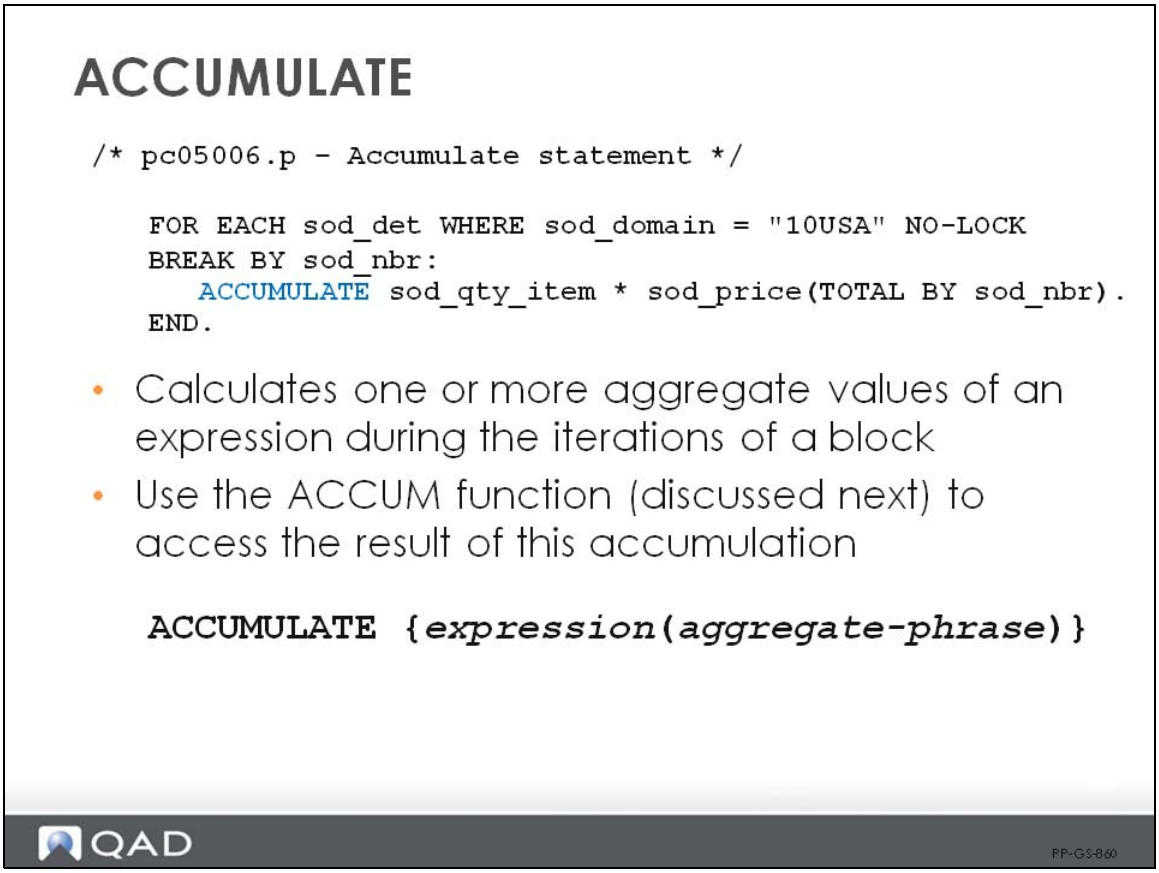
1 Write a procedure to display the customer number on the first line of a list of sales orders, then SKIP a line after the last order of that customer.

Hint. Use domain "10USA" in the WHERE clauses.

Optional Exercise

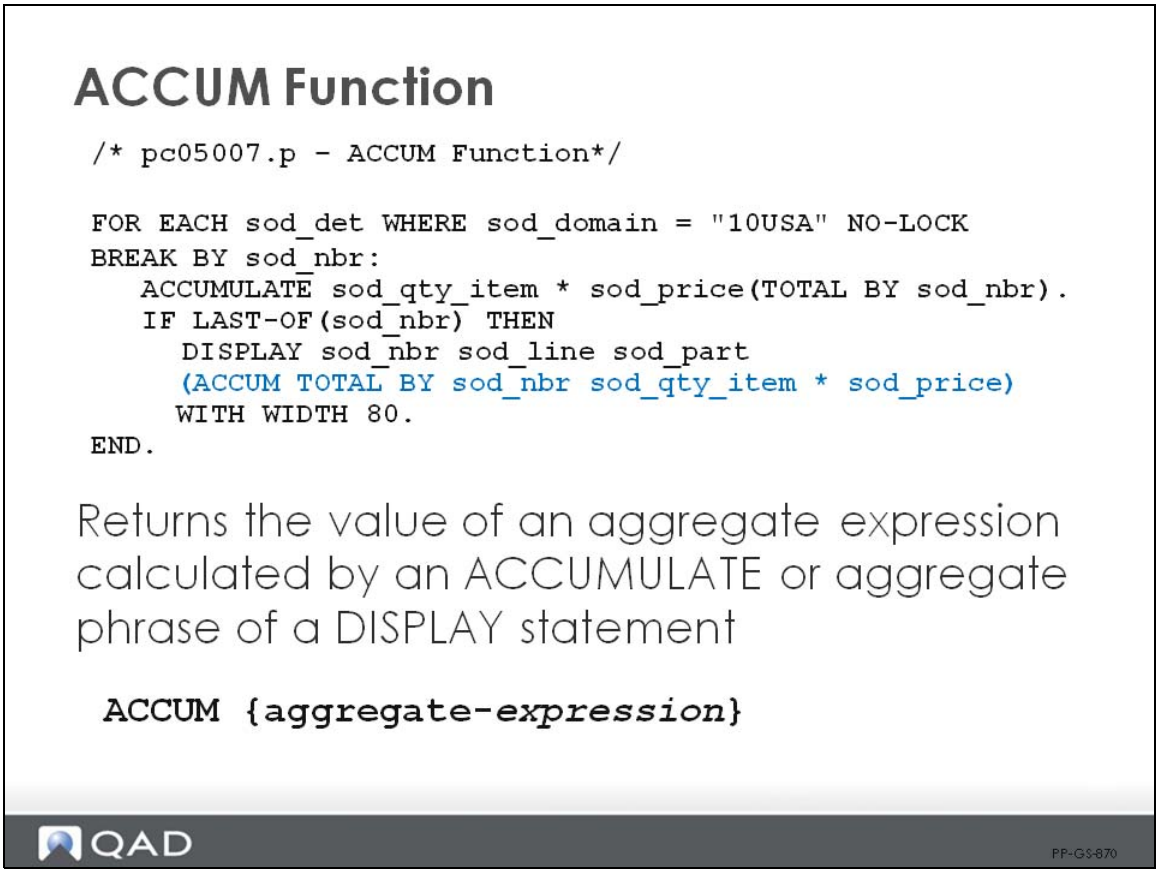
2 Write a procedure to display the different transaction types that have been written to the tr_hist table.

ACCUMULATE

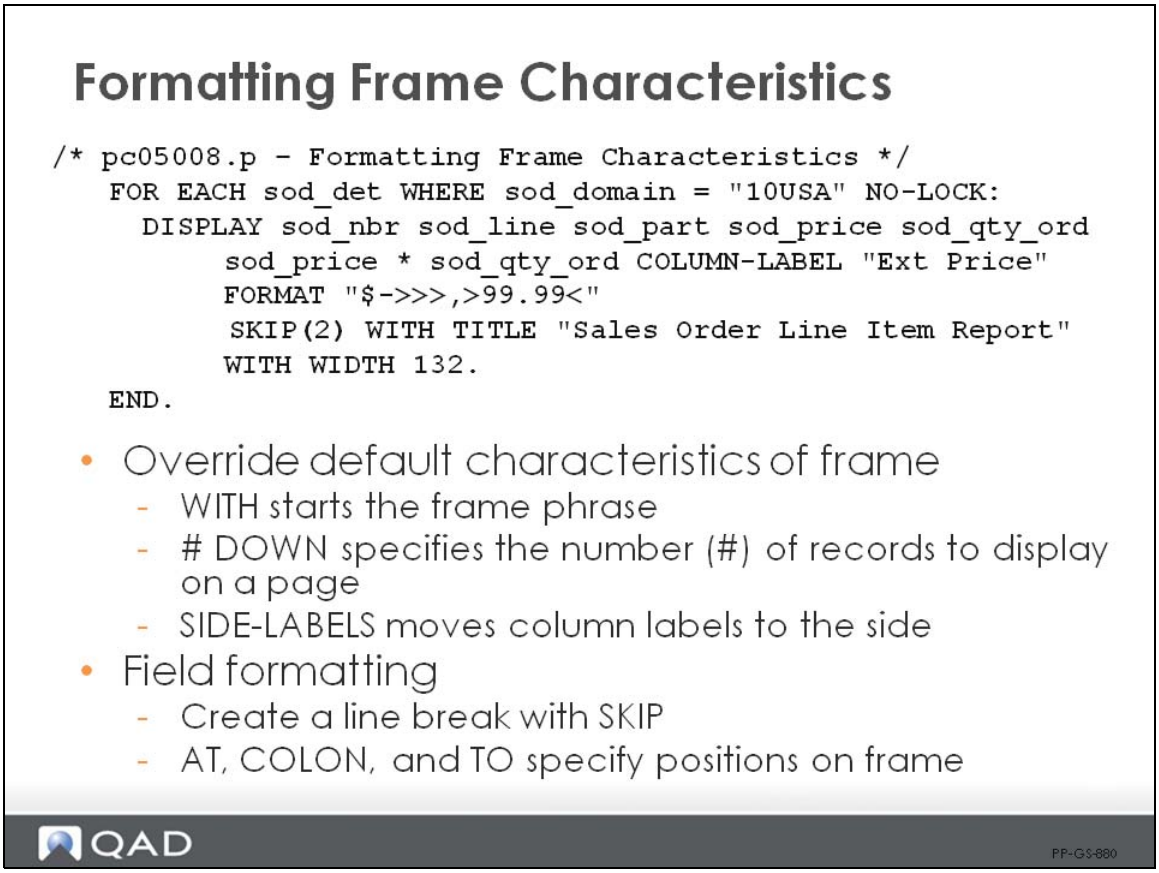


The ACCUM function is discussed on the next page.

ACCUM Function



Formatting Frame Characteristics



Progress automatically creates a new frame when it is needed to display data. A FOR EACH and REPEAT block creates a frame by default. You can override the default characteristics for the box or frame created by Progress in your procedure using the frame phrase. The frame phrase determines the overall layout and characteristics of a frame.

Note It is preferable to use the following features within a FORM statement defining the frame outside the FOR EACH block. See “Sample Inquiry” on page 9 for an example.

- TITLE “string” causes the “string” to be printed within the top line of the frame box.
- CENTERED tells Progress to center the display box on the screen.
- Progress does not limit you to a column format. In many cases, a column format is not sufficient, for example, when you have too many columns to fit across the screen. SIDE-LABELS causes the label to print to the left of each field instead of above. 2 COLUMNS specifies to print two fields per line.
- A SKIP can be inserted after a field (or expression) and will advance to the next line before displaying the following field (or expression). This is like a line break in a word processor. SKIP(2) requests that two blank lines be left between each displayed line.
- # DOWN specifies the number (#) of item records that are to appear on each page of the display. A frame is called a down frame because it can display multiple rows of data as it moves down the page.

Exercise 5-7: Formatting Frame

1 Experiment with the following frame phrase options within procedure `pc05008.p`:

- CENTERED
- NO-UNDERLINE
- NO-BOX
- WIDTH 80
- 2 COLUMNS
- THREE-D

2 Add COLUMN 15 SIDE-LABELS to the frame phrase. Experiment with the AT and COLON format phrase options.

3 Format extended price to display a pound sign or dollar sign. Format "$->>>,>99.99<"

4 Modify this procedure:

From: `FOR EACH sod_det WHERE sod_domain = "10USA" NO-LOCK`
To: `FOR EACH sod_det WHERE sod_domain = "10USA" NO-LOCK WITH 3 DOWN:`

5 Create a procedure to produce the following display.

Hint. Use the AT or COLON format phrase options to position information on the screen. Use pt_domain = "10USA" in the WHERE clause.

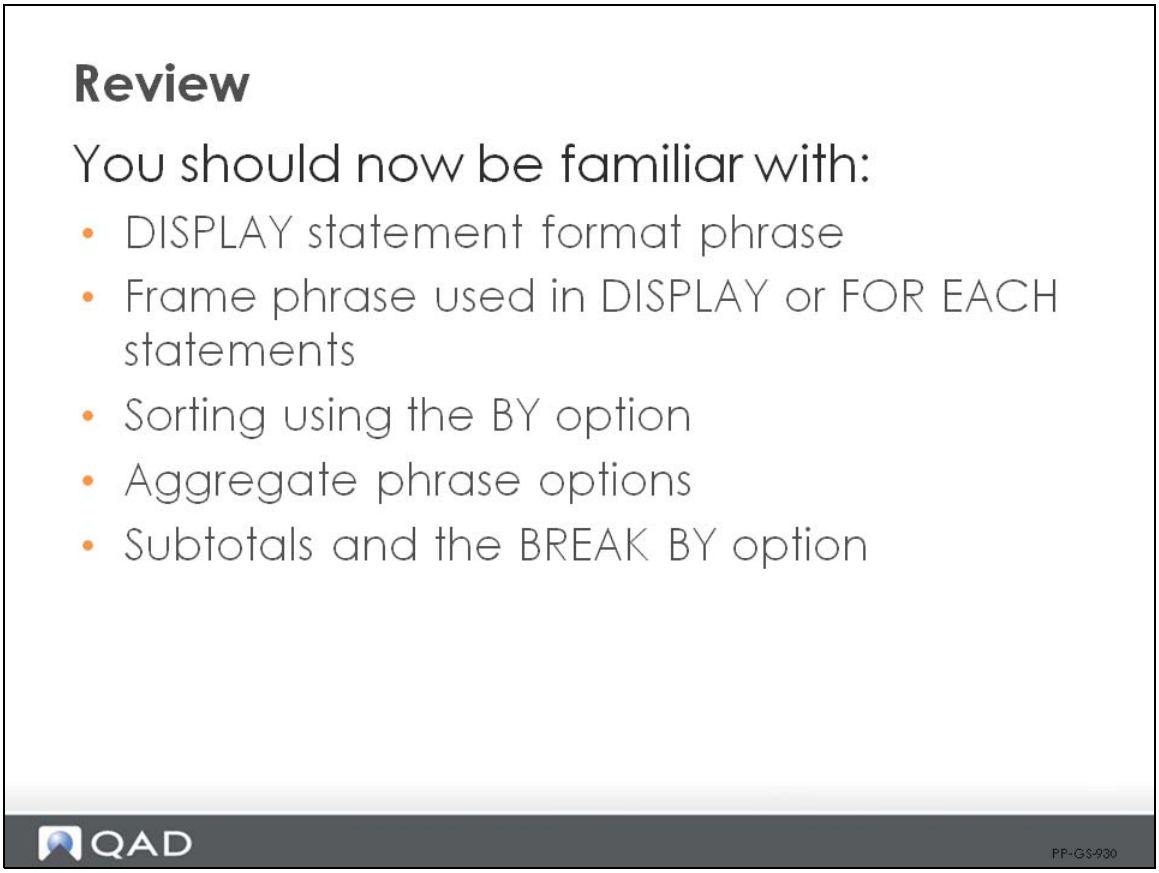
```
                        I T E M S   C A T A L O G

Item Number   :   XXXXXXXXXXXXXXXXXX          Price:   $99.99
Description   :   XXXXXXXXXXXXXXXXXXXXXXXX
              :   XXXXXXXXXXXXXXXXXXXXXXXX

Item Number   :   XXXXXXXXXXXXXXXXXX          Price:   $99.99
Description   :   XXXXXXXXXXXXXXXXXXXXXXXX
              :   XXXXXXXXXXXXXXXXXXXXXXXX

Item Number   :   XXXXXXXXXXXXXXXXXX          Price:   $99.99
Description   :   XXXXXXXXXXXXXXXXXXXXXXXX
              :   XXXXXXXXXXXXXXXXXXXXXXXX
```

Review



In this lesson, you have learned how to use the FOR EACH and DISPLAY statements. You have learned how to use different options to control how and where fields are displayed. You then learned different techniques for sorting records.

- Use the format phrase within the DISPLAY statement to modify the way in which individual expressions were displayed.
- Use the frame phrase at the end of the DISPLAY or FOR EACH statement to modify overall frame characteristics.
- Use the BY option in the FOR EACH statement to sort records based on the value of any field or expression.

Various statistics, such as a report total, that can be calculated using aggregate phrase options. You must put the aggregate phrase after the field/expression in a DISPLAY statement. Subtotals are produced by adding the BREAK BY option to the FOR EACH statement and the aggregate phrase in the DISPLAY statement.

The BREAK BY option with the FOR EACH statement permits the use of the functions FIRST, FIRST-OF, LAST, and LAST-OF.

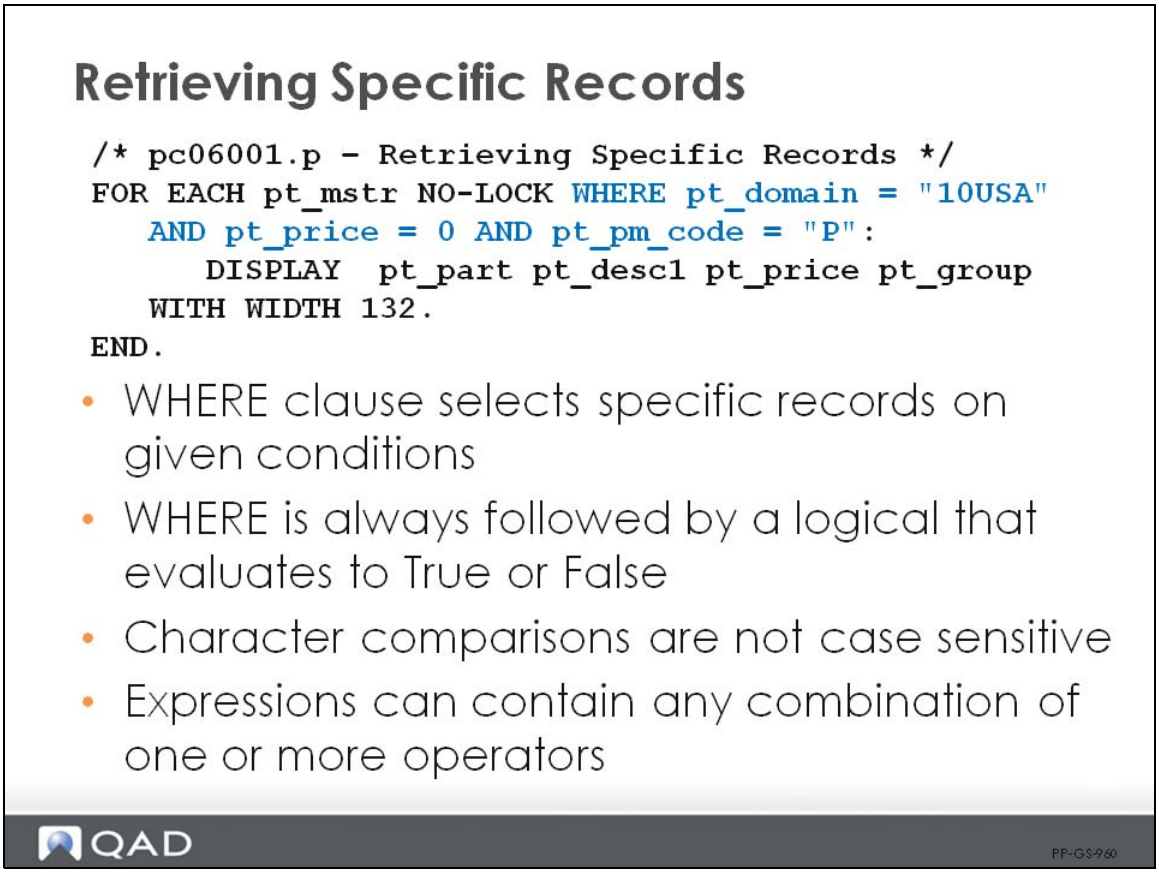
Lesson 6: Selecting Specific Records

In the previous lessons, the FOR EACH and DISPLAY statements were used to display all records within a file. This lesson covers the selection of specific records for use in the procedure.

Upon completion of this lesson, you will be familiar with the following topics:

- Selecting specific records using the record phrase in the FOR EACH statement
- The BEGIN and MATCHES Progress functions
- Conditional expressions using Progress operators
- Index specification and record selection
- Conditional processing using IF ... THEN ... ELSE and the CASE statement
- Requesting user input with the PROMPT-FOR statement

Retrieving Specific Records



By using the WHERE clause within a record phrase, you can select records that satisfy a given condition when using the FOR EACH statement. The WHERE keyword is always followed by a LOGICAL expression that is always true or false.

The definition of a LOGICAL expression is any constant, field name, variable name, or any combination of these, separated by an operator. For example, in this expression:

```
pt_price = 0
```

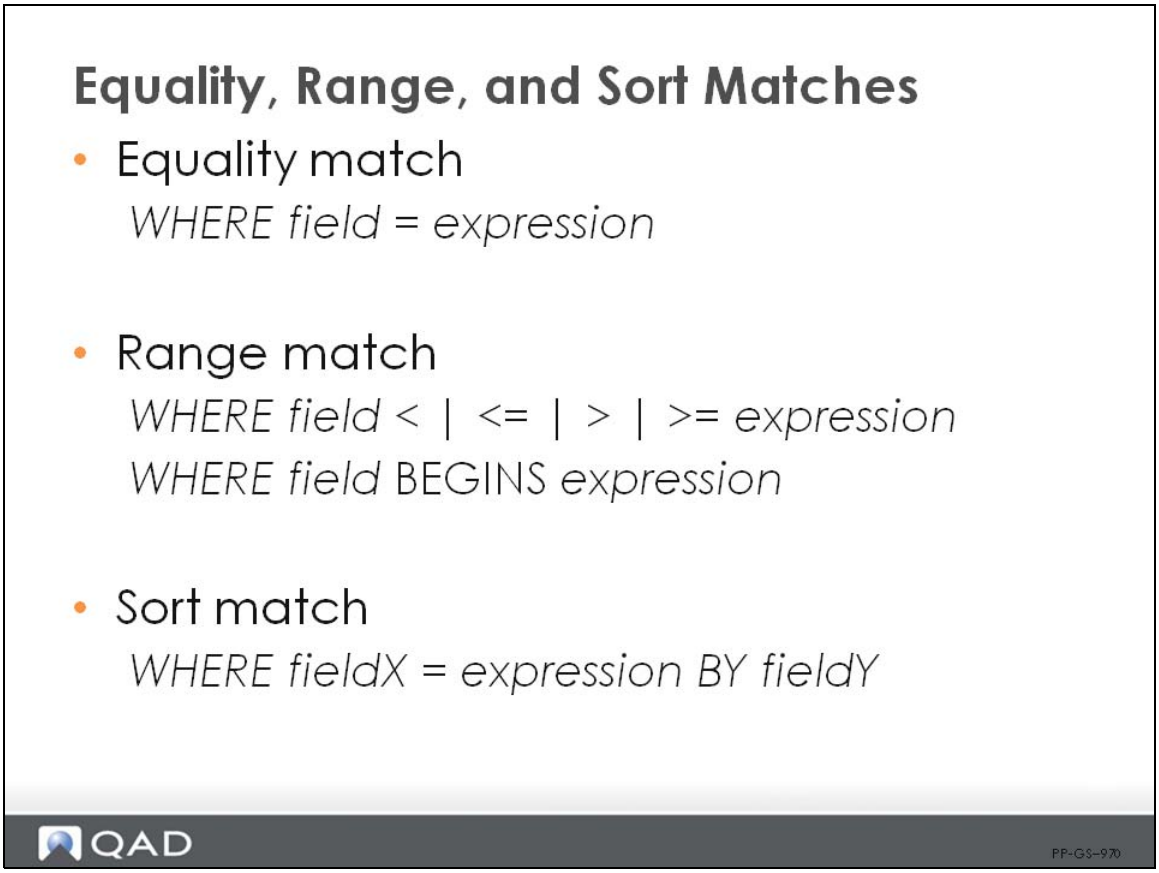
- pt_price is the field name
- = is the operator
- 0 is a constant

Note The QAD standard is to not use alphabetic operators such as LT and LE.

Character comparisons in Progress are not case sensitive. Expressions can contain any combination of one or more Progress operators. Here are some useful ones:

Addition	+	Less than	LT or <
Subtraction	–	Less than or equal to	LE or <=
Multiplication	*	Greater than	GT or >
Division	/	Greater than or equal to	GE or >=
		Equal to	EQ or =
		Not equal to	NE or <>

Equality, Range, and Sort Matches



Equality Match. WHERE field is exactly the same as a value or expression.

Range Match. WHERE field is greater than, less than, greater than or equal to, less than or equal to a value.

Sort Match. Select based on X criteria, but sort based on field Y using the BY word. This means that Progress cannot find an index to use based on the fields in the WHERE clause, but one can be found based on the sort criteria.

Exercise 6-1: Retrieving Specific Records

1 Modify procedure `pc06001.p`:

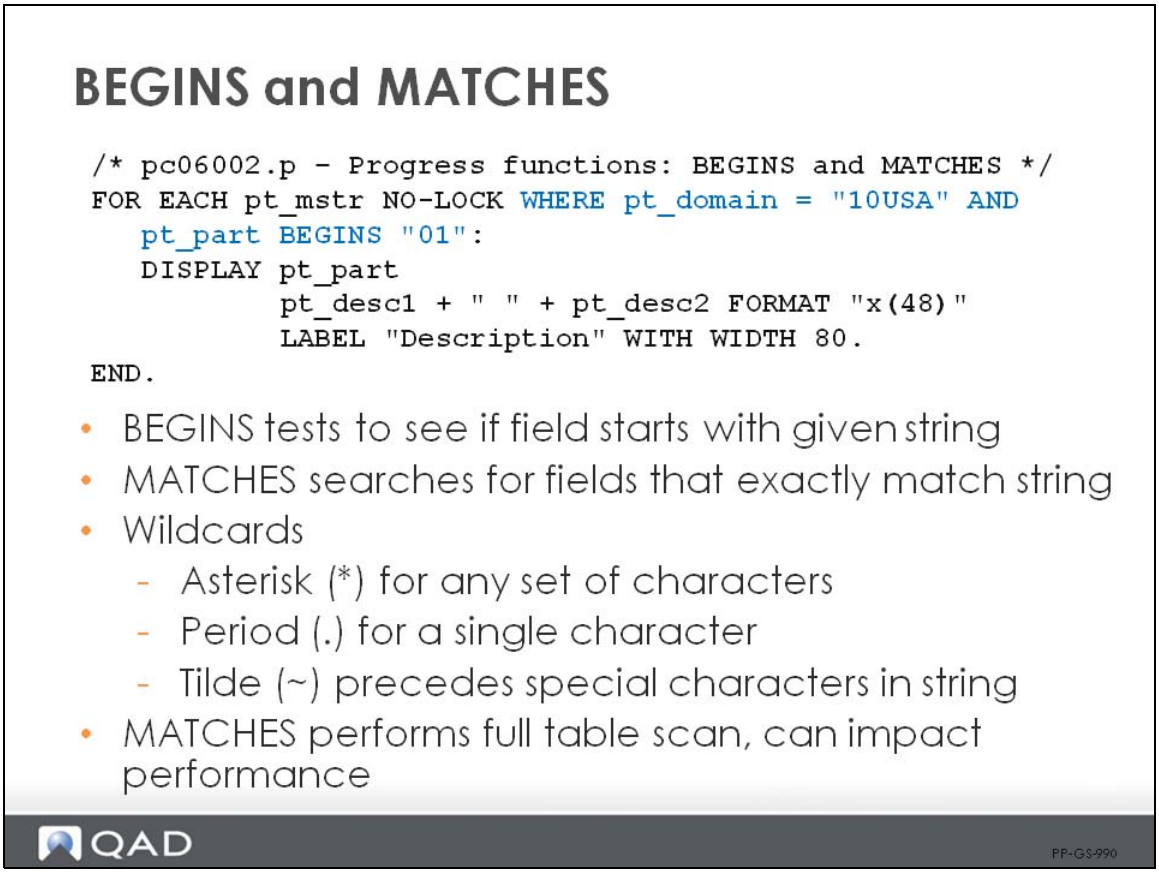
From: `pt_price = 0`
To: `pt_price <> 0`

From: `pt_price <> 0 AND pt_pm_code = "P"`
To: `pt_price = 0 AND pt_pm_code = "M"`

2 Modify this procedure to display only items in the product line 10.

Hint. Character constants must be enclosed within quotation marks.

BEGINS and MATCHES



Functions are used to evaluate fields, constants, and other expressions. Often logical expressions do not provide enough flexibility. You may want to specify only those records beginning with or containing a certain string of characters. This cannot be done easily with operators such as EQ or GT, but it can be accomplished by using functions within an expression.

- BEGINS tests a character field to see if it starts with the given character string. Progress uses an index when it uses the BEGINS function on an indexed field.

  Example BEGINS “John” will find “Johnson” and “John Smith.” The query will not find “J Johnson & CO.”

- MATCHES searches for records containing fields that exactly match specified strings.

Warning Progress does not use an index with the MATCHES function, even when used on an indexed field. It always scans the entire table. On large tables, this can be a significant performance issue.

You can use wildcards with the MATCHES command. An asterisk (*), when used as a wildcard, stands for any set of characters, in the same way it is used in the DOS and UNIX operating systems. In this example, if pt_part MATCHES “*01*” replaces pt_part BEGINS “01”, any item number with the number 01 anywhere in it would be chosen. This would include 50010, 01010, and 02001. It would not include 02102, since the string 01 must be matched.

The asterisk (*) is used for multiple characters, and the period (.) is used for a single character. If you need to refer to a special character within a string, it should be preceded by a tilde (~).

Example: `MATCHES "*~"*:` will find strings with a double-quote (") character.

Exercise 6-2: BEGINS and MATCHES

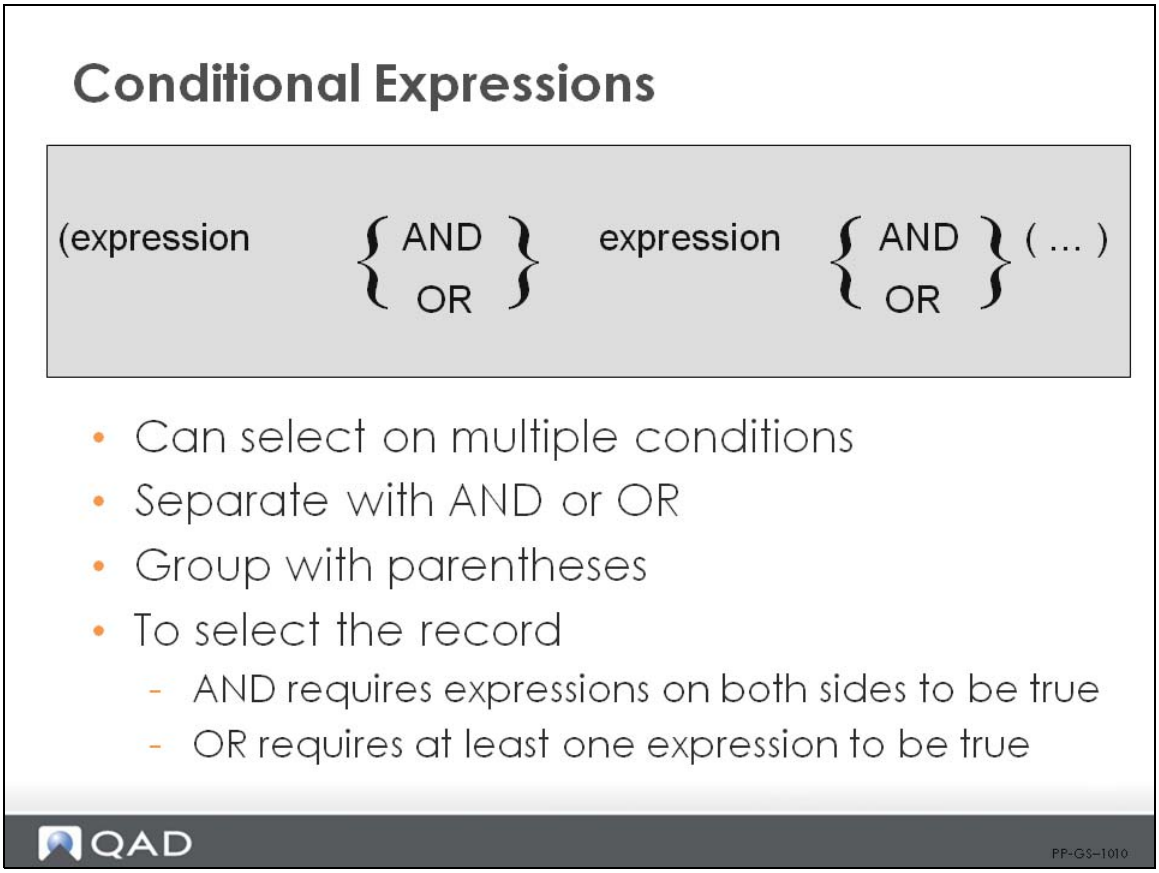
1 Modify the procedure `pc06002.p` and run:

From:	`BEGINS "01"`
To:	`BEGINS "0"`
From:	`BEGINS "0"`
To:	`MATCHES "*01*"`

2 Often companies have the same or similar item description with very different item numbers. A useful application of the MATCHES function is to display all of the items with a certain word in their description.

Modify the procedure to display all items with the word “PACK” in their description.

Conditional Expressions



You can select records based on multiple conditions separated by AND or OR.

- The operator AND requires the expressions on both sides to be TRUE for a record to be selected.
- The operator OR requires that at least one of the expressions be TRUE in order for the record to be selected.

Expressions inside parentheses are evaluated first.

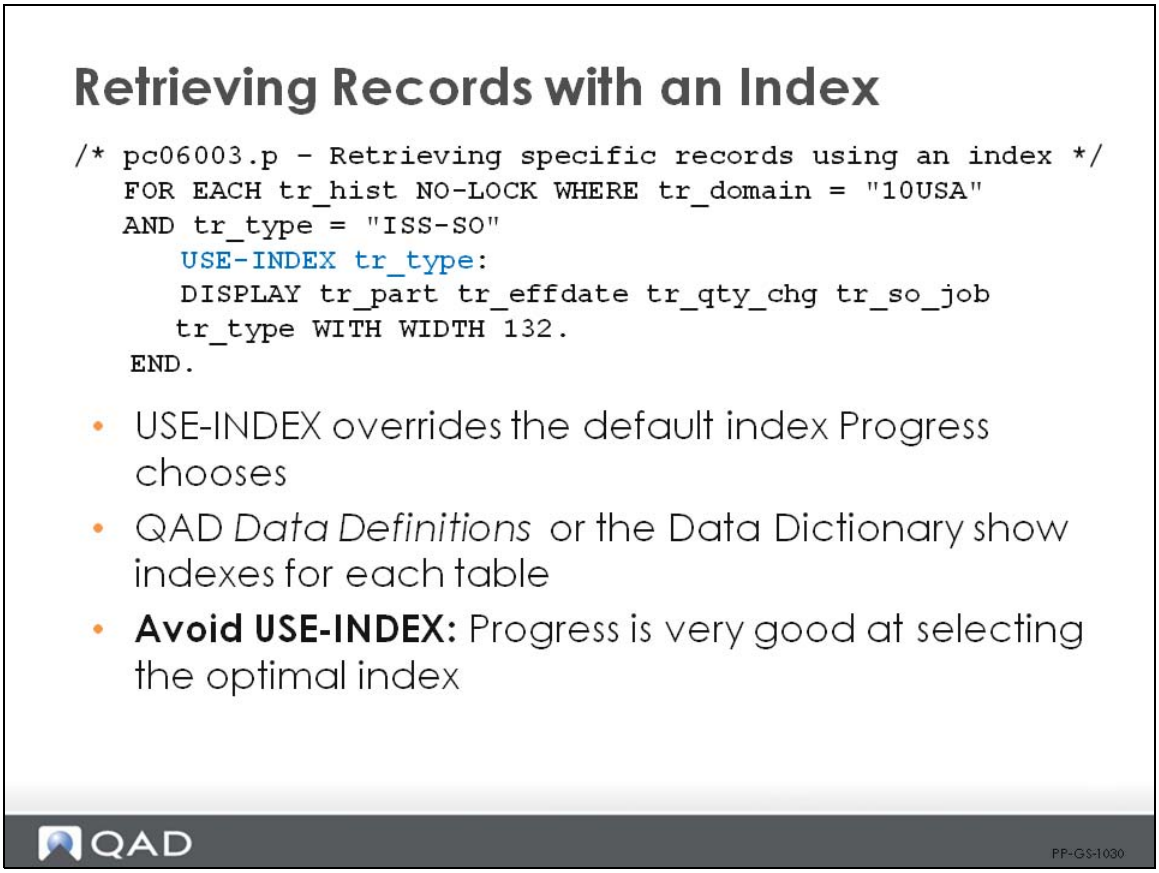
Exercise 6-3: Conditional Expressions

Evaluate the following statements as True or False. Use the following values for the respective field names:

```
pt_run = 2          pt_setup = 1          pt_sfty_time = 3
```

1. `pt_run > 0 AND (pt_run > pt_setup AND pt_run > pt_sfty_time`
2. `pt_run > 0 AND (pt_run > pt_setup OR pt_run > pt_sfty_time)`
3. `pt_run LE 0 OR pt_setup < pt_sfty_time`
4. `(pt_run > 0 AND pt_run > pt_setup) OR pt_run > pt_sfty_time`
5. `((pt_run > 0 and pt_run < pt_setup) AND (pt_setup < pt_sfty_time)) AND pt_run > pt_sfty_time`

Retrieving Records with an Index

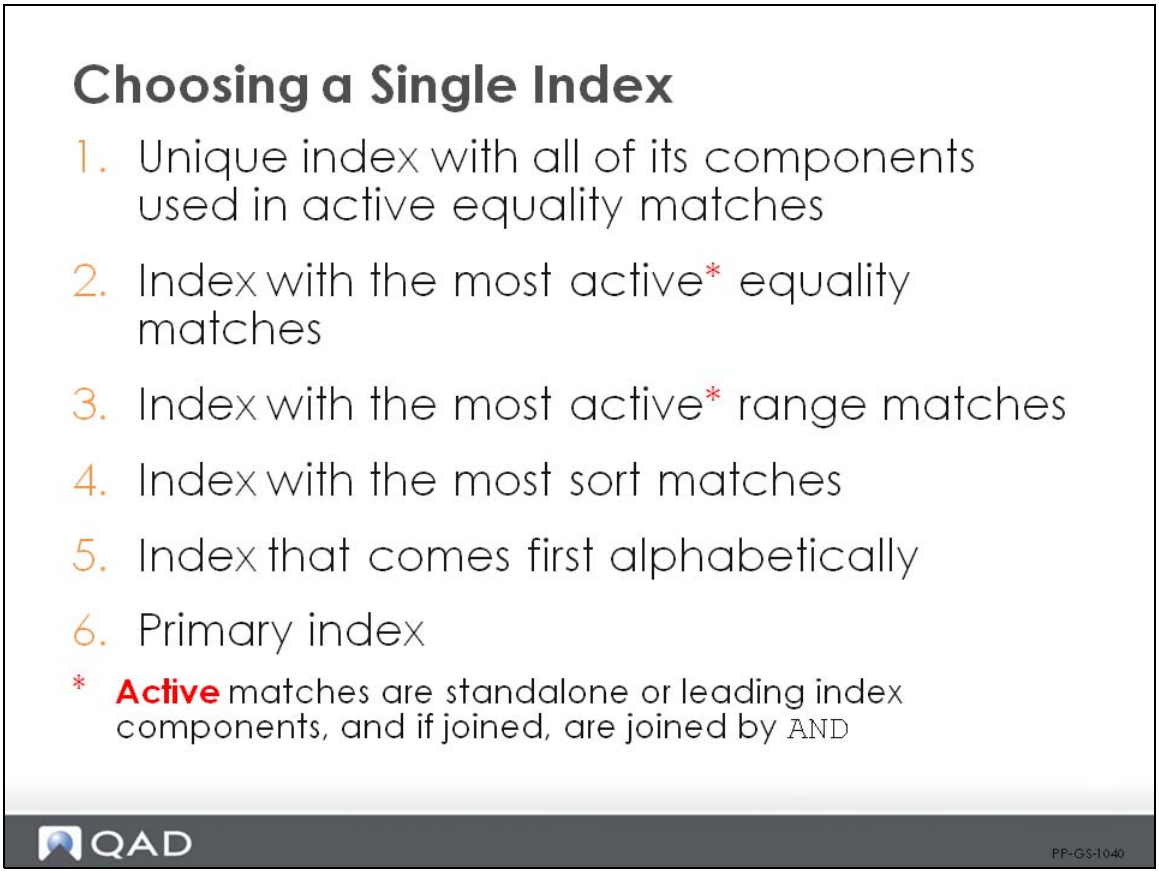


Progress selects the best index to use based on the fields specified in a properly formed WHERE clause and [BREAK] BY options. In most cases, the primary index for the given table is used. Refer to the Progress documentation under the FOR statement for more details.

Important Progress is very good at choosing the best index. While it is possible to override the Progress default with the USE-INDEX clause, this option is discouraged and you should only use it on the rare occasion when an exception is needed.

Table index names are listed in the *Data Definitions* manual at the end of each table listing.

Choosing a Single Index

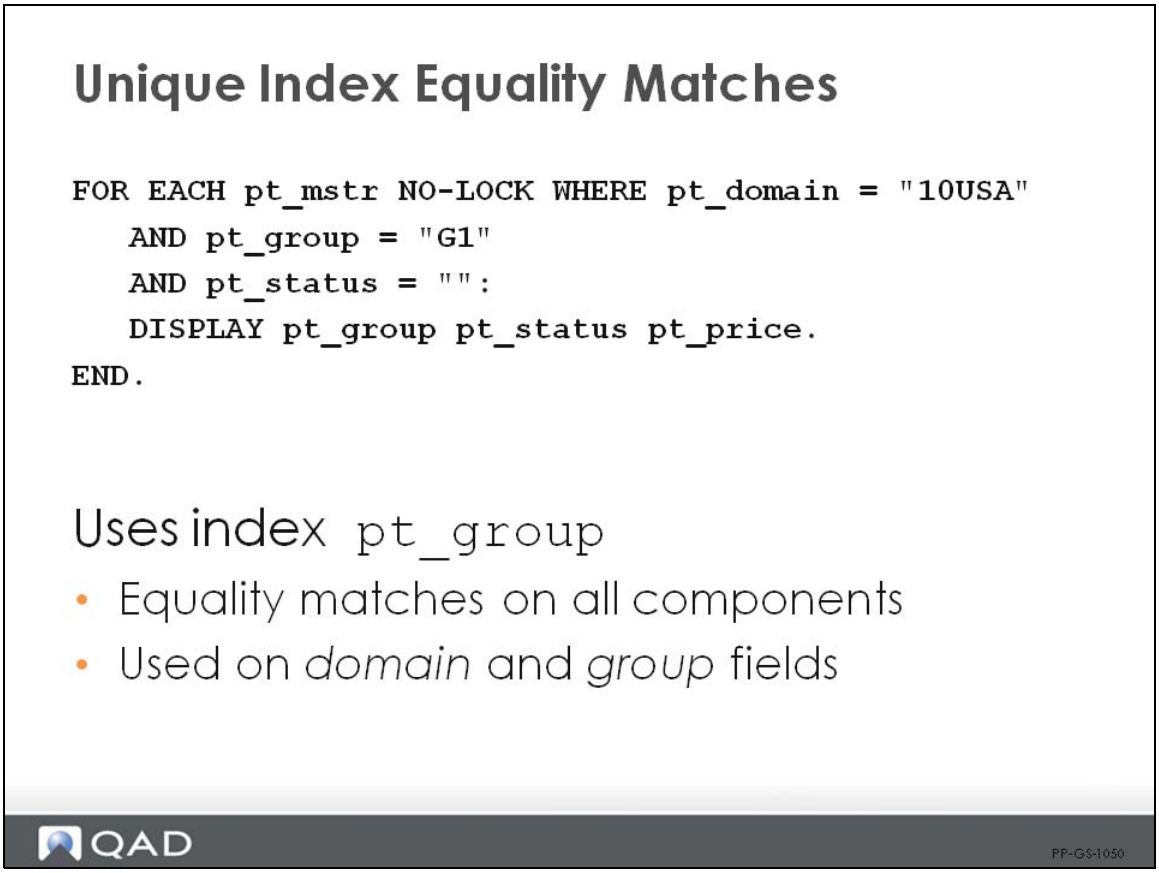


This is the order in which Progress selects an index for any lookup if the index is not mandated with the USE-INDEX option.

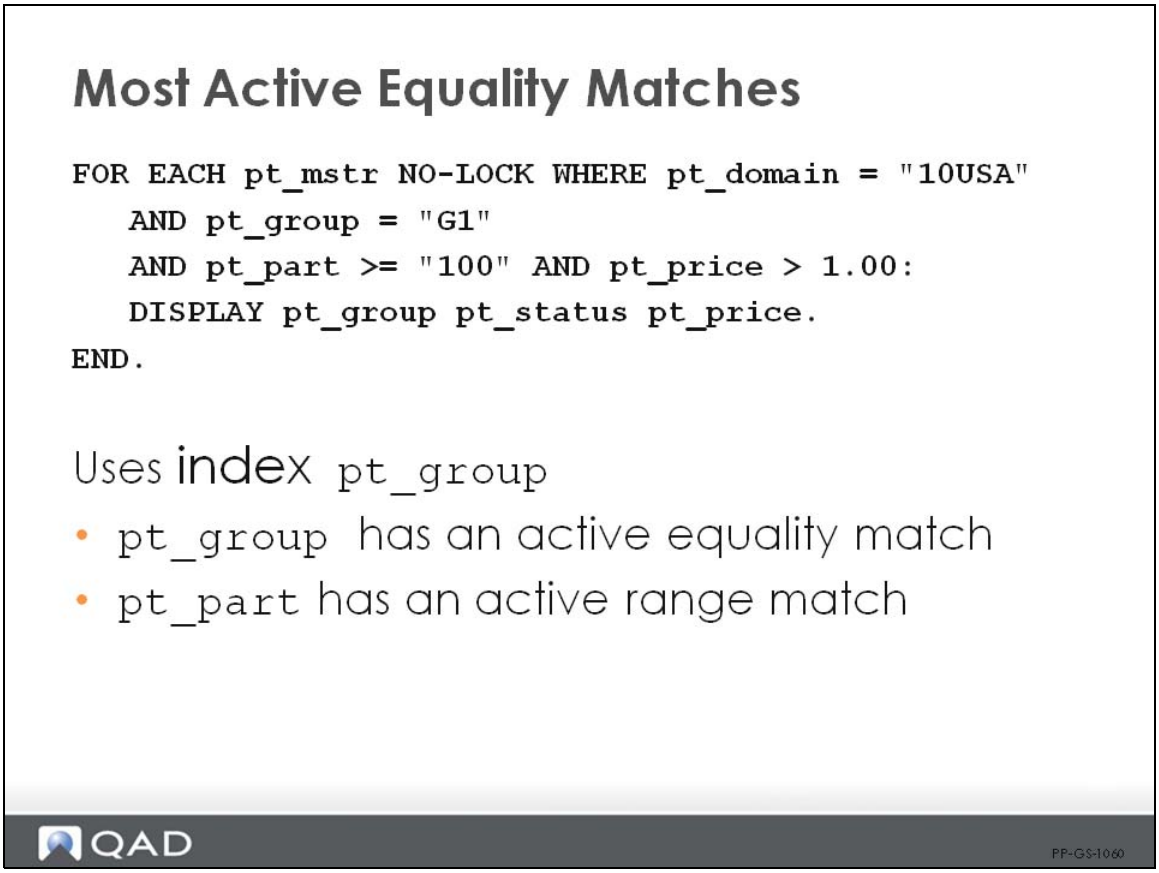
Important Progress is very good at choosing the best index. USE-INDEX is discouraged and you should only use it on the rare occasion when you need to override the Progress default.

1. If one index is unique and all of its components are involved in active equality matches and the other index is not unique, or if not all of its components are involved in active equality matches, ABL chooses the former of the two.
2. Select the index with more active equality matches.
3. Select the index with more active range matches.
4. Select the index with more active sort matches.
5. Select the first index alphabetically by index name.
6. Select the index that is the primary index.

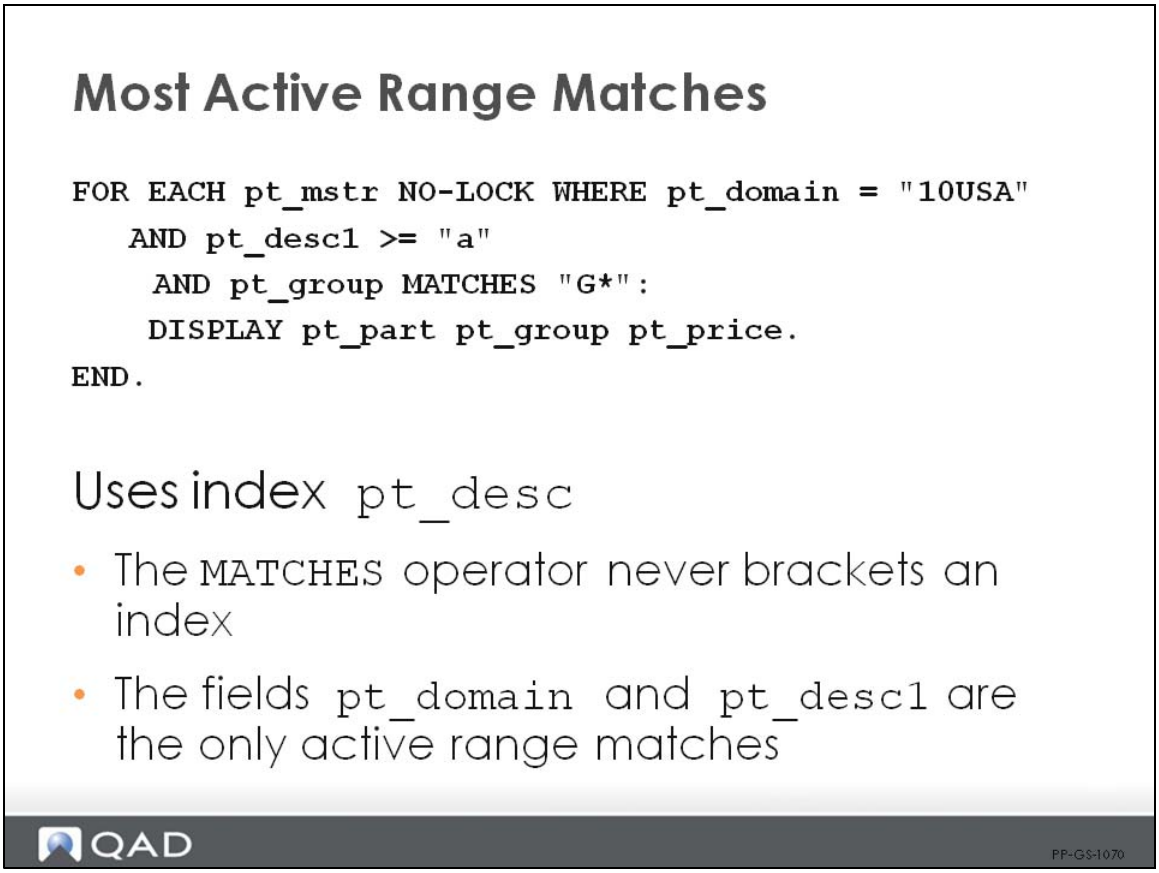
Unique Index Equality Matches



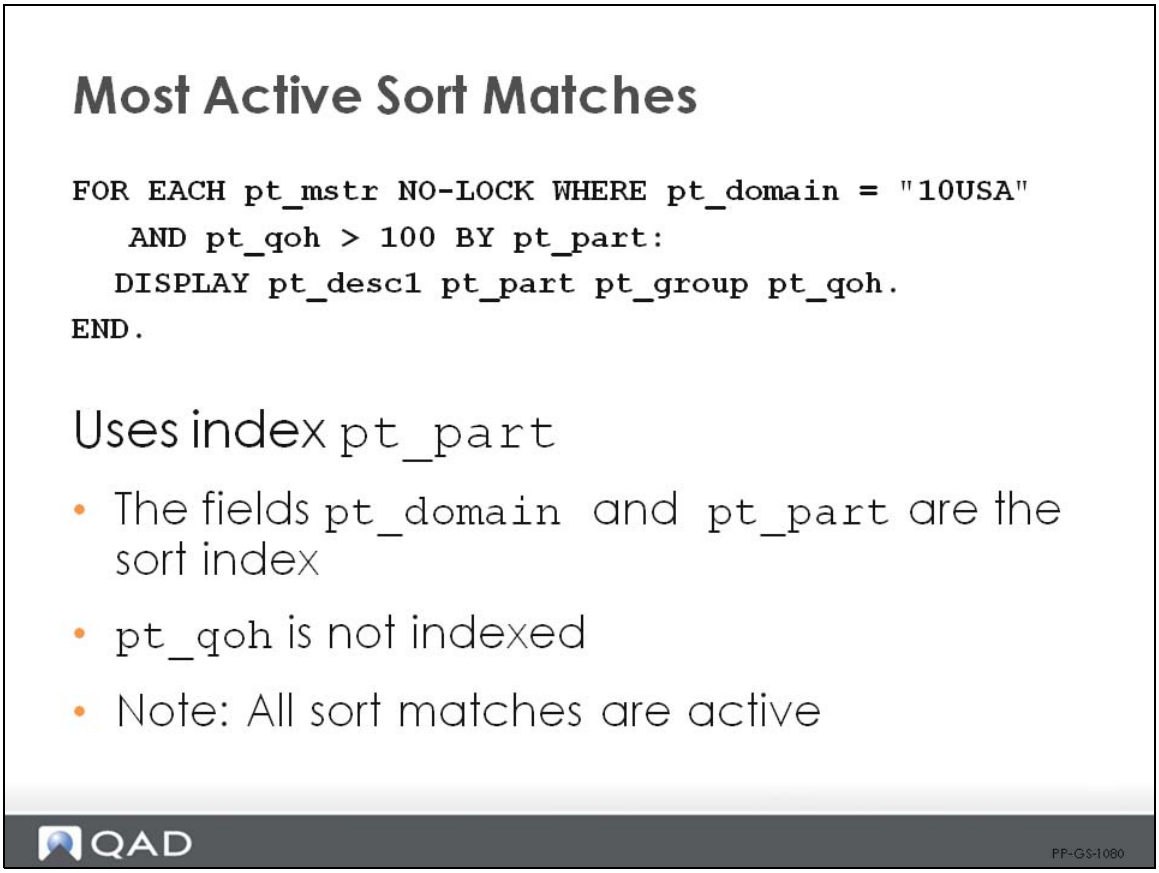
Most Active Equality Matches



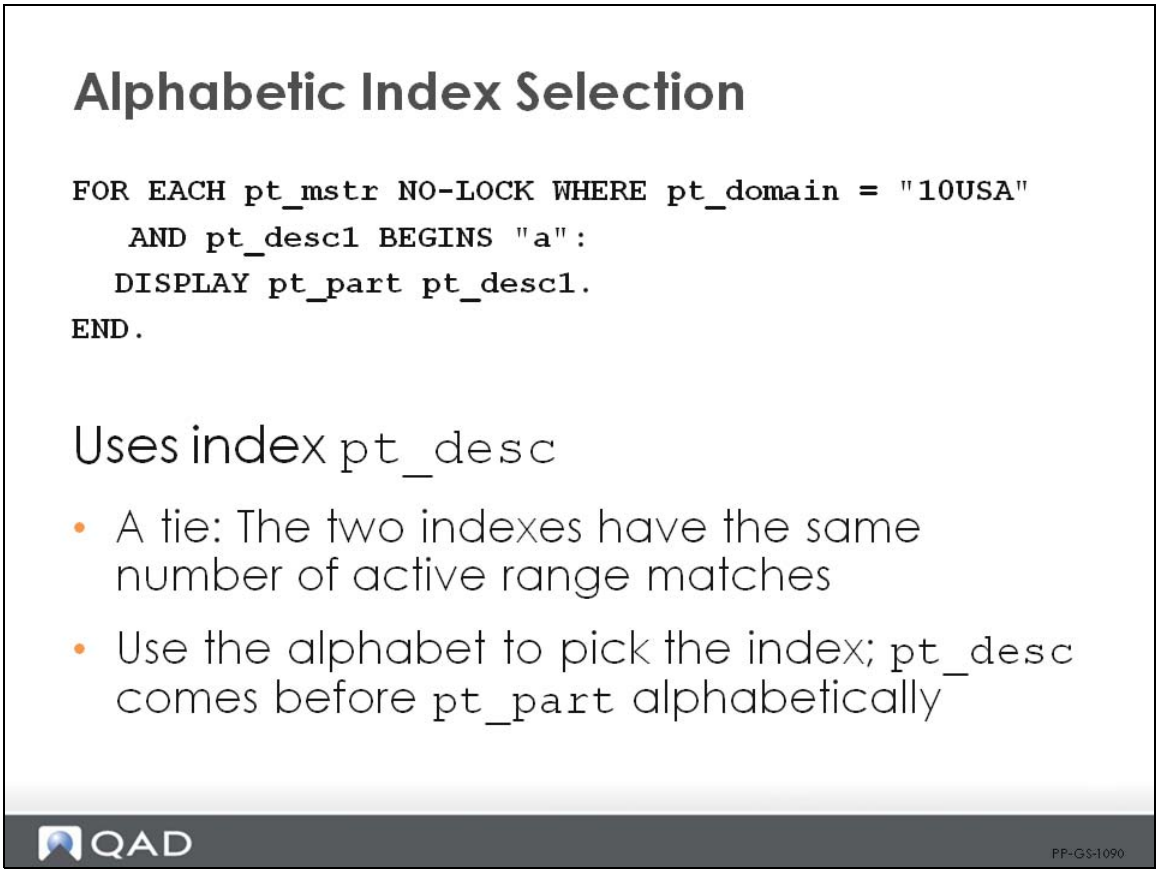
Most Active Range Matches



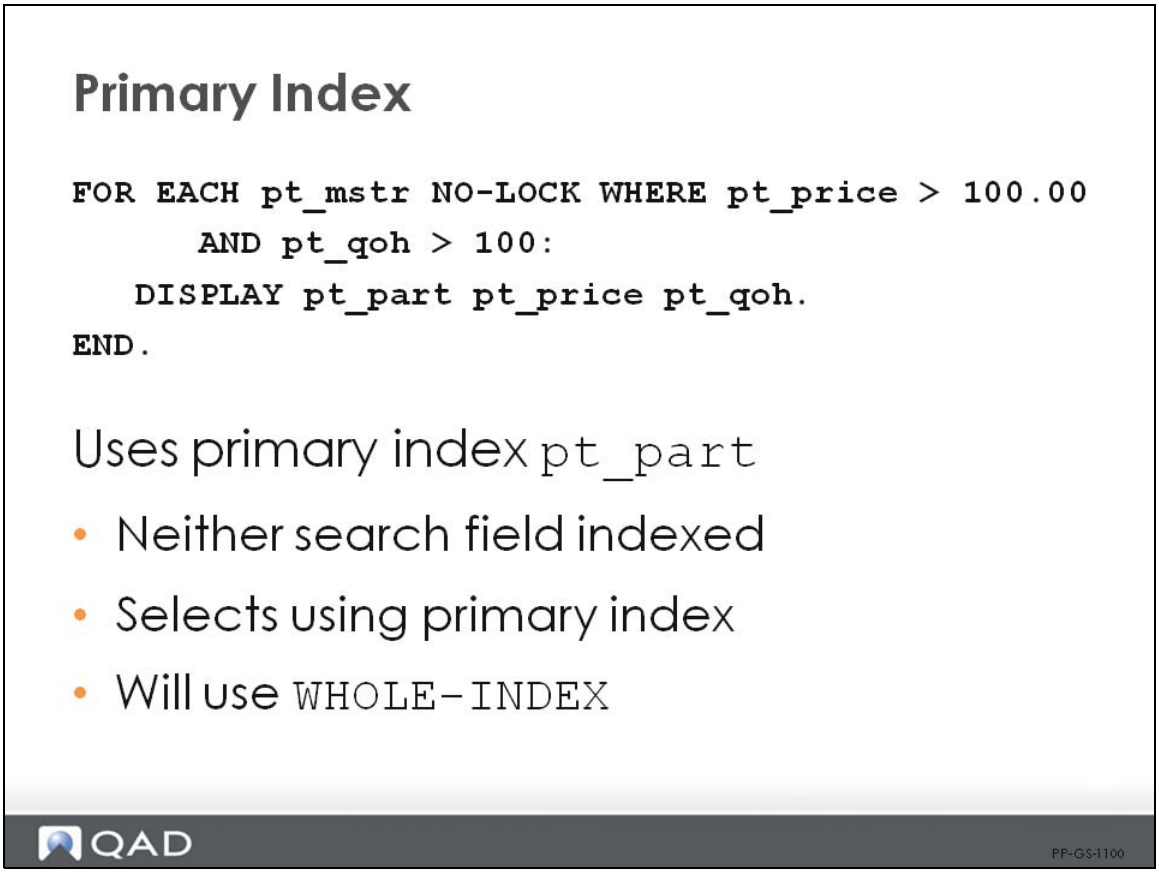
Most Active Sort Matches



Alphabetic Index Selection

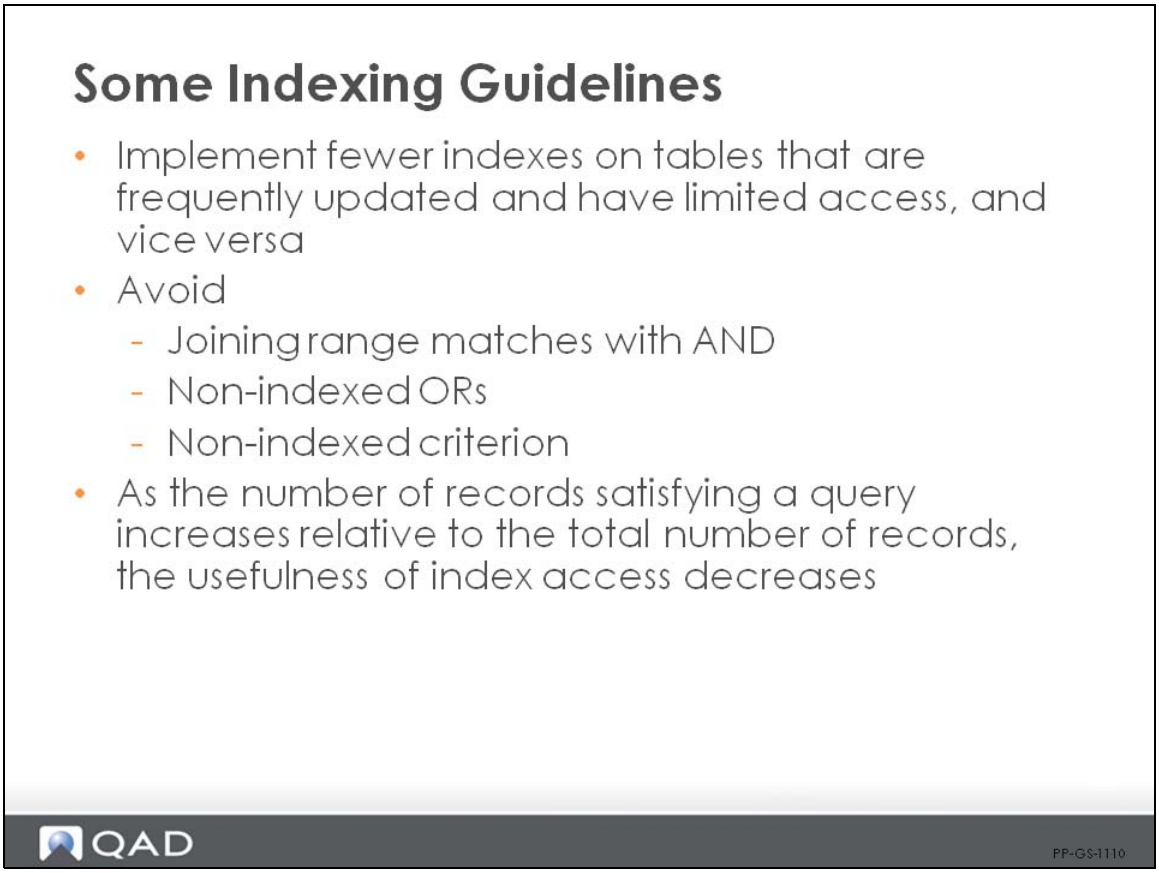


Primary Index



A WHOLE-INDEX search reported for a table occurs when an entire index is used to search the table. That is, the bracket used by the query to search the table spans the entire index. This can occur when no selection criteria are specified to limit the range of index keys searched (that is, to bracket a subset of the index) or when no appropriate index is available to optimize the selection criteria.

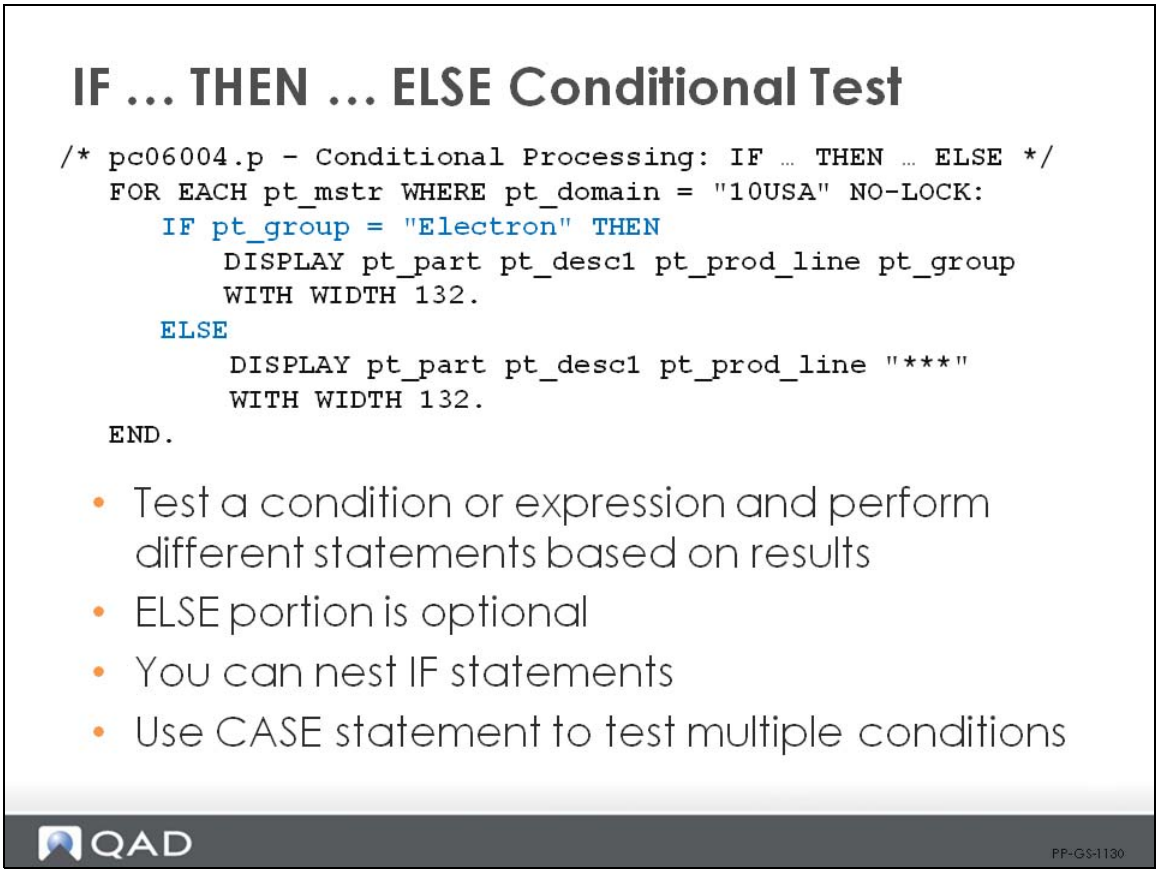
Some Indexing Guidelines



Exercise 6-4: Using Indexes

Modify the procedure `pc06003.p` to sort the output by item number.

IF ... THEN ... ELSE Conditional Test



The IF...THEN...ELSE structure lets you test a condition or expression and perform a different set of statements based on the result of a test.

If the value of the expression following the key word IF is true, Progress processes the statements following the key word THEN. Otherwise, Progress processes the statements following the key word ELSE.

The ELSE portion of the statement is optional.

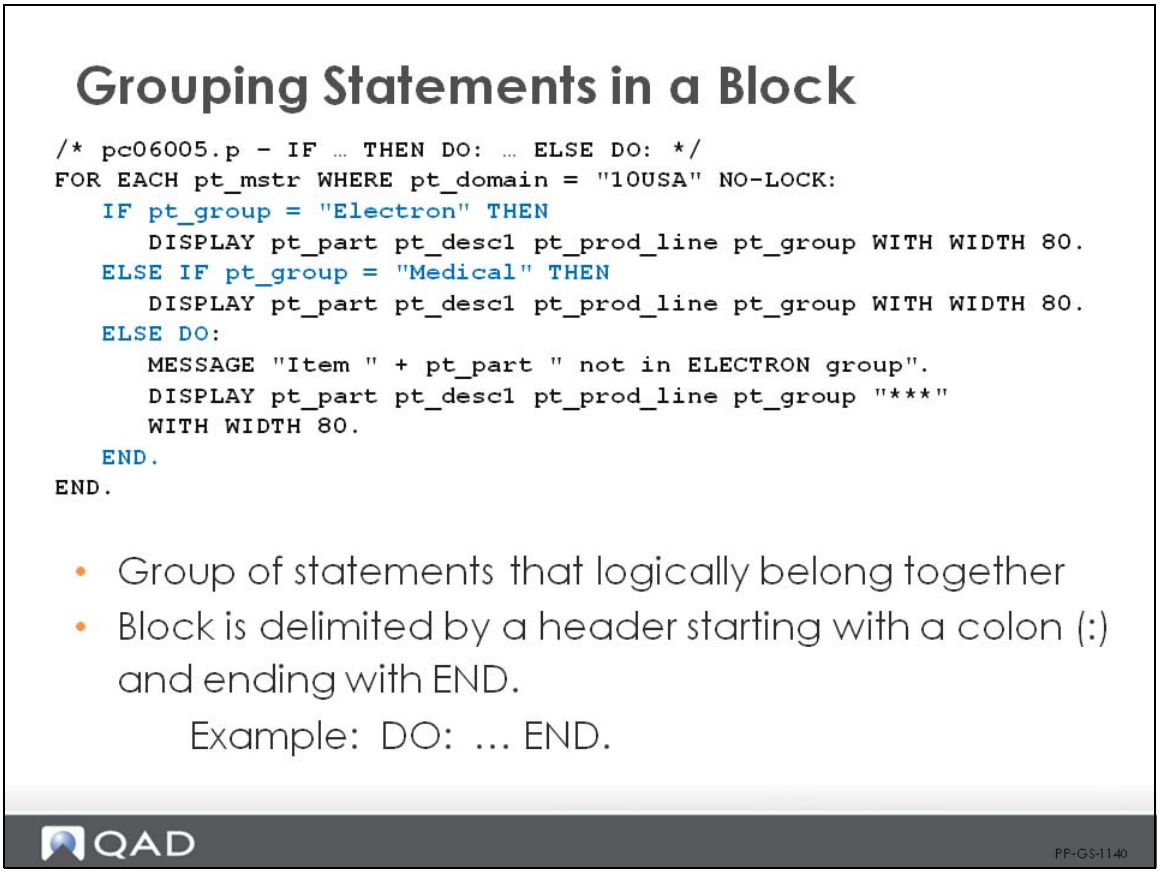
No keyword indicates the end of the IF statement. By default, only the statement immediately after the IF or ELSE is processed.

Exercise 6-5: IF...THEN...ELSE

Procedure `pc06004.p` highlights items that do not belong to a group.

1. Modify this procedure to highlight those items with a price = 0.
2. Modify the procedure by deleting the ELSE statement.
3. Add a title for this inquiry.

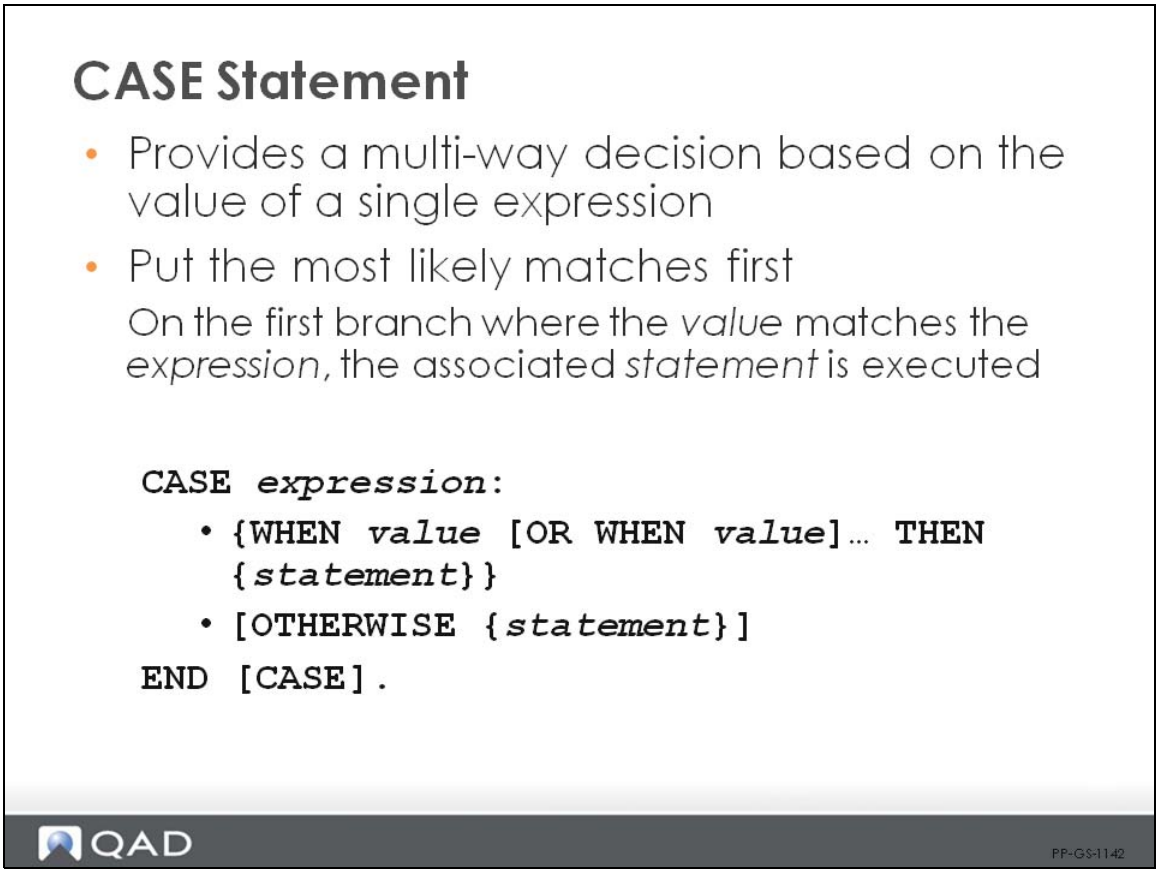
Grouping Statements in a Block



If more than one statement needs to be processed due to this conditional branching, encase it in a DO… END so that a block of statements can follow THEN or ELSE.

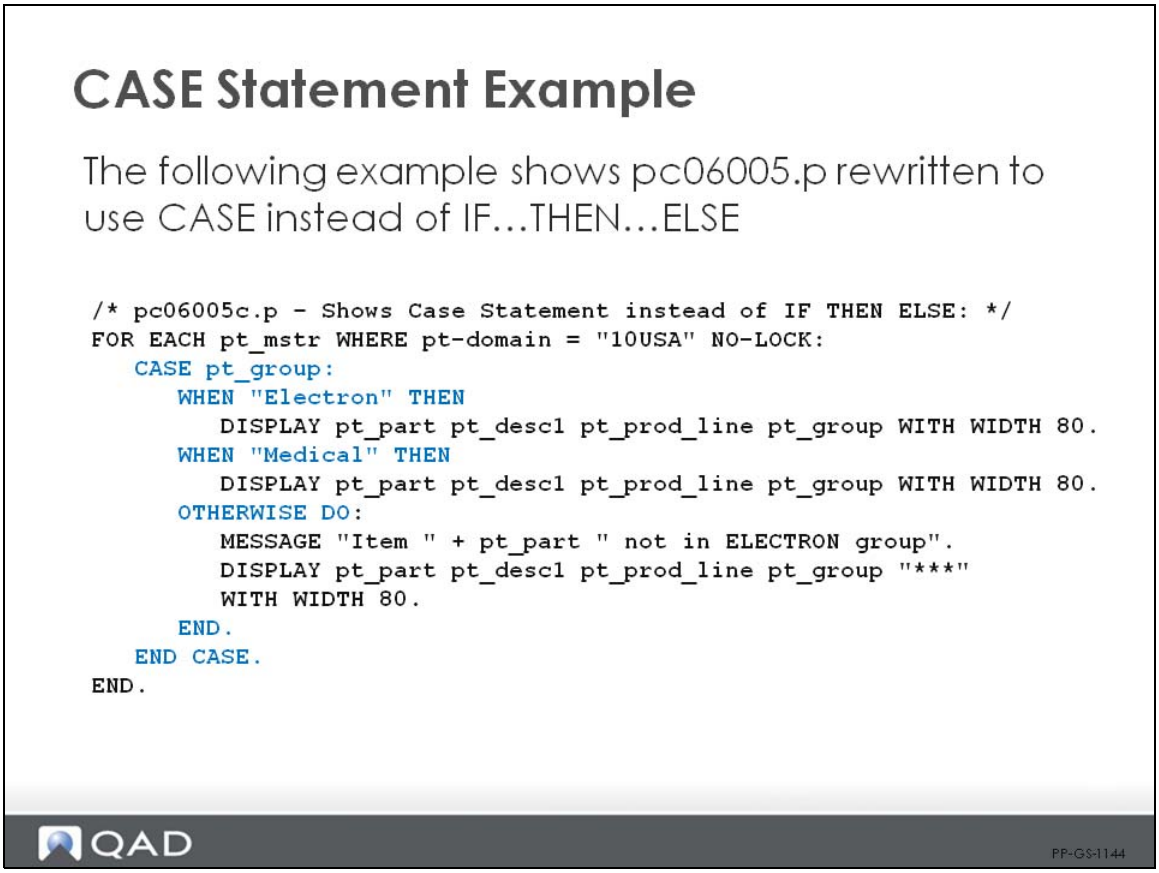
A THEN DO: or ELSE DO: starts a block that contains more than one Progress statement. The block starts even if there is another IF statement before the END.

CASE Statement

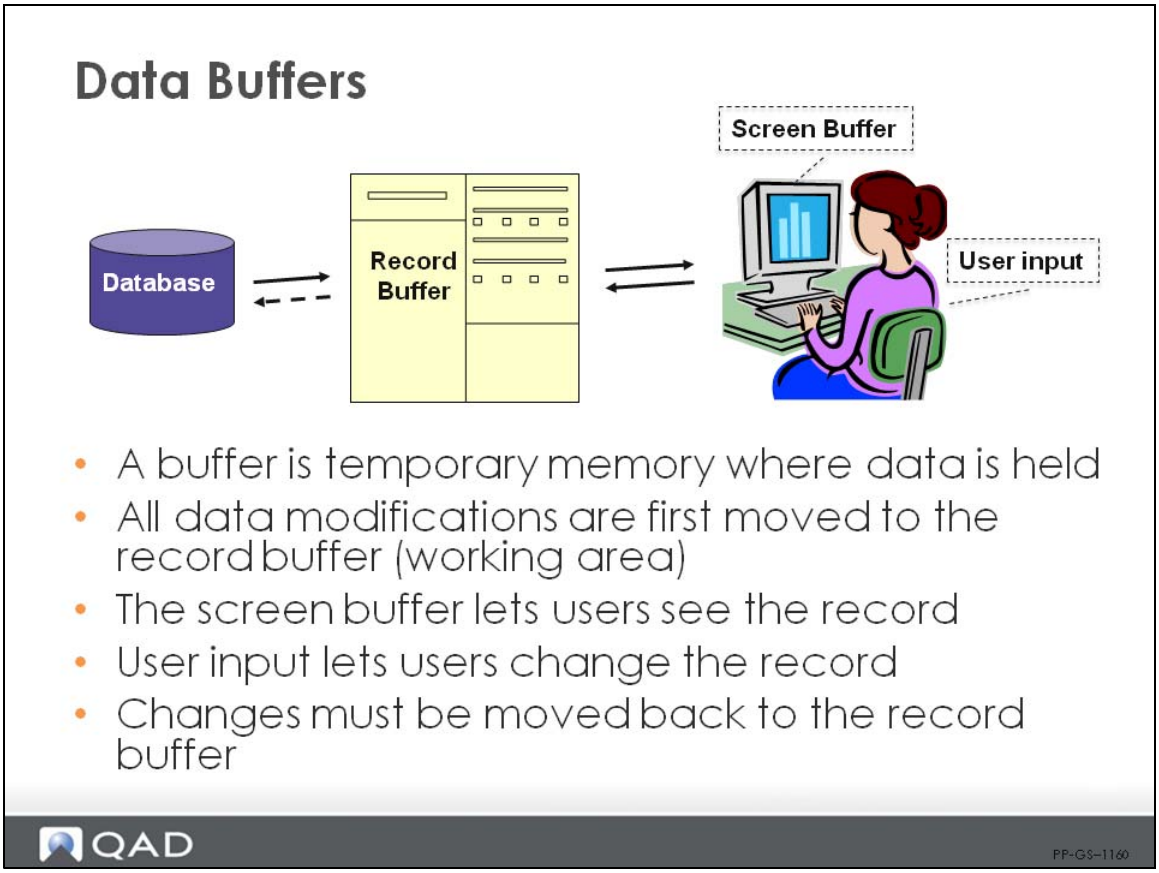


The CASE statement has better performance than IF...THEN...ELSE when testing more than two conditions.

CASE Statement Example

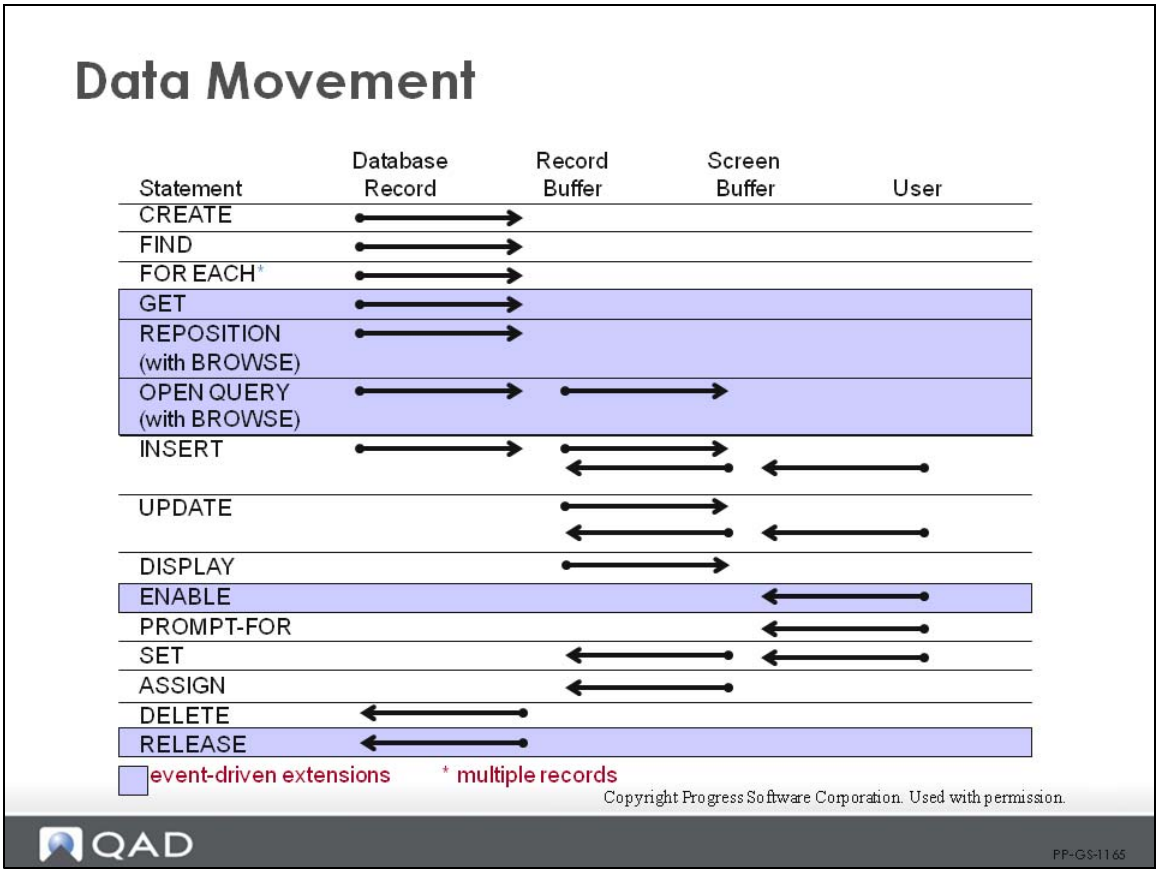


Data Buffers



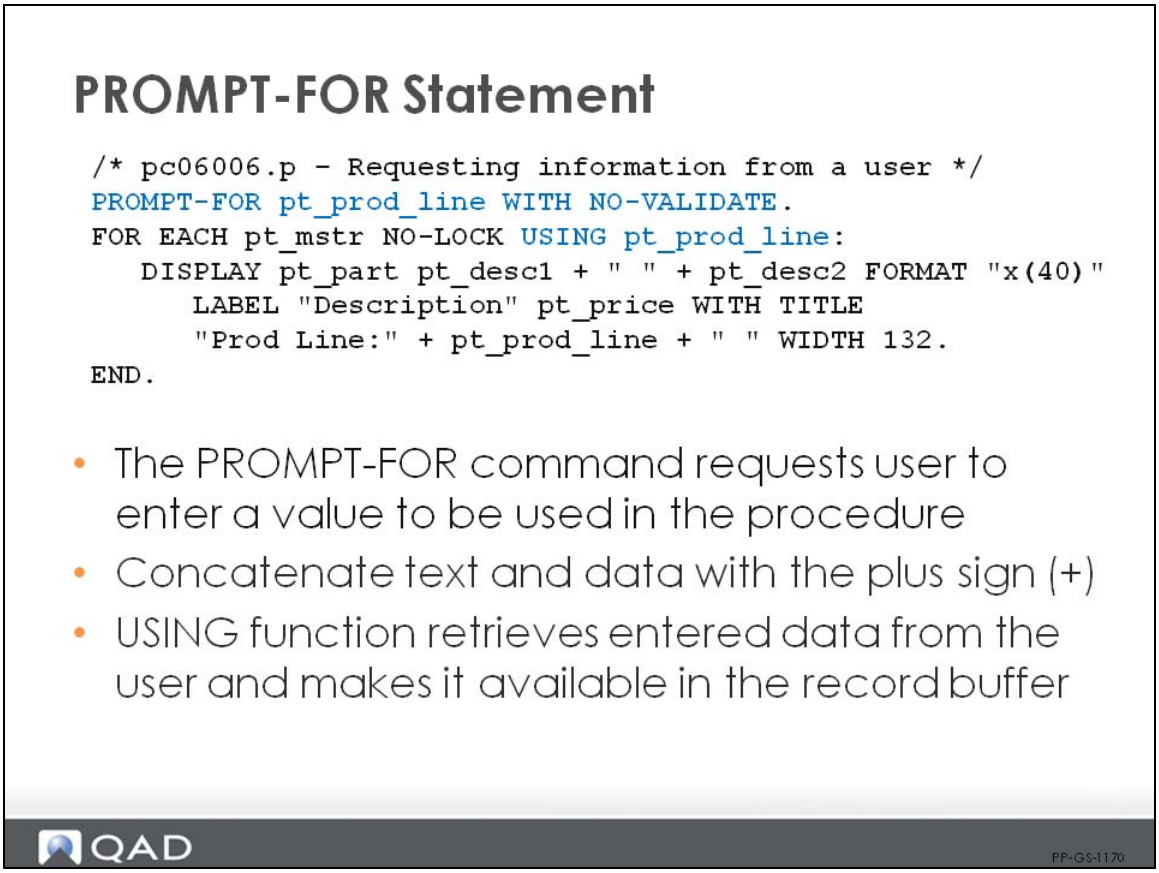
In Progress, the record buffer is not the same as the screen buffer. A buffer is a place in temporary memory where data is held.

Data Movement



This slide shows the detailed direction of data movement for various Progress statements. For example, the FIND statement moves the data from the database into the record buffer where it can be further manipulated.

PROMPT-FOR Statement



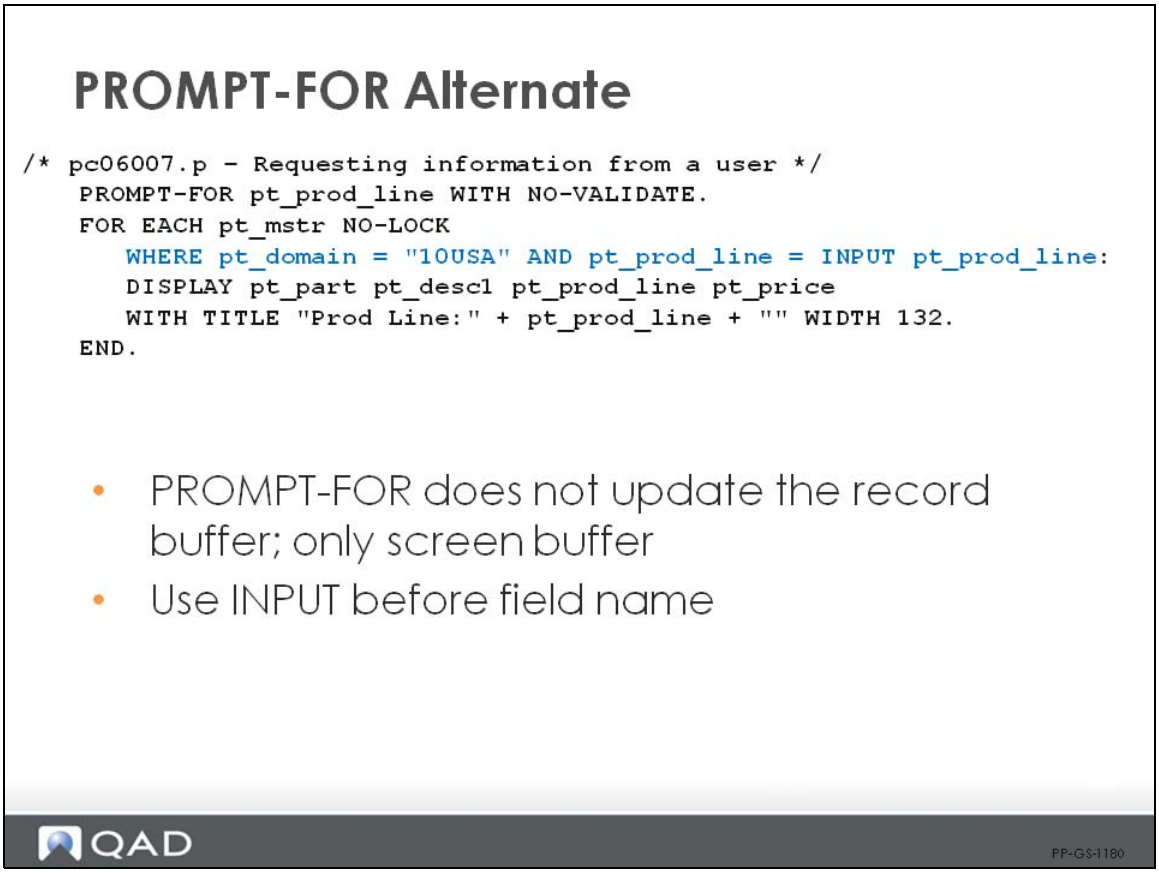
In general, you want users to be able to enter selection criteria when they generate a report or inquiry. The PROMPT-FOR command requests the user to enter a value to use in the procedure. NO-VALIDATE, when used within the frame phrase, turns off the validation of database fields.

In the above sample procedure, notice that the frame phrase option TITLE can display both text and data fields. The plus sign (+) is used to combine both text and data into one title or message line.

The USING statement in the record phrase specifies the name of an indexed field. Progress translates the USING option into the equivalent WHERE option.

Note The USING statement is not commonly utilized in the operational area of QAD EA because it is not very efficient. USING only accepts a single input criteria, such as pt_prod_line, so the data returned cannot be further qualified with the session's working domain. For tables with many records, this can create a significant performance issue.

PROMPT-FOR Statement Alternate



PROMPT-FOR does not update the record buffer, only the screen buffer. Remember, in Progress, these two buffers are not the same.

Field values entered after a PROMPT-FOR must be referenced as INPUT variables. That is, the field name is preceded by the word INPUT.

Exercise 6-6: PROMPT-FOR

1 Modify the procedure `pc06006.p`:

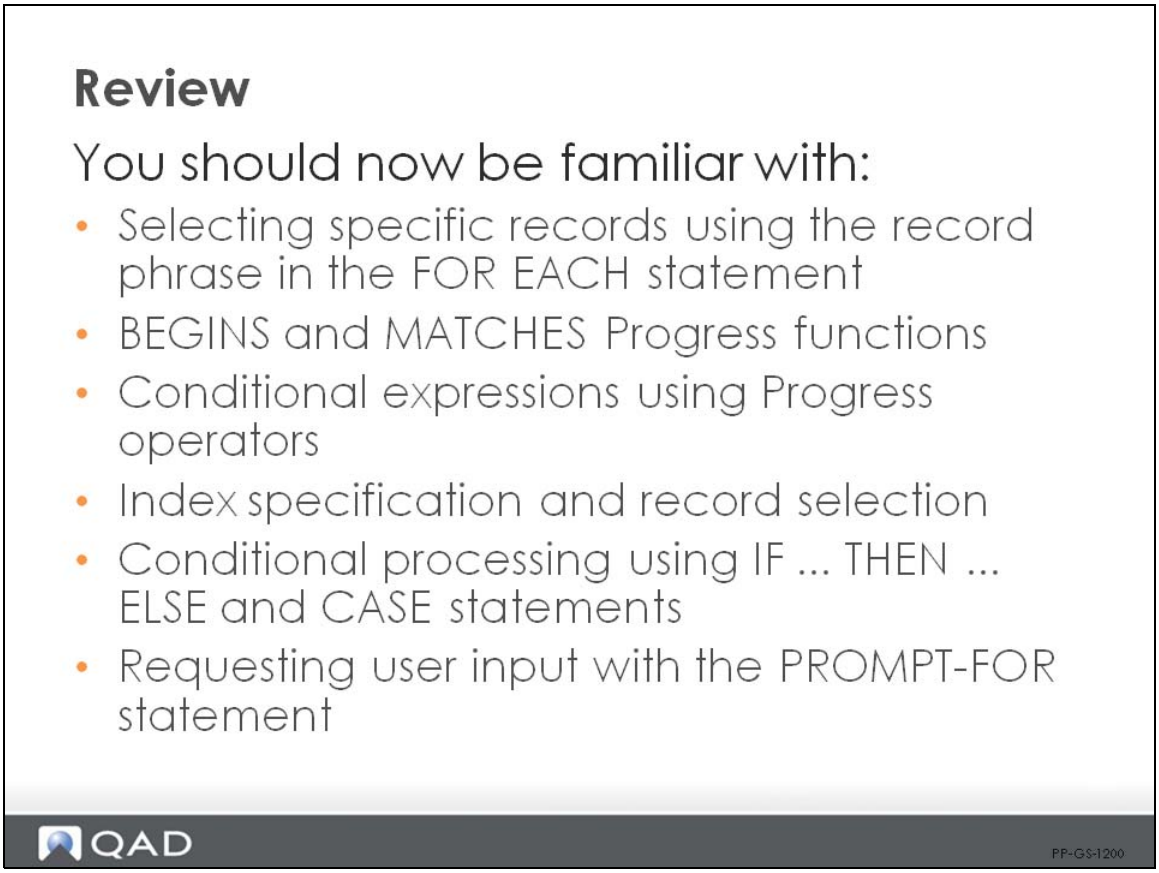
From: `USING pt_prod_line`
To: `WHERE pt_prod_line = INPUT pt_prod_line`

2 Modify this procedure to use item number input to create the same inquiry.

3 Write a procedure to input a location, then report all the items and quantities on hand in that location.

Hint. Use ld_det and remember to set ld_domain = "10USA" in the WHERE clause.

Review



In this lesson, you learned to modify the FOR EACH statement to retrieve selected records from a table. You should know how to request data from a user, how to use that data in a procedure, and how to use IF...THEN...ELSE conditional processing.

The WHERE record phrase option is used within the FOR EACH statement to specify a conditional expression for record selection.

Progress functions such as BEGINS and MATCHES are used with string data types to simplify the process of creating a conditional expression.

IF...THEN...ELSE conditional processing can be used in addition to the WHERE option and the CASE statement can be used for complex conditions.

Chapter 2

Writing More Complex Reports

Agenda for Section 2

This section of the class consists of five lessons:

- Lesson 7: Using FIND, REPEAT, and Defined Variables
- Lesson 8: Accessing Multiple Tables
- Lesson 9: Frame Control
- Lesson 10: Using Include Files
- Lesson 11: Arrays, Temp-Tables, Buffers, and Other Useful Functions

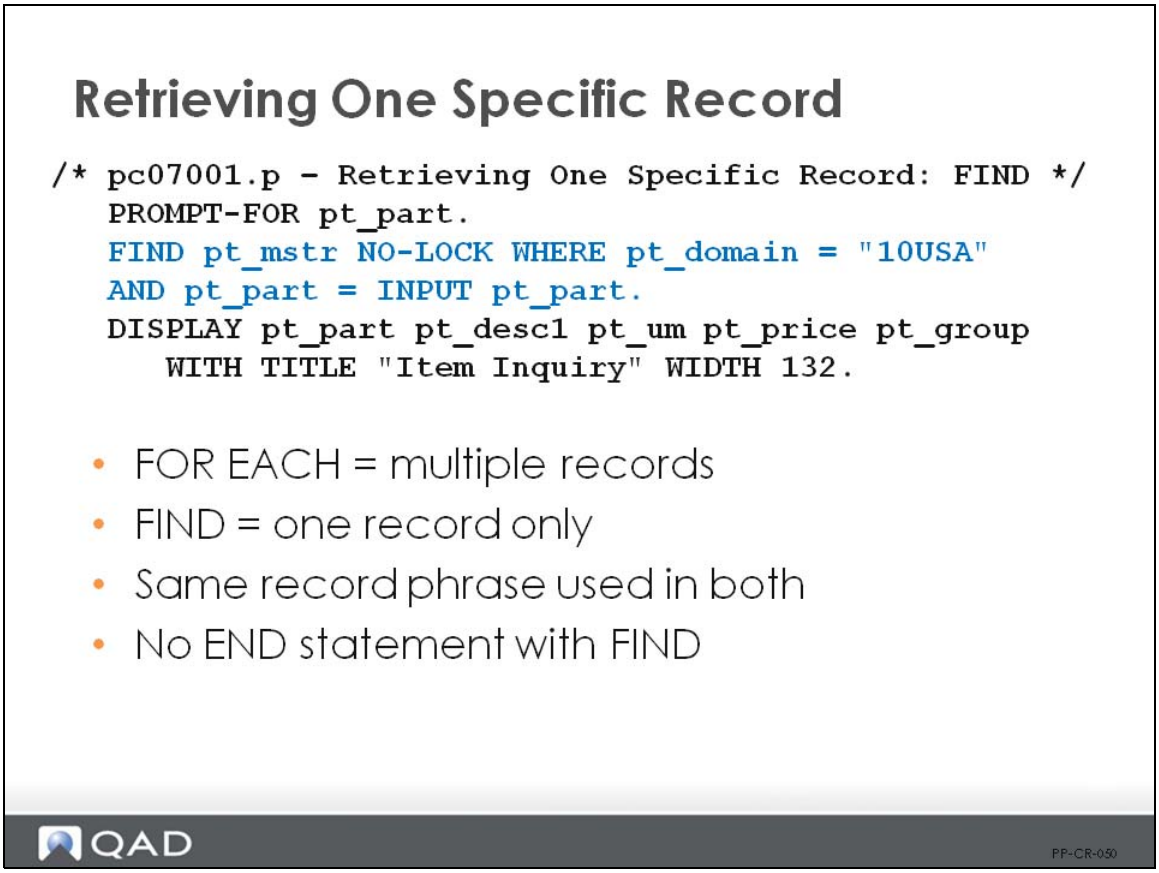
Each lesson includes goals, concepts, and hands-on exercises.

Lesson 7: Using FIND, REPEAT, and Defined Variables

Upon completion of this lesson, you will be familiar with the following topics:

- Using a FIND statement to retrieve a single record
- Using the NO-ERROR and AVAILABLE options of the FIND statement
- Using the REPEAT statement
- Using the DEFINE statement to create user-defined variables
- Using the UPDATE statement and how it differs from the PROMPT-FOR statement

Retrieving One Specific Record



As we have seen in previous examples, the FOR EACH statement reads multiple records from a table and retrieves all records that meet the specified criteria. This is not very efficient if you just want to look at one specific record. To retrieve a single, unique record, use the FIND statement.

The FIND statement ends with a period. No END statement is required, since the FIND statement does not constitute a block, and executes only once. The FIND statement uses the same record phrase as the FOR EACH statement. The FIND statement is typically used to access data in control tables (xxc_ctrl) and in procedures that access multiple tables.

Exercise 7-1: FIND

1 Modify procedure `pc07001.p`. Notice that you can position the NO-LOCK phrase anywhere.

From:
```
FIND pt_mstr NO-LOCK WHERE pt_domain = "10USA"
AND pt_part = INPUT pt_part.
```
To:
```
FIND pt_mstr WHERE pt_domain = "10USA"
AND pt_part = INPUT pt_part NO-LOCK.
```

From:
```
FIND pt_mstr WHERE pt_domain = "10USA"
AND pt_part = INPUT pt_part NO-LOCK.
```
To:
```
FIND pt_mstr WHERE pt_domain = "10USA"
AND pt_part > INPUT pt_part NO-LOCK.
```

Hint. Enter 01010 when prompted for an item.

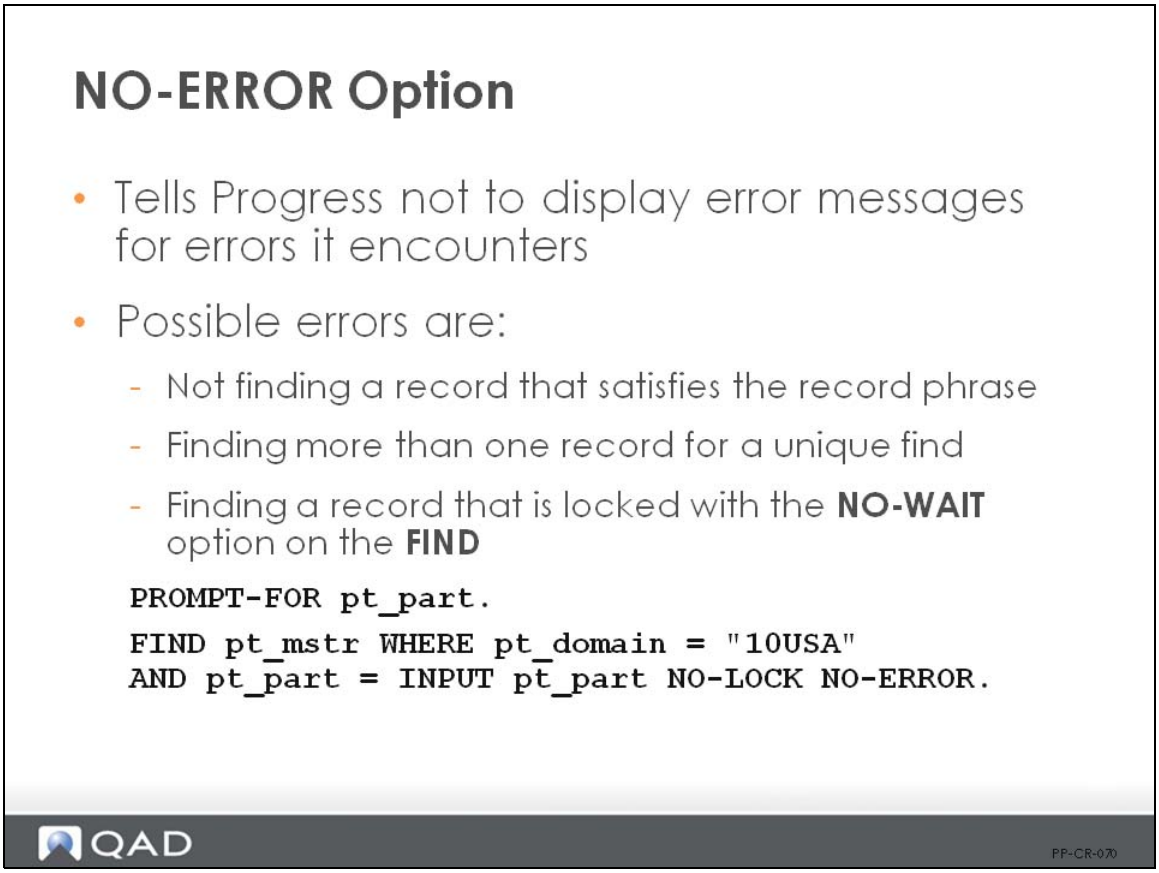
2 Modify `pc07001.p` to look for item 04-5025, which does not exist in the database. What happens?

3 Create a program that prompts for a customer number and then displays the customer name and address information.

Hint. Use ad_mstr and remember to use ad_domain = "10USA" in the WHERE clause.

Note The field labeled Address (ad_addr) contains the customer and supplier number, not their physical location.

NO-ERROR Option



The Progress error message you saw in the last exercise is not very user friendly. You may prefer to create your own message and allow further processing in the event a record is not found.

NO-ERROR inhibits the Progress error messages that are displayed when the specified record does not exist. You must provide the error handling in your procedure when using the NO-ERROR option.

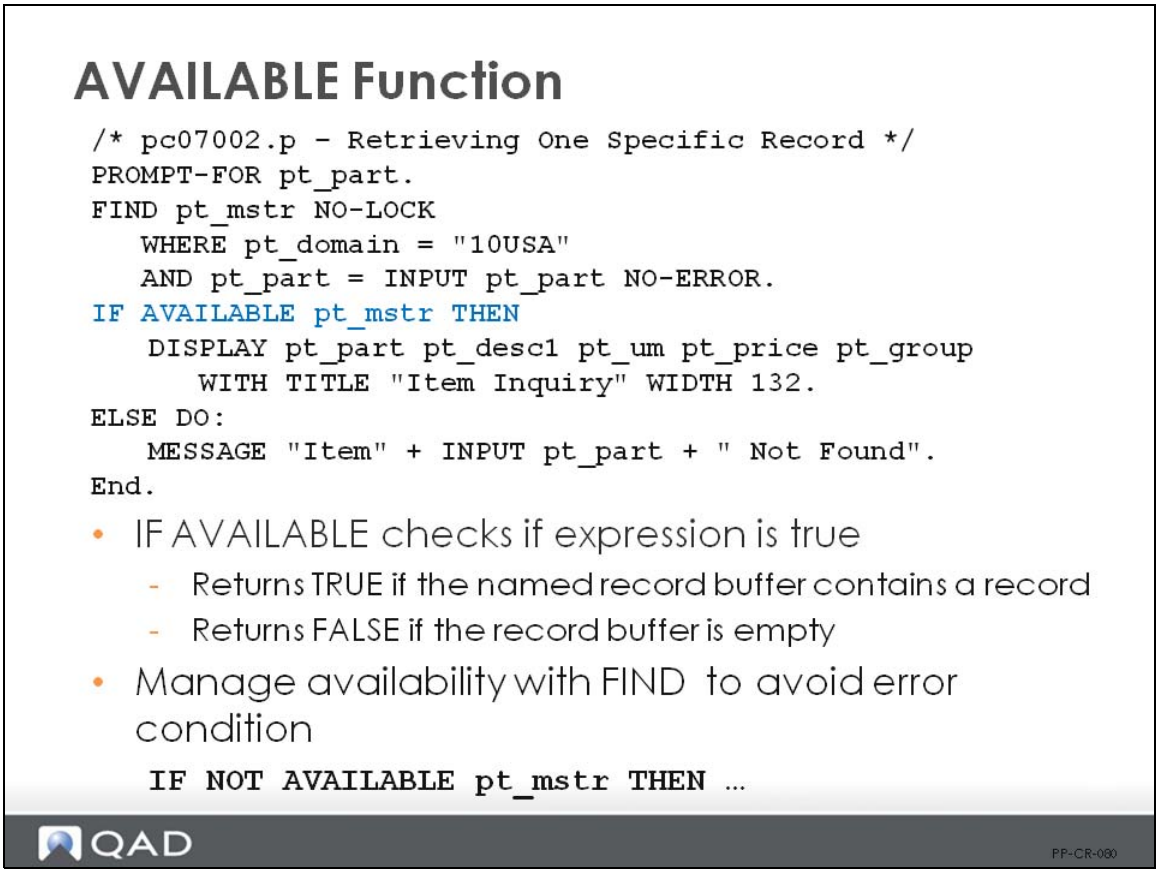
The NO-WAIT option on the FIND statement causes the FIND to return immediately without waiting when the record is locked by another user.

To suppress Progress error messages and to prevent termination of the procedure, you must put the NO-ERROR clause on the FIND statement.

Possible errors are:

- Not finding a record that satisfies the record phrase
- Finding more than one record for a unique find
- Finding a record that is locked with the NO-WAIT option on the FIND

AVAILABLE Function



The IF AVAILABLE expression is TRUE when one, and only one, record is found that meets the test condition in the FIND statement. If more than one record is found, the IF AVAILABLE expression is FALSE.

The conditional expression IF AVAILABLE uses the AVAILABLE function. The IF AVAILABLE expression would be TRUE if the requested record was retrieved. The IF NOT AVAILABLE expression would be TRUE if the requested record was not found.

The IF statement lets you test a condition and then do something if the condition is TRUE or FALSE. In the sample procedure `pc07002.p`, if the record is not found, an error message is displayed; otherwise, the record is displayed.

MESSAGE displays the message text in the message area. The last three lines of the screen are reserved: two lines for MESSAGES and the last line for a system STATUS message.

Exercise 7-2: AVAILABLE

1 Modify procedure `pc07002.p`.

```
From:           MESSAGE "Item" + INPUT pt_part + " Not Found".
To:             MESSAGE "Item" + pt_part + " Not Found".
```

2 Run the modified program with a valid item (01010).

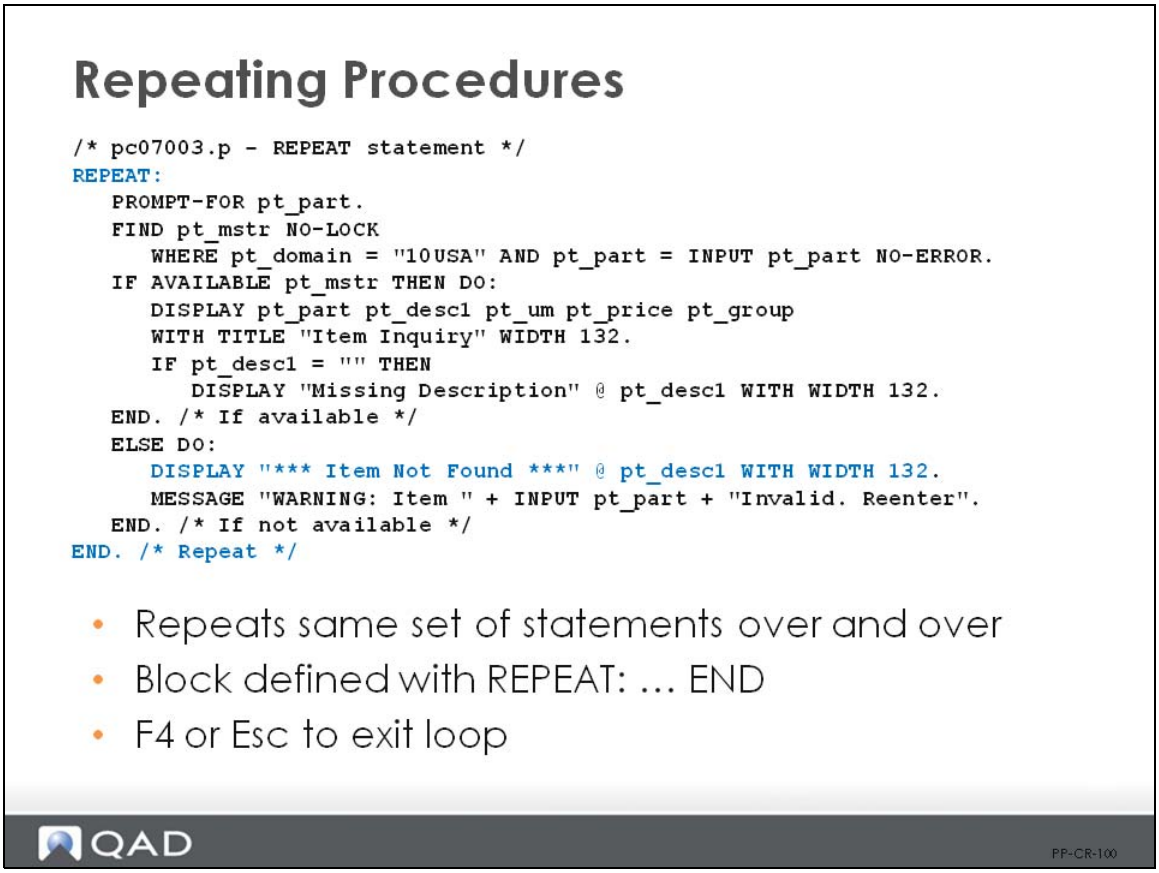
3 Run the program with an invalid item (your initials).

4 Why did removing INPUT from pt_part cause a Progress error instead of the message?

5 Modify the program as follows and rerun first with a valid and then an invalid item.

```
From:           ELSE DO:
To:             IF NOT AVAILABLE pt_mstr THEN DO:
```

Repeating Procedures



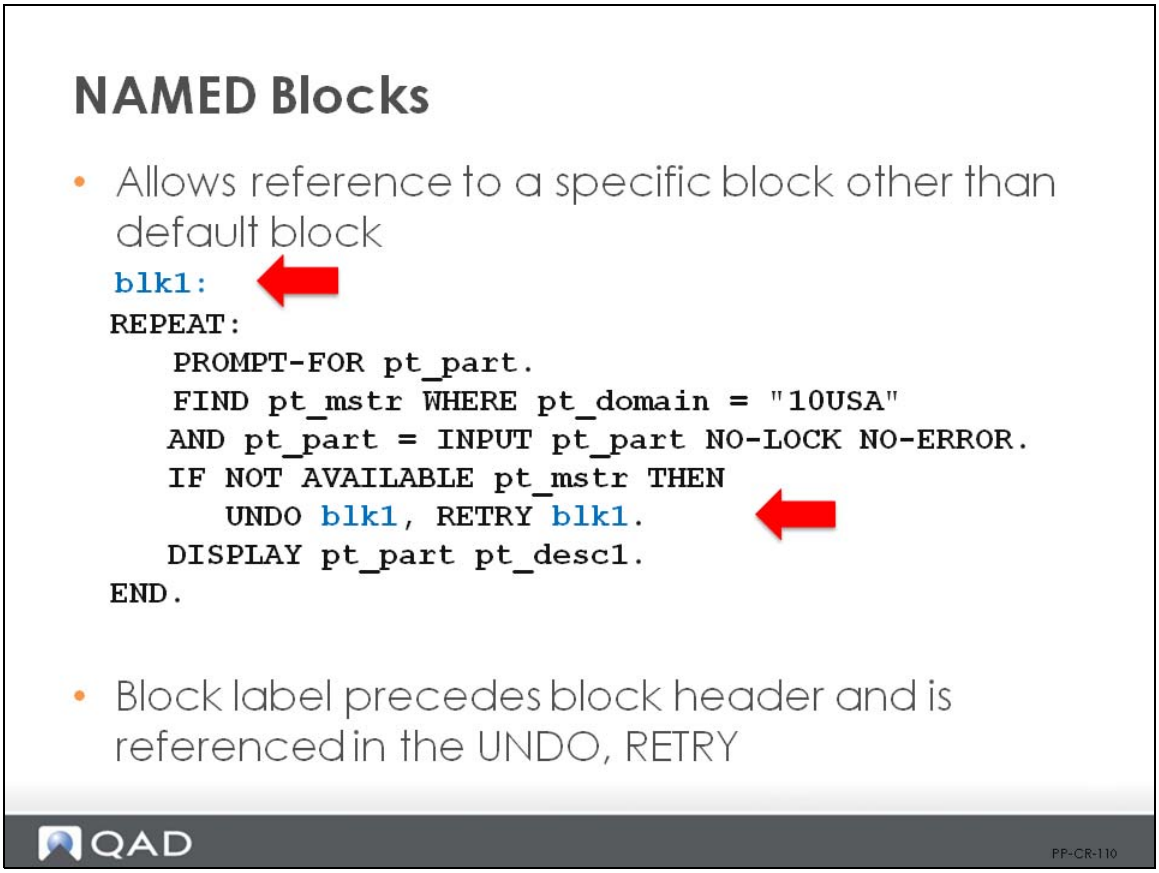
Use a REPEAT block to execute the same set of statements over and over. The REPEAT statement must end with a colon.

The block of commands to execute within the REPEAT loop must be followed by an END statement. Pressing the F4 or Esc key tells Progress to end and exit from the REPEAT block.

The frame phrase can be applied to the REPEAT statement, as well as the DISPLAY and FOR EACH statements.

It is good programming practice to put at least one DISPLAY statement inside a REPEAT loop. This avoids endless looping within a program.

NAMED Blocks



The name of the block blk1: does not require an END statement to complete the block. The real block starts at the REPEAT: statement and ends at the END. The blk1: statement is just a name for the block.

Exercise 7-3: REPEAT

1 Modify procedure `pc07003.p`, and add a new UNDO, RETRY statement on the line below MESSAGE. Note any differences in the function of the procedure.

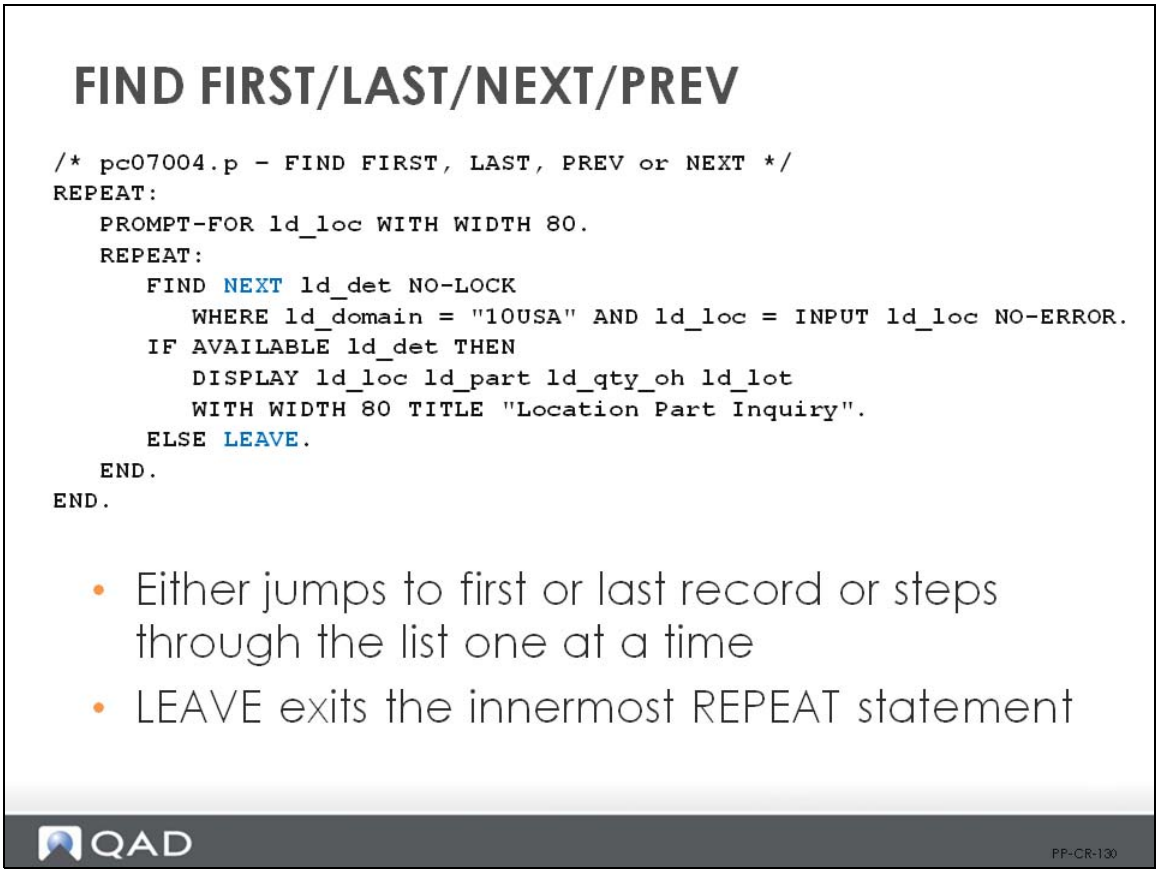
2 Review the function principle used in:

```
DISPLAY "(Missing Description)" @ pt_desc1.
DISPLAY "*** Item Not Found ***" @ pt_desc1.
```

3 Modify `pc07003.p` again:

From: `UNDO, RETRY.`
To: `UNDO, LEAVE.`

FIND FIRST/LAST/NEXT/PREV



This sample procedure could have been written using FOR EACH rather than FIND, but using FIND FIRST/NEXT rather than FOR EACH gives you additional capability. You can do something solely for the first record occurrence and not the others.

Use LAST/PREVIOUS, then the REPEAT statement, to continue accessing and displaying the next record while a certain condition is true. This is not possible with the FOR EACH command.

Exercise 7-4: FIND FIRST/LAST/NEXT/PREV

Make the following changes to procedure `pc07004.p`. Run the program after each change with location 1000 and note the differences in execution:

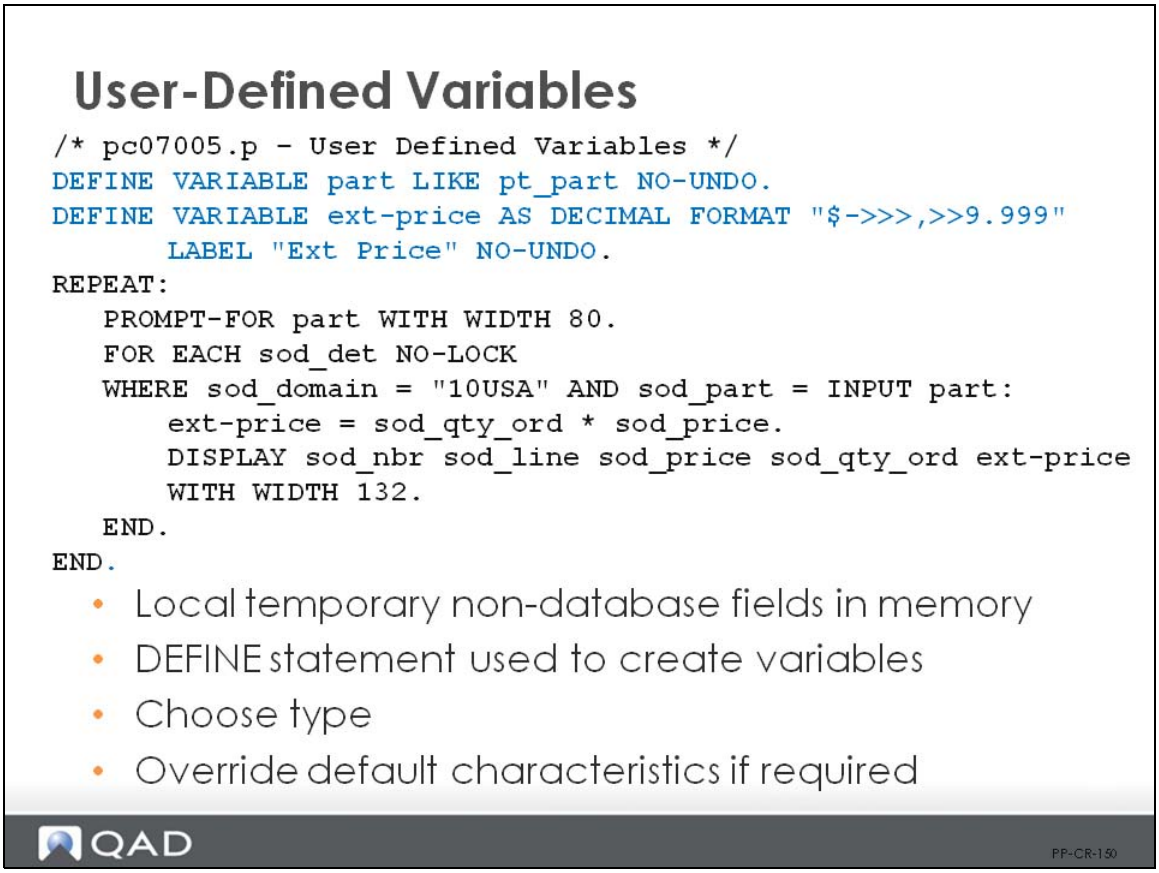
```
From:  FIND NEXT ld_det NO-LOCK WHERE ld_domain = "10USA"
       AND ld_loc = INPUT ld_loc NO-ERROR.
To:    FIND FIRST ld_det NO-LOCK WHERE ld_domain = "10USA"
       AND ld_loc = INPUT ld_loc NO-ERROR.

From:  FIND FIRST ld_det NO-LOCK WHERE ld_domain = "10USA"
       AND ld_loc = INPUT ld_loc NO-ERROR.
To:    FIND LAST  ld_det NO-LOCK WHERE ld_domain = "10USA"
       AND ld_loc = INPUT ld_loc NO-ERROR.
```

From:
```
FIND LAST ld_det NO-LOCK WHERE ld_domain = "10USA"
AND ld_loc = INPUT ld_loc NO-ERROR.
```
To:
```
FIND PREV  ld_det NO-LOCK WHERE ld_domain = "10USA"
AND ld_loc = INPUT ld_loc NO-ERROR.
```

From:
```
FIND PREV ld_det NO-LOCK WHERE ld_domain = "10USA"
AND ld_loc = INPUT ld_loc NO-ERROR.
```
To:
```
FIND ld_det NO-LOCK WHERE ld_domain = "10USA"
AND ld_loc = INPUT ld_loc NO-ERROR.
```

User-Defined Variables



Sometimes defining a variable to store the result of a complex expression or calculated field is more efficient than rebuilding the expression each time you want to use it. This is especially true if you plan to use the calculations more than once.

User-defined variables, sometimes called local variables, are used to create temporary non-database field for use within a procedure.

The DEFINE statement is used to create user-defined variables. Its two most basic forms are:

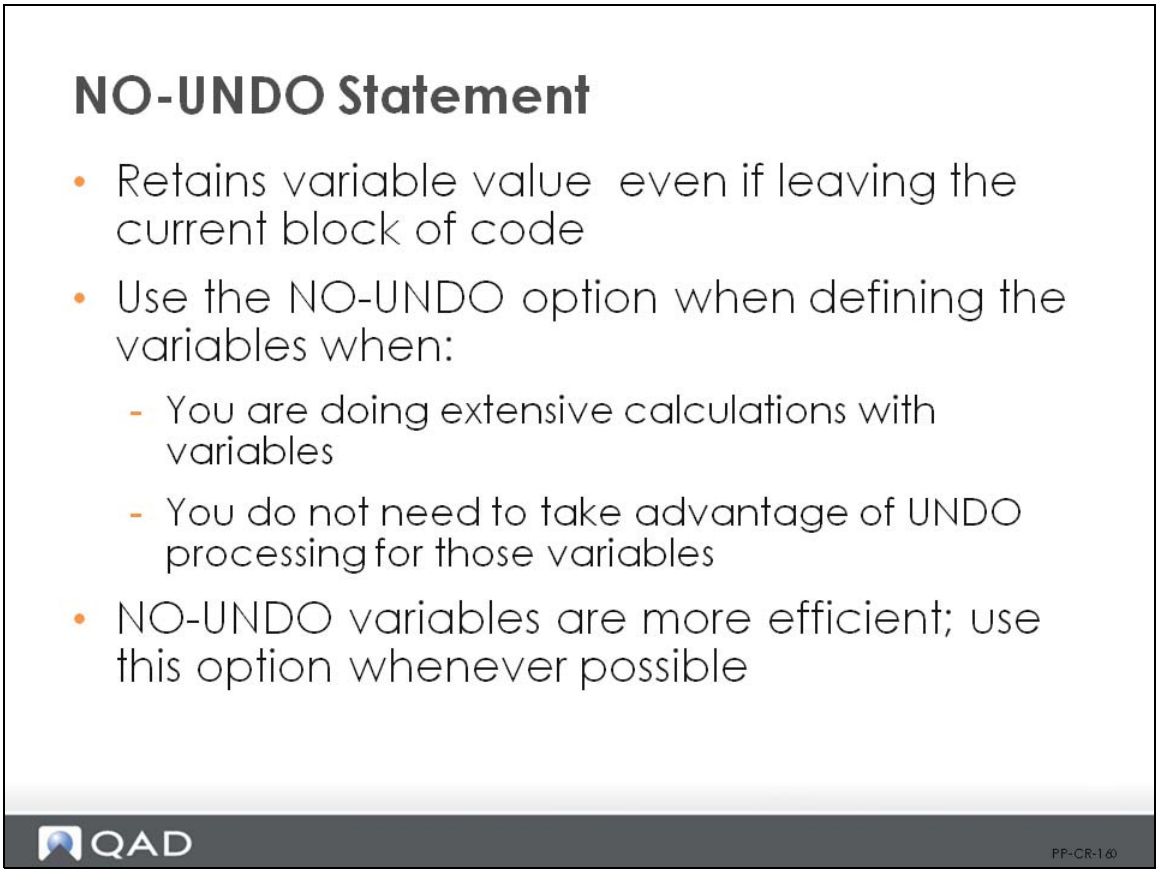
```
DEFINE VARIABLE variable LIKE field
DEFINE VARIABLE variable AS data-type
```

When you define a variable LIKE a database field, it takes on the characteristics of that field. This includes the data type, format, number of decimals, extents, column label, and label. An extent is the number of elements in an array field or variable.

An alternate form of the DEFINE statement is the AS data type. You are not cloning the characteristics of any other field with this. You must supply all of the formatting and label information yourself.

The default characteristics of a variable can be modified using options such as LABEL, COLUMN-LABEL, and FORMAT. Valid data types are: CHARACTER, DATE, DECIMAL, INTEGER, LOGICAL, and HANDLE. See “Data Types” on page 23 for details.

NO-UNDO Statement

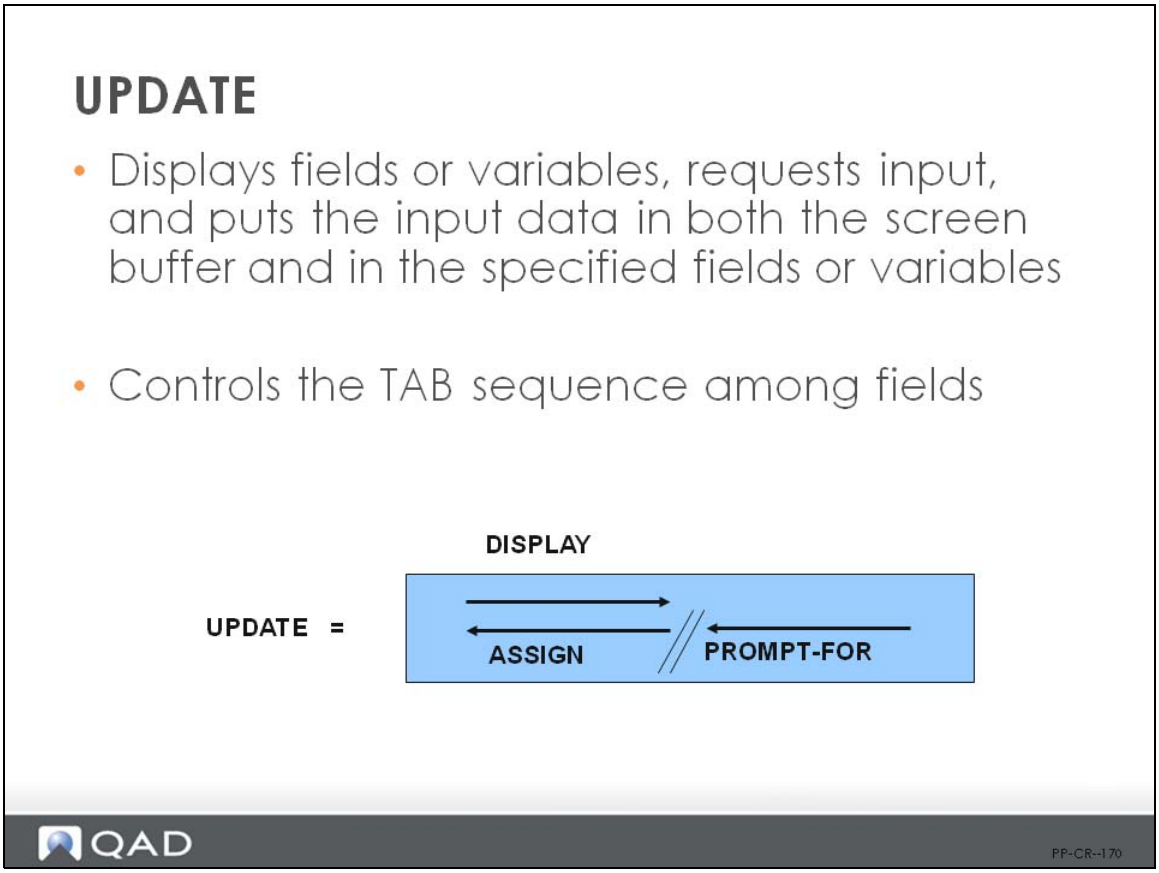


Use a NO-UNDO statement when a value does not need to be undone when a particular block is terminated. Without NO-UNDO, Progress retains a variable value beyond transaction scope by writing the original value to the `.bi` file.

Note Progress uses the before and after image files to support data integrity and backing out of changes when necessary. See “Managing a Transaction” on page 226 for details.

If you are doing extensive calculations with variables and you do not need to take advantage of undo processing for those variables, use the NO-UNDO option when defining the variables.

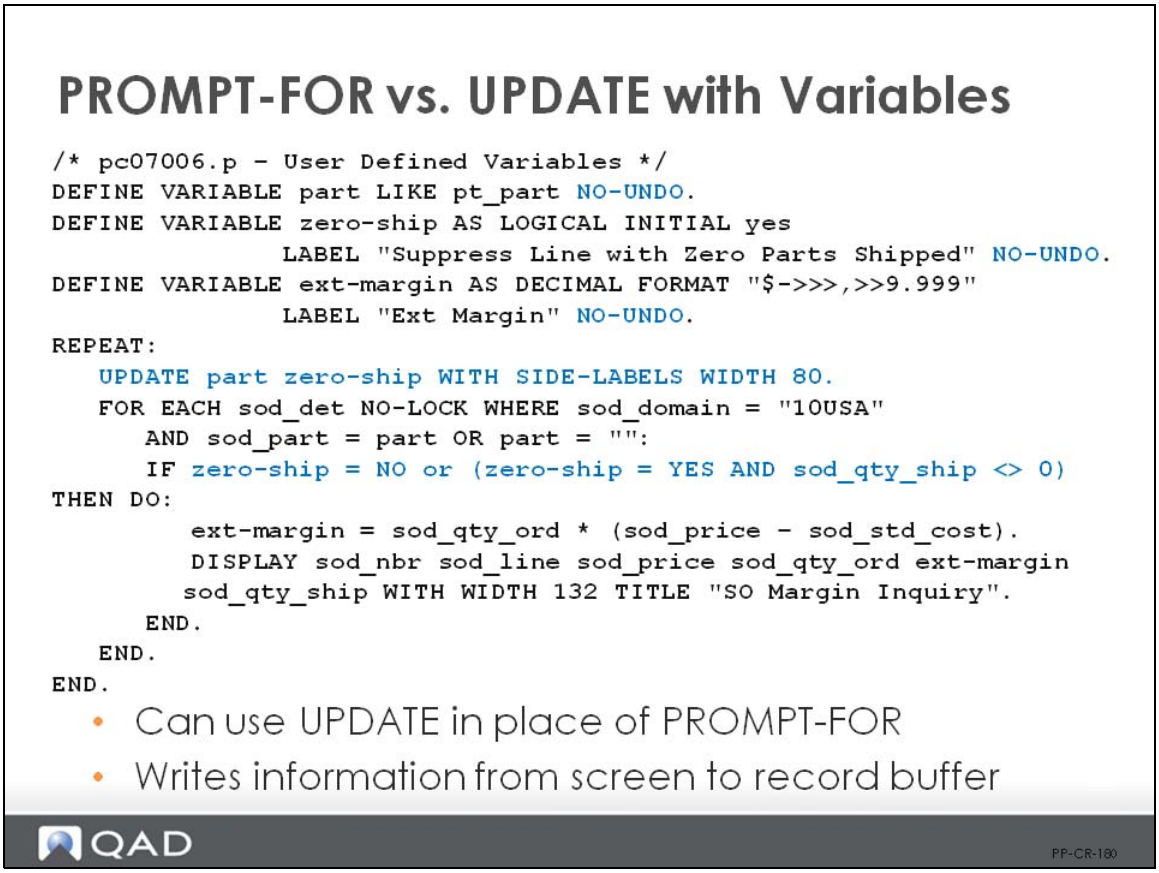
UPDATE



The UPDATE statement is a combination of the following statements:

- DISPLAY: Moves the values of fields or variables into the screen buffer and displays them
- PROMPT-FOR: Prompts the user for data and puts that data into the screen buffer
- ASSIGN: Moves data from the screen buffer to the record buffer

PROMPT-FOR vs. UPDATE with Variables



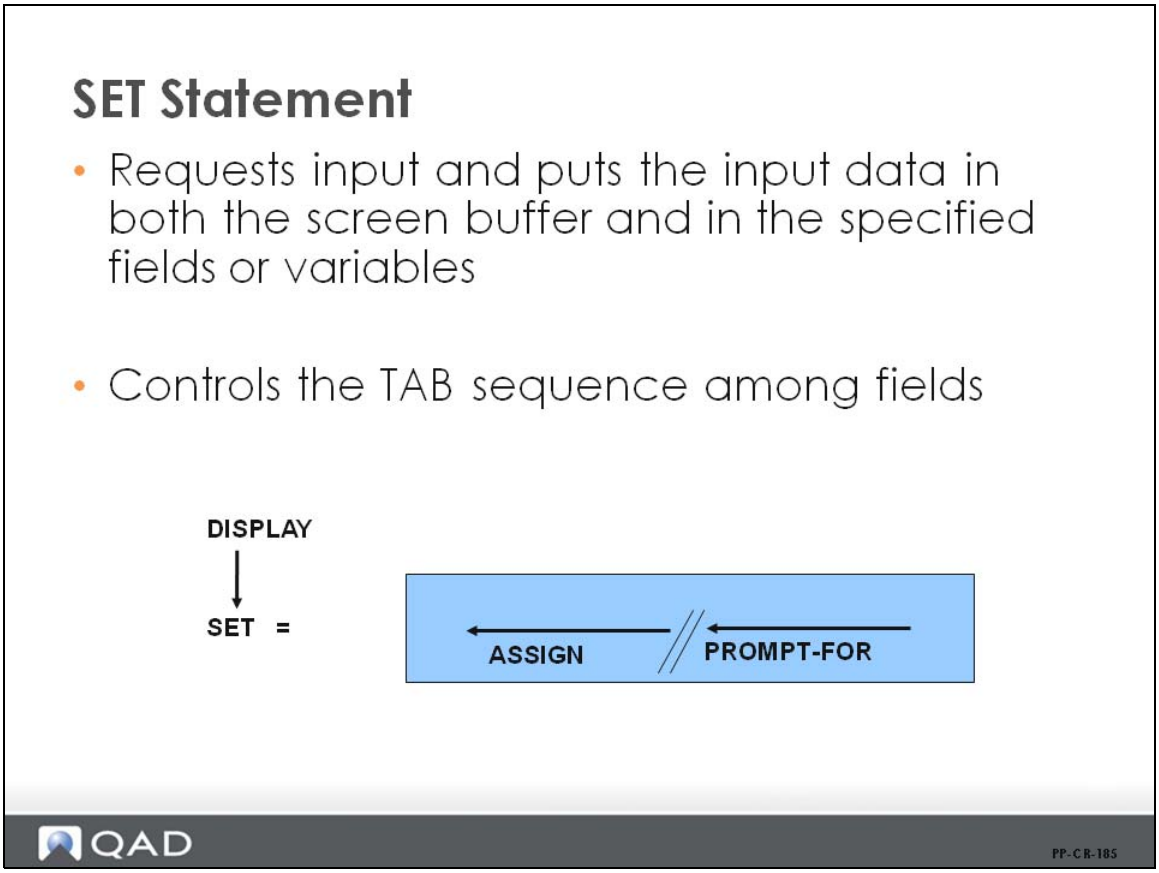
UPDATE can be used in place of PROMPT-FOR. The UPDATE statement differs from the PROMPT-FOR statement in that it writes to the record buffer and can update the database.

When writing report and inquiry procedures, you should only use the UPDATE statement with local variables. Since UPDATE writes to the record buffer, it is not necessary to refer to the user input data as an INPUT variable.

PROMPT-FOR writes to the screen buffer, but does not update the record buffer. This prevents the database from being updated by a value entered as a selection criteria when using the PROMPT-FOR statement.

You can set INITIAL values for the variable in the DEFINE statement. But you may need to reset them again inside reiterating blocks (REPEAT, FOR EACH) that use the variable.

SET Statement

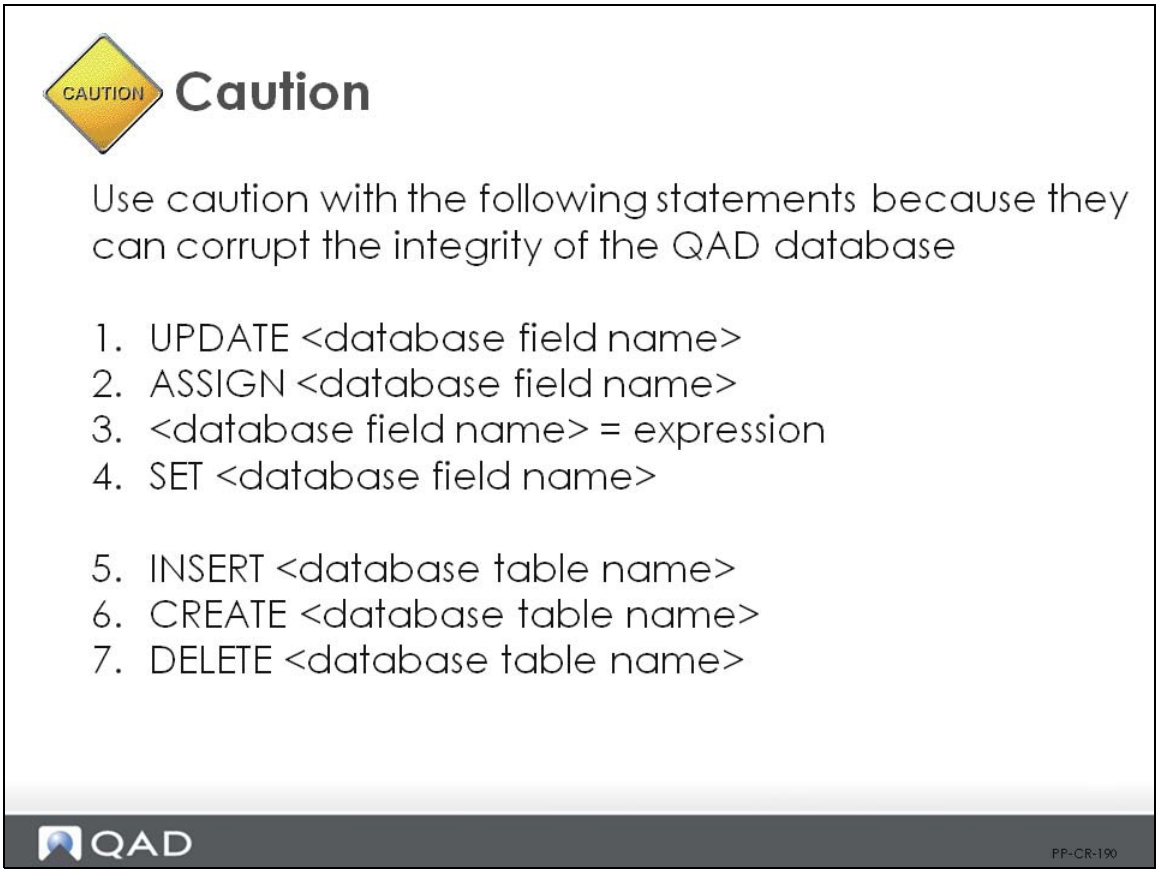


A DISPLAY statement generally precedes the SET statement.

The SET statement is a combination of the following statements:

- PROMPT-FOR: Prompts the user for data and puts that data into the screen buffer
- ASSIGN: Moves data from the screen buffer to the record buffer

Caution



Exercise 7-5: User-Defined Variables

1 Note any advantages or differences between using this:

```
"UPDATE part zero-ship"
```

Or this:

```
"PROMPT-FOR part zero-ship".
```

2 (Optional) Modify the program `pc07006.p` to round the value of ext-margin using the ROUND function.

Hint. ROUND (expression, precision)

expression = ext-margin

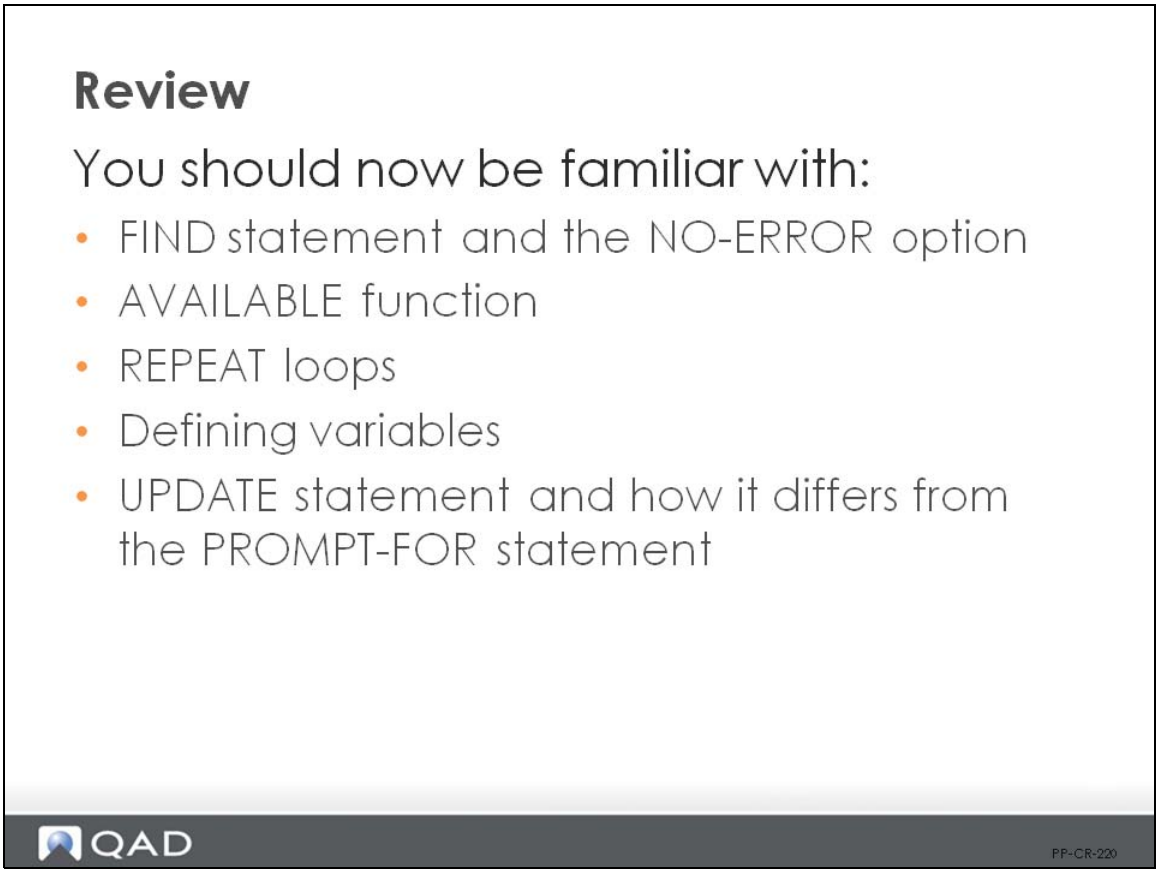
precision = the number of decimal places for rounding

3 Write a procedure that uses user-defined variables and the UPDATE statement. Ask a user for a location. Then list all of the items in that location.

Hints. Use the location detail table (ld_det) to find the items in a location. Display the lot number, item number, and quantity on hand.

Remember to use ld_domain = "10USA" in the WHERE clause. When prompted for location, specify 1000.

Review



In this lesson, you learned how to write repeating procedures, how to retrieve specific records using the FIND statement, and how to define your own variables (local variables). You were also introduced to the UPDATE statement.

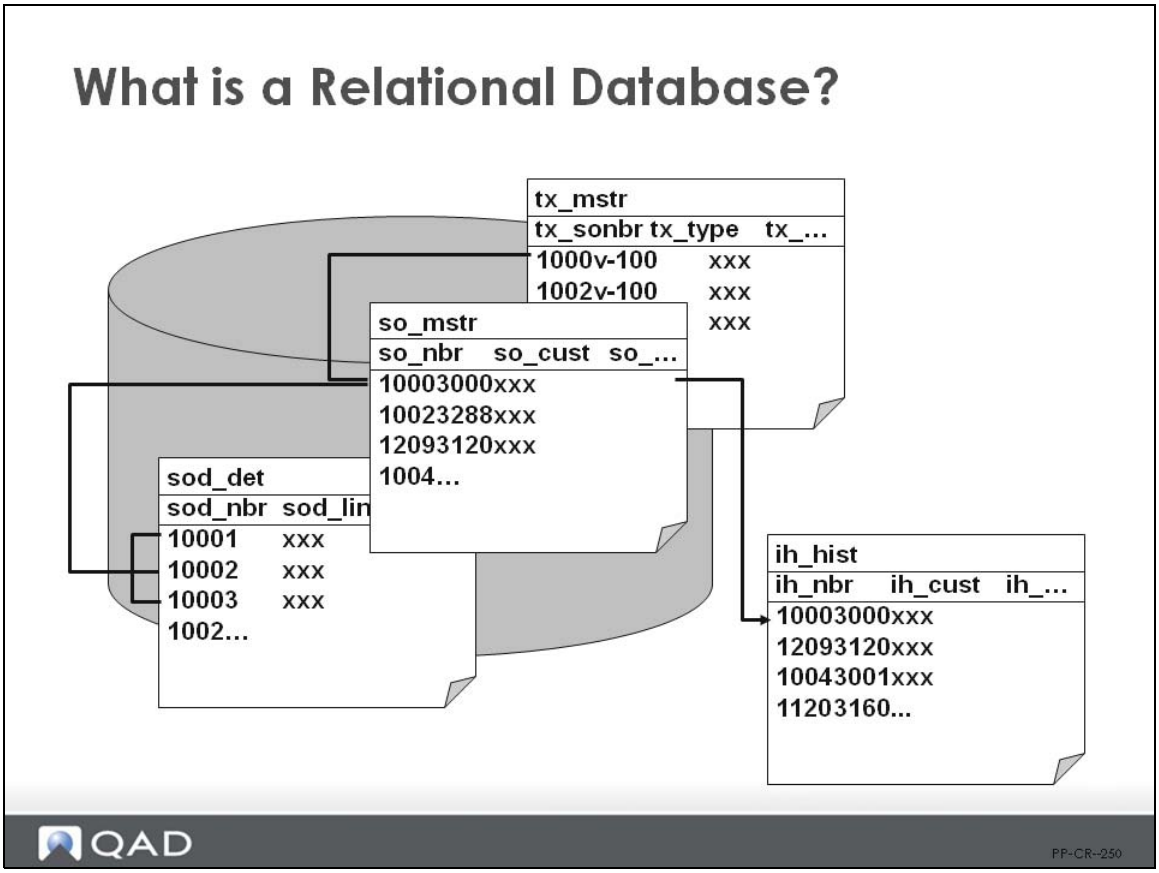
- You should use FIND to retrieve a single record, whereas you use FOR EACH to retrieve multiple records.
- REPEAT can be used to execute a set of statements over and over until F4 is pressed.
- User-defined variables are used to create temporary non-database fields to use with a procedure.
- The UPDATE statement is used for user-defined variables. Use this statement and the others below with caution since they modify the database.
  - UPDATE <database field name>
  - INSERT <database table name>
  - SET <database field name>
  - <database field name> = expression
  - CREATE <database table name>
  - ASSIGN <database field name>
  - DELETE <database table name>

Lesson 8: Accessing Multiple Tables

Upon completion of this lesson, you will be familiar with the following topics:

- Database table relationships
- How to determine table relationships using the QAD EA *Entity Diagrams* manual
- Accessing two or more tables using combinations of FIND and FOR EACH statements

What is a Relational Database?



In a relational database, each table is a self-contained collection of data about a single thing (entity). Often, all of the data for a report cannot be retrieved from a single table. It may be necessary to access two or more tables to obtain the required information.

Different database tables are related to each other by fields that contain common information. The Item Master table (pt_mstr) is related to the Inventory Master Detail table (in_mstr) by their respective item number fields: pt_part and in_part.

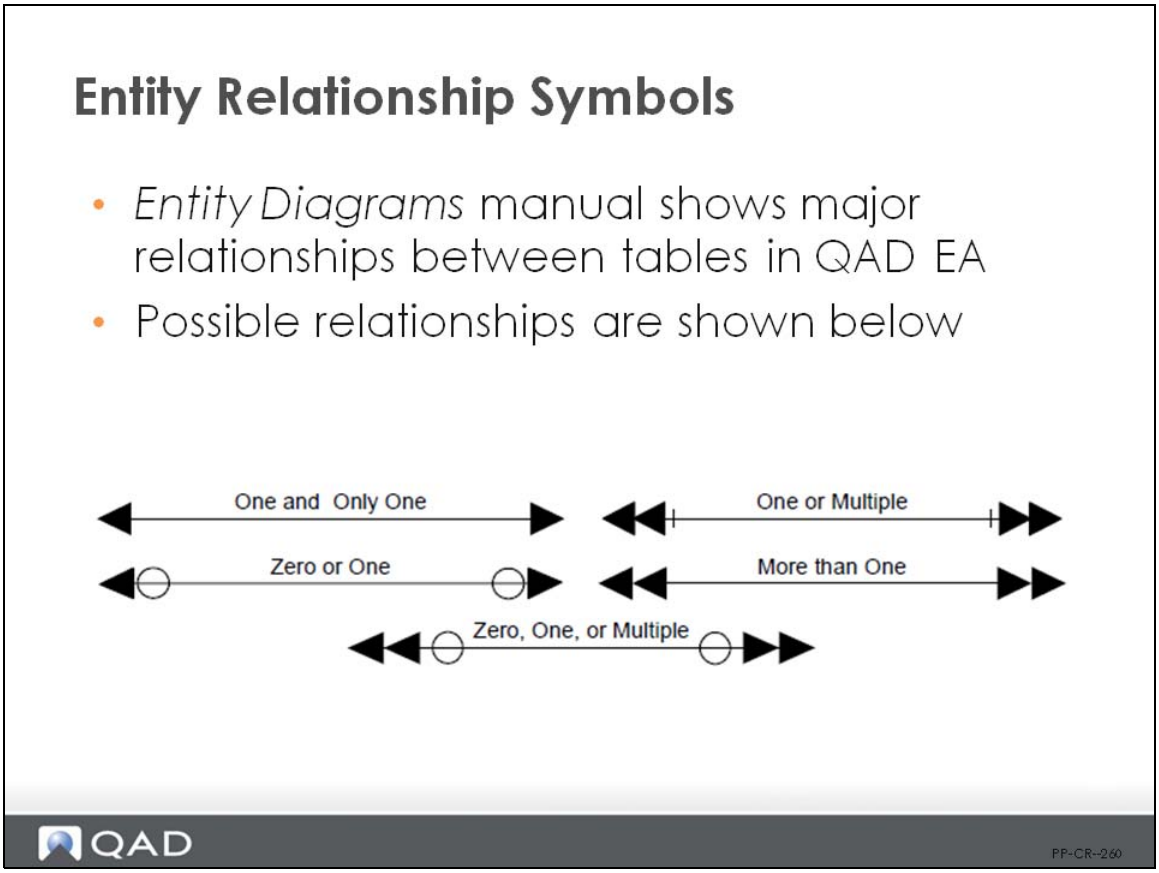
Important Fields between two tables can be related even though their field names are not similar. For instance, the field pt_part is related to two Product Structure Master (ps_mstr) fields: component item (ps_comp) and parent item (ps_par). The field cm_addr in the Customer Master table holds the same customer number as the field so_cust in the Sales Order Master table.

A table can also be related to another using an intermediate table. The Sales Order Master table (so_mstr) accesses the Item Master table (pt_mstr) through the Sales Order Detail table (sod_det). This relationship may be necessary to obtain Item Master information for a sales order.

One table can be related to many other tables. For example, address information is centrally stored in the Address Master (ad_mstr). Address Code (ad_mstr) is linked to the primary index fields of Customer Master (cm_mstr), Supplier Master (vd_mstr), and Salesperson Master (sp_mstr) tables.

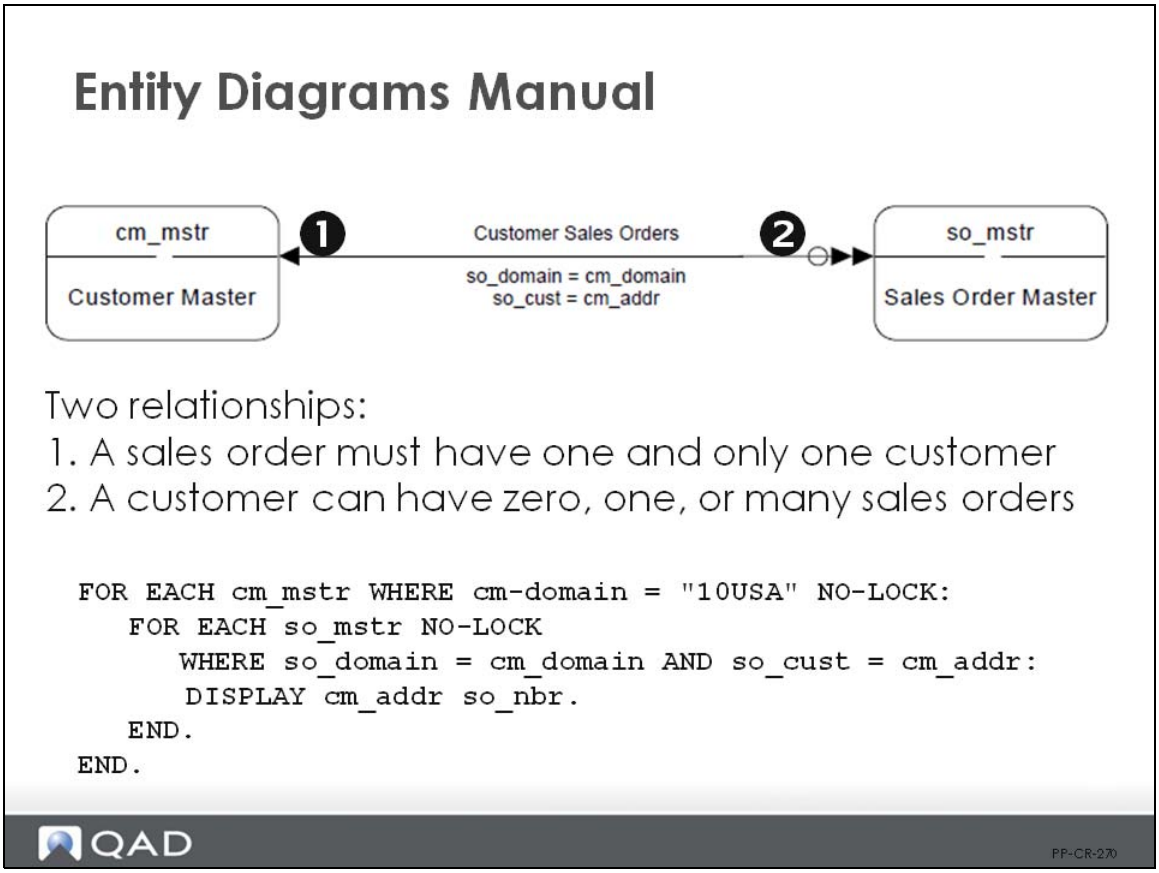
Note Address, customer and supplier data is managed differently in QAD Enterprise Edition than in Standard Edition. In EE, these tables contain only selected pieces of data copied from the primary Enterprise Financial tables for use in the operational modules.

Entity Relationship Symbols



There are five relationship line identifiers that you can combine to distinguish 15 specific entity relationships.

Entity Diagrams Manual



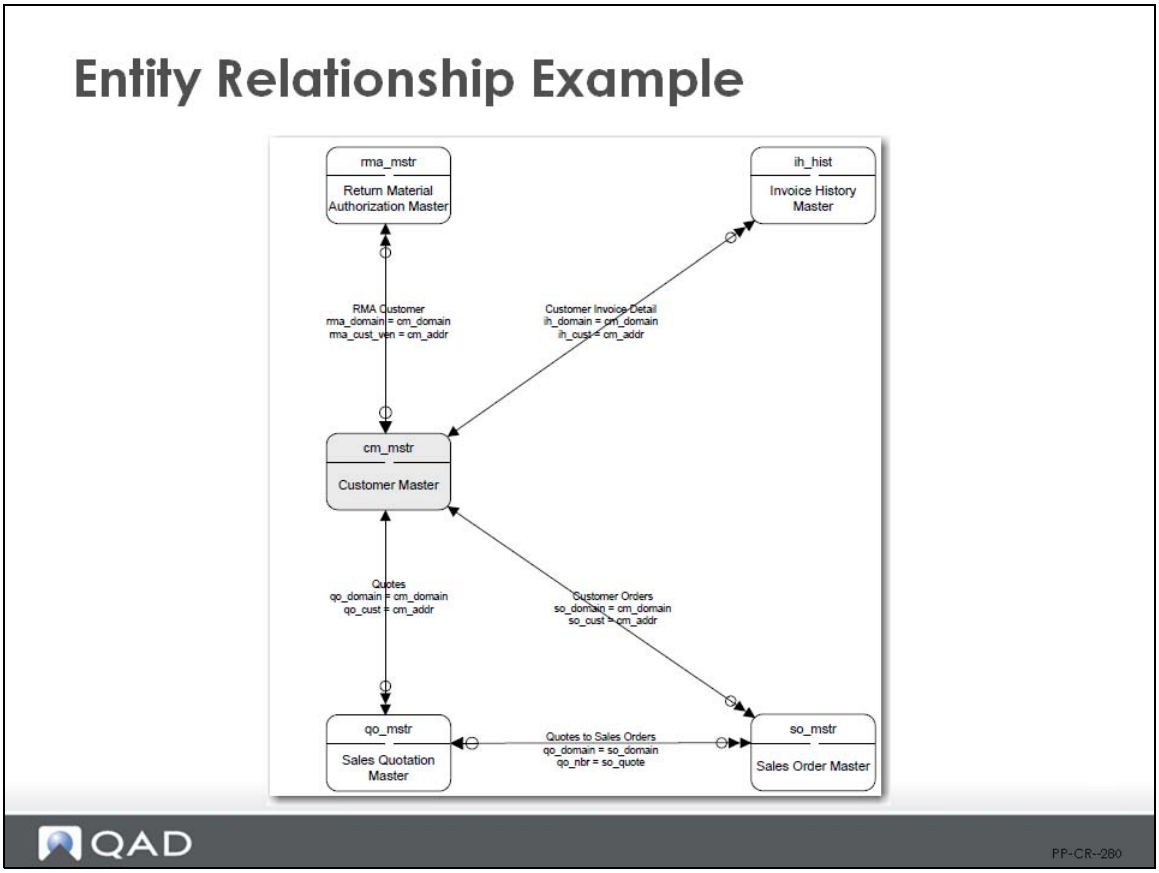
The diagram in this figure describes two relationships:

1 A customer can have zero, one, or many sales orders.

2 A sales order must have one, and only one, customer.

Below the relationship name (Customer Sales Orders), are the fields used for joining the entities. This diagram shows that to join these two tables within Progress, you would set so_cust equal to cm_addr or vice versa and so_domain equal to cm_domain.

Entity Relationship Example



These diagrams are designed to help:

- Programmers writing queries or custom reports.

  The diagrams show which tables can be joined and the field specifications required to make them join. Note that not all possible table relationships are shown, just the major ones.
- Administrators interested in the availability of custom reports.

  The diagrams show which reports you can prepare.
- Programmers interested in customizing or modifying the standard QAD system.

  The diagrams show relationships that must be preserved in any customization or modification.

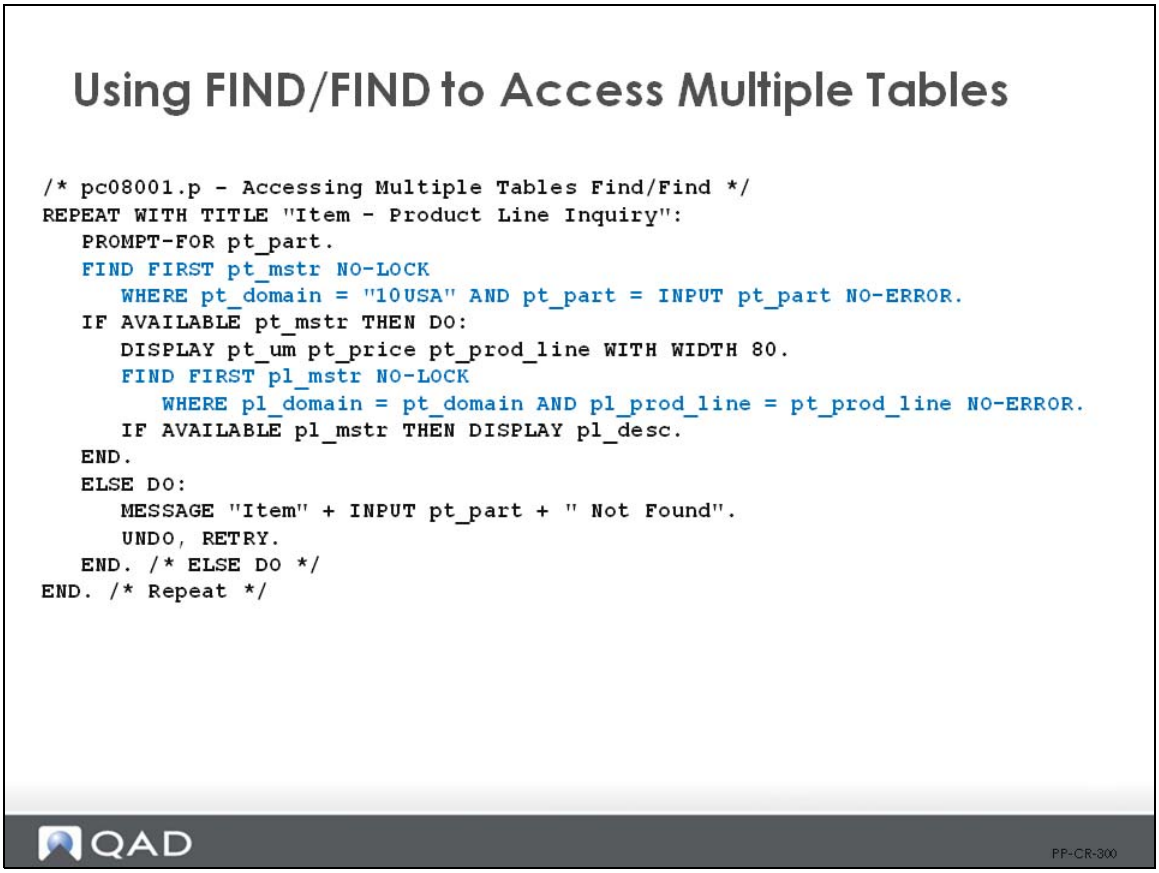
When creating a report or browse, use the entity relationship diagrams to decide which tables to use and the join relationships. If no relationship exists between two tables, a report showing such a relationship is not possible.

Exercise 8-1: Entity Relationships

1 Review the Customer Master relationships in the *QAD EA Entity Diagrams* manual.

2 How many purchase order lines (pod_det) records can a purchase order (po_mstr) have?

Using FIND/FIND to Access Multiple Tables



This sample program displays an item, its unit of measure, price, product line, and product line description. Not all of this information is in the Item Master table. The product line description is in the Product Line Master table.

The WHERE `pl_domain = pt_domain AND pl_prod_line = pt_prod_line` is the crucial part of the Product Line Master FIND statement that retrieves the record related to the specific Item Master record. The WHERE clause retrieves related data from multiple tables. FIND is used for the Product Line Master because a specific item number can be associated with one, and only one, product line.

You can access information from multiple tables with two FIND commands to retrieve a record in one table, and then use this record to retrieve another record from a related table.

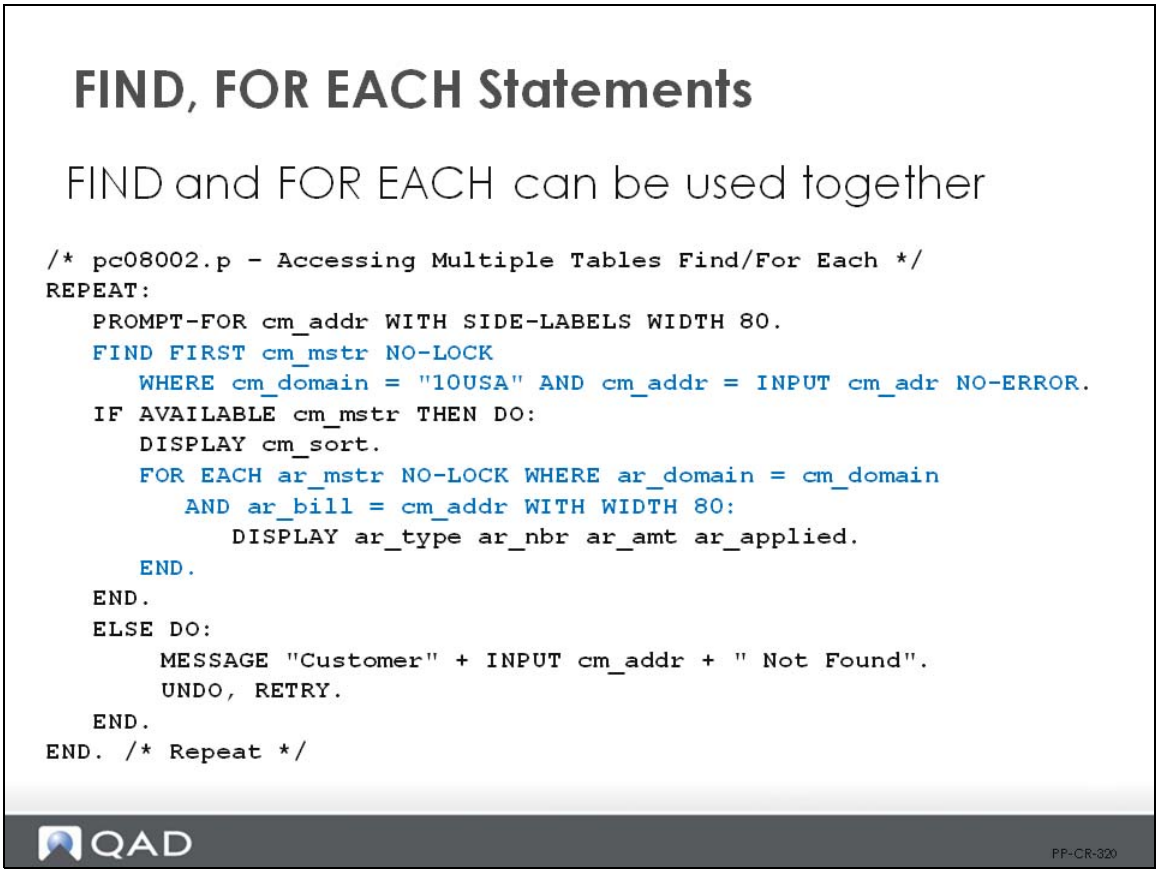
Since the FIND statement is used to retrieve one unique record in a table, the WHERE condition must uniquely identify one record.

Exercise 8-2: FIND/FIND

Write a program that prompts the user for a sales order number. Then display the order date, customer number, customer name, and address. Use the following hints:

- In the Sales Order Master table (so_mstr) display the so_ord_date field.
- Use 10USA as a valid domain.
- Link the tables using the so_cust and ad_addr fields.
- Locate the address in the Address Master table ad_mstr and display the fields ad_name and ad_line1.
- When prompted for a sales order, specify SO011204.

FIND, FOR EACH Statements



FIND and FOR EACH can also be used together.

Exercise 8-3: FIND, FOR EACH

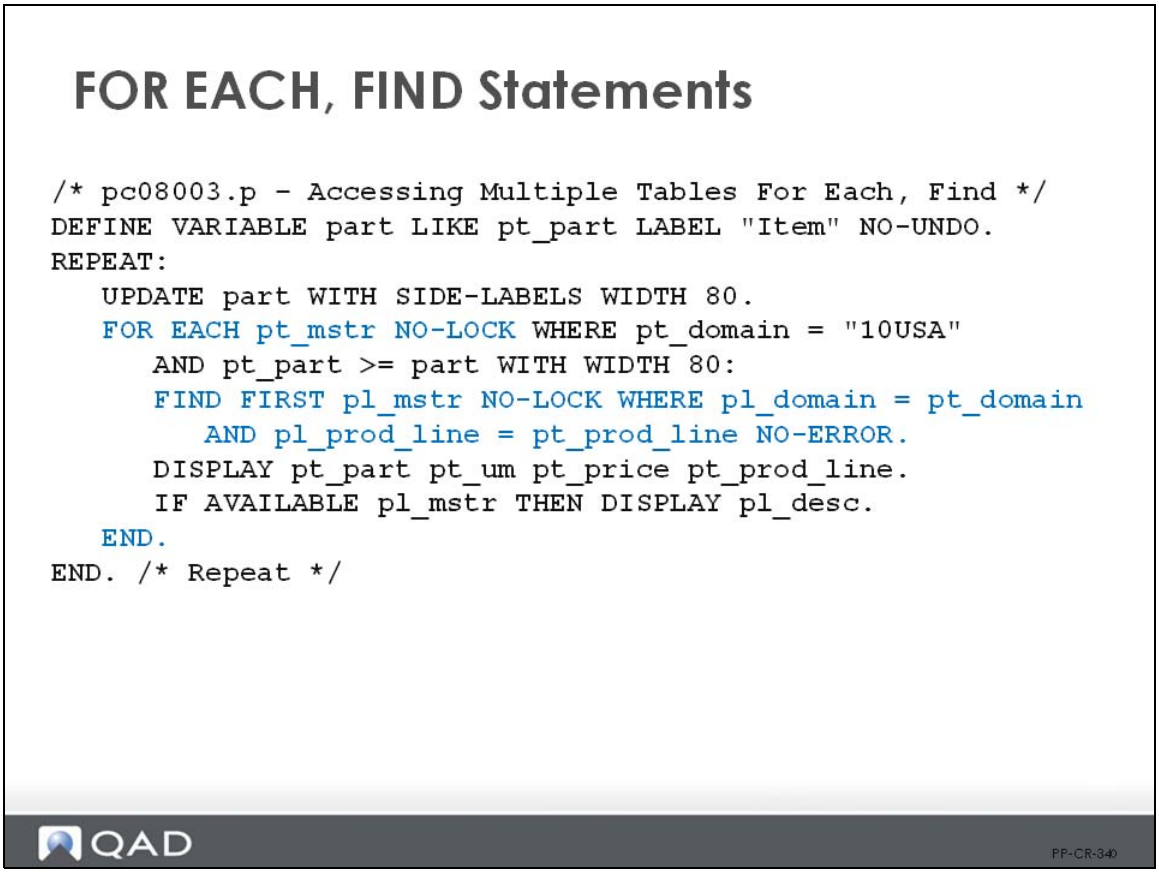
1 Write a new procedure that:

- Prompts for an item number
- Displays the item description and product line number (pt_desc1 and pt_prod_line)

Hint. Remember to use "10USA" as a valid domain in the WHERE clause. Item 01010 is a valid item number for this exercise.

2 Then display sales order line item information, sales order number, sales order line number, quantity ordered, and order price (sod_nbr sod_line sod_qty_ord sod_price).

FOR EACH, FIND Statements

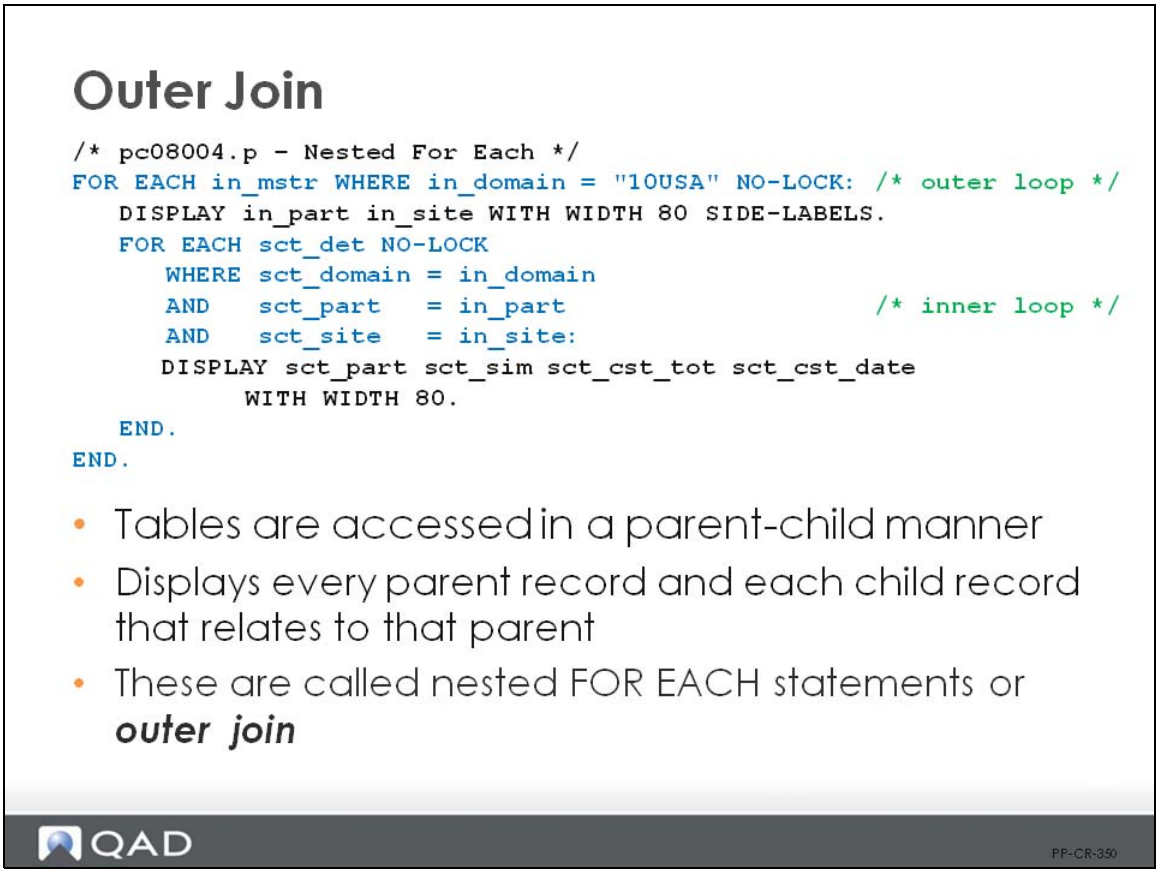


The combination of the FOR EACH and the FIND statement can be used in either order.

The procedure `pc08003.p` retrieves the Item Master table using the FOR EACH statement before completing a FIND for the Product Line Master table.

You can access information from multiple tables with the FOR EACH, FIND statement combination to retrieve a record in one table, and then use this record to retrieve another record from a different table that is related to the record specified in the FOR EACH statement.

Outer Join



Multiple FOR EACH statements may be appropriate for a specific report or inquiry. The intertable relationship should be specified within the WHERE clause of the record phrase.

You can access information from multiple tables with a FOR EACH statement followed by a second FOR EACH statement. These are called nested FOR EACH statements.

- The outer (or first) FOR EACH statement retrieves multiple records from a given table.
- In the inner (or second) FOR EACH statement, multiple records are retrieved from a different table that is related to the record specified in the outer FOR EACH statement.

These nested blocks are also known as an outer join.

The FOR EACH, FOR EACH combination is often used to display header and line item data for multiple purchase orders, work orders, batch processing jobs, and to list all sales quotes for multiple customers. The tables are accessed in a parent-child manner. For every parent record displayed, the child records that relate to that parent are displayed as multiple lines.

Exercise 8-4: Outer Join

Write a program that lists:

- The customer name, region, and last sale date
- Then list all sales orders along with the items, quantities sold, and prices of these items

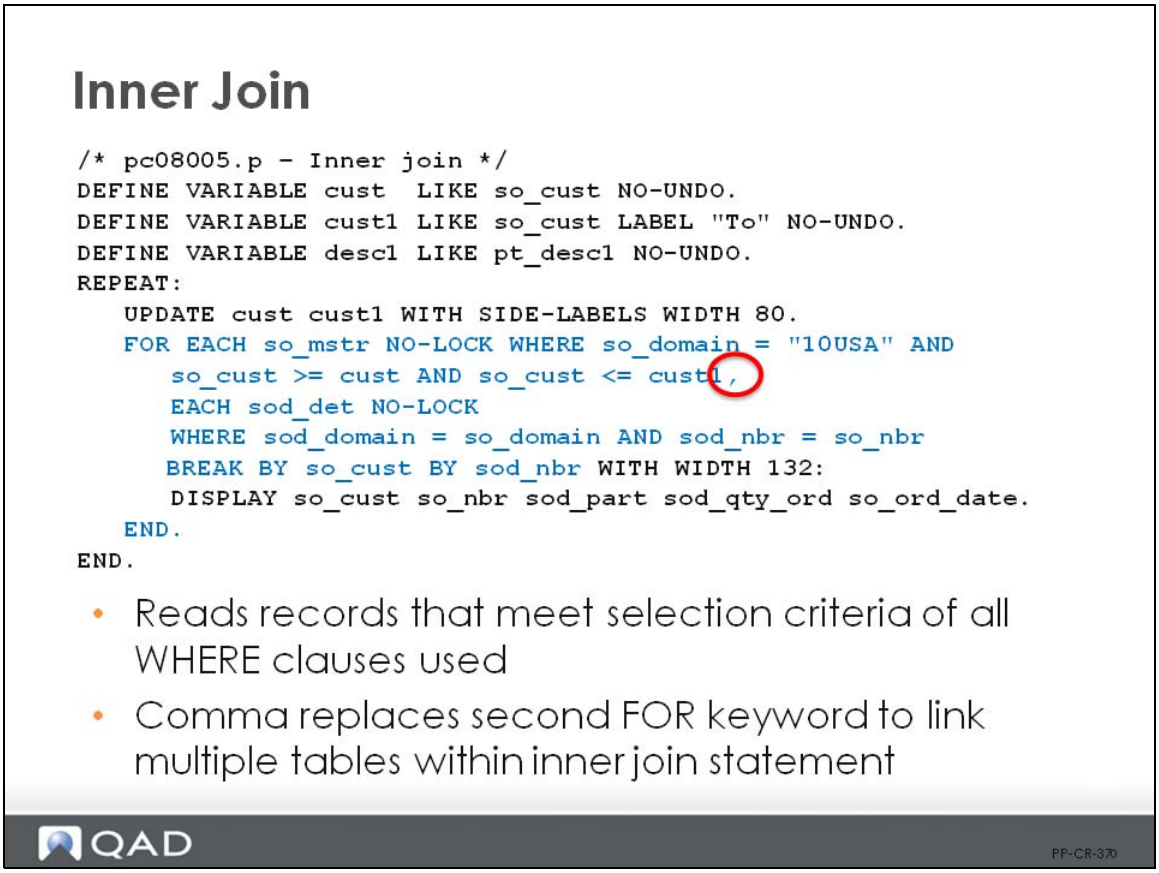
Hint. Set the domain to "10USA". You can use customer 10C1001 to test this program.

First Question

What tables have this information?

Customer Master (cm_mstr)	cm_addr cm_sort cm_region cm_sale_date
Sales Order Master (so_mstr)	so_nbr
Sales Order Detail (sod_det)	sod_line sod_part sod_qty_ord sod_price

Inner Join



Multiple tables can be joined using a WHERE clause, which allows fields to be displayed from any of the joined tables. The inner join enables you to sort on any field contained in the joined tables. The FOR EACH inner join reads only the records that meet the selection criteria of all WHERE clauses used.

Note Procedure performance may be affected since multiple tables are opened at the same time.

A comma replaces the FOR keyword to link multiple tables within the inner join statement.

To summarize the difference between an outer and inner join:

- The outer join displays every parent record (even those without any child records) and any child records that relates to the parent.
- An inner join displays only parents that have related child records.

Exercise 8-5: Inner Join

1 Modify procedure `pc08005.p` to sort by sales order date.

Note When you run the program, enter 10C1000 for Customer and 10C1001 for To Customer

Optional

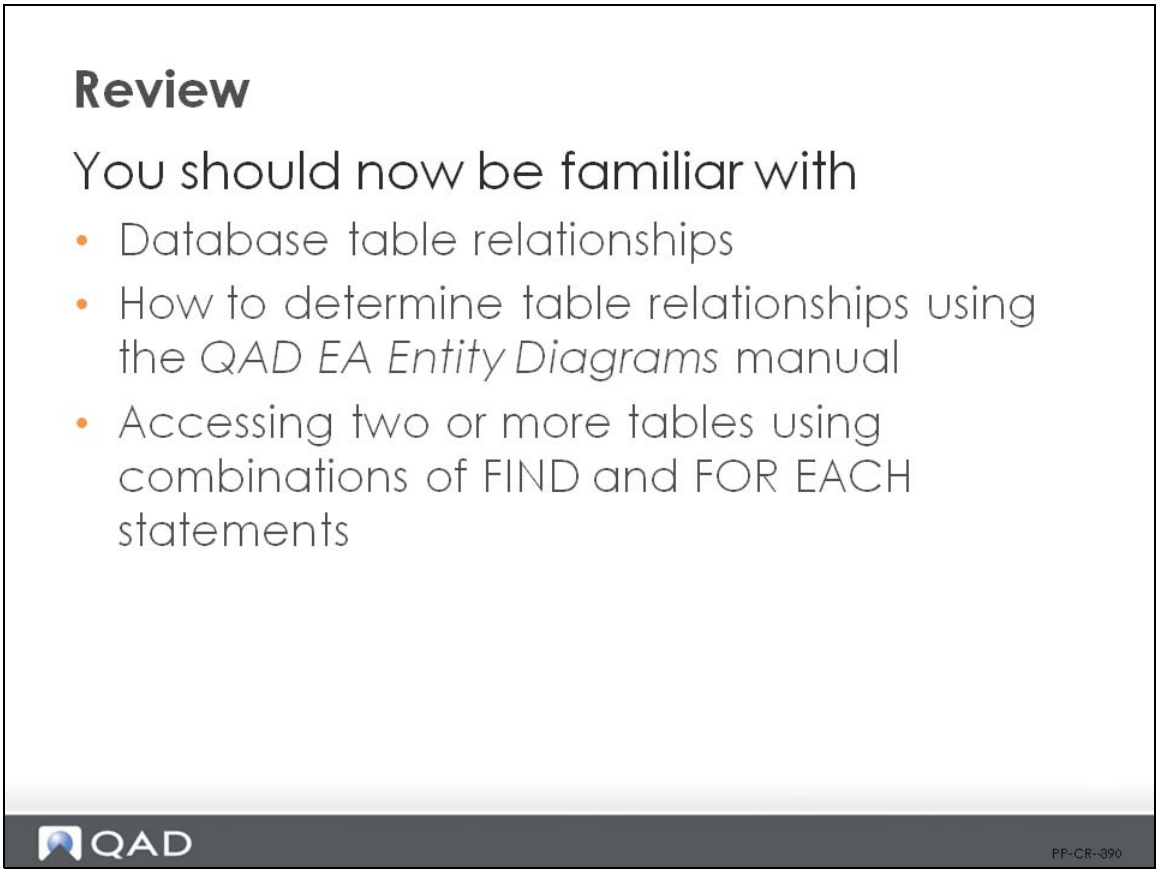
2 Write a procedure that displays purchase order detail information (pod_det) sorted by supplier.

Hint. Supplier is field po_vend. Set pod_domain = "10USA" in the WHERE clause.

3 Write a procedure that uses user-defined variables and the UPDATE statement to ask a user for a location; then list all items in that location.

Hint. Use the location detail table (ld_det) to find items in a location. Display the lot number, item number, and quantity on hand. Remember to set ld_domain = "10USA" in the WHERE clause.

Review



In this lesson, you were introduced to the relationships among database tables.

You should be able to access multiple tables by specifying intertable relationships with the WHERE clause of a record phrase. Different tables are related to each other by fields that reference common information.

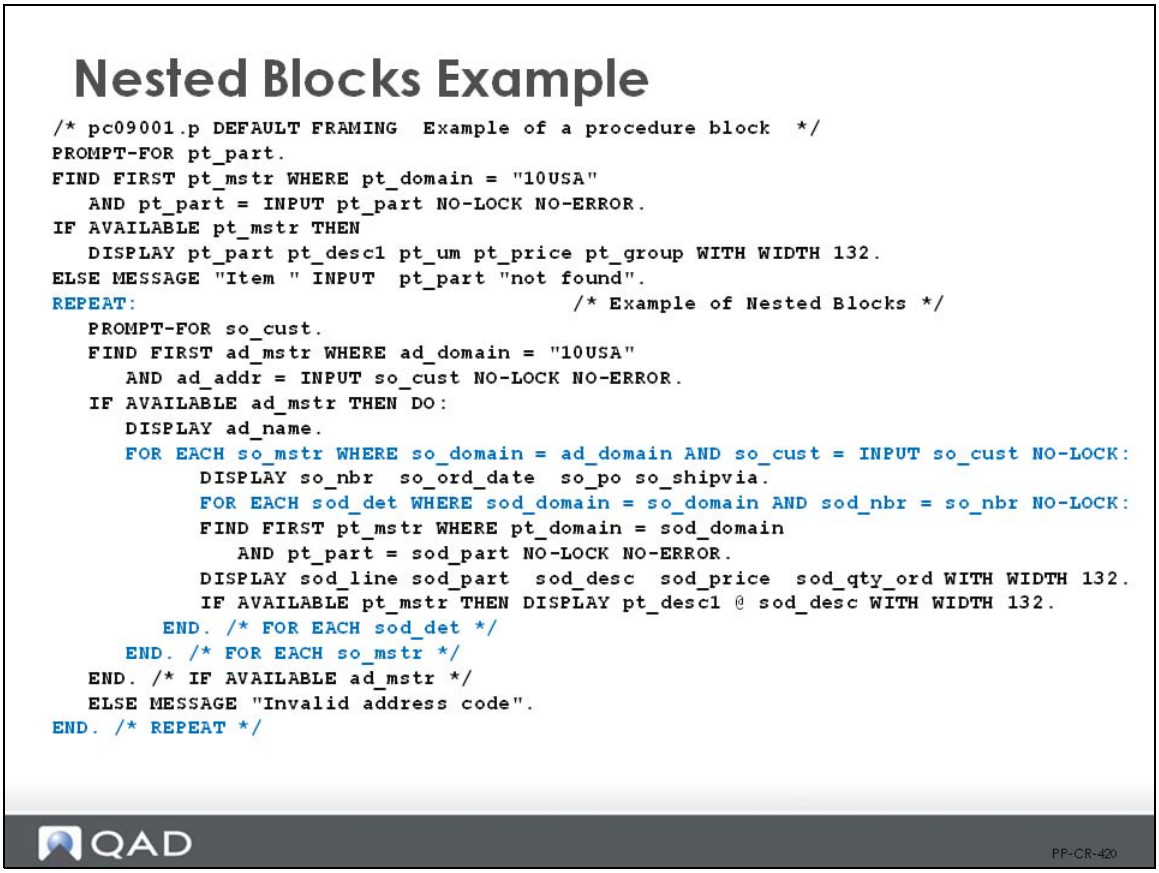
The use of the FIND versus the FOR EACH statement is determined by the type of tables, the relationships among those tables, and the output design. The intertable relationship is specified within the WHERE clause of the record phrase. The FOR EACH inner join allows additional flexibility in sorting information by allowing sorts on fields from different tables.

Lesson 9: Frame Control

Upon completion of this lesson, you will be familiar with the following topics:

- Progress default framing rules
- Frame names
- Basic frame control using the form and record phrases
- FORM statement
- How to prevent record flashing

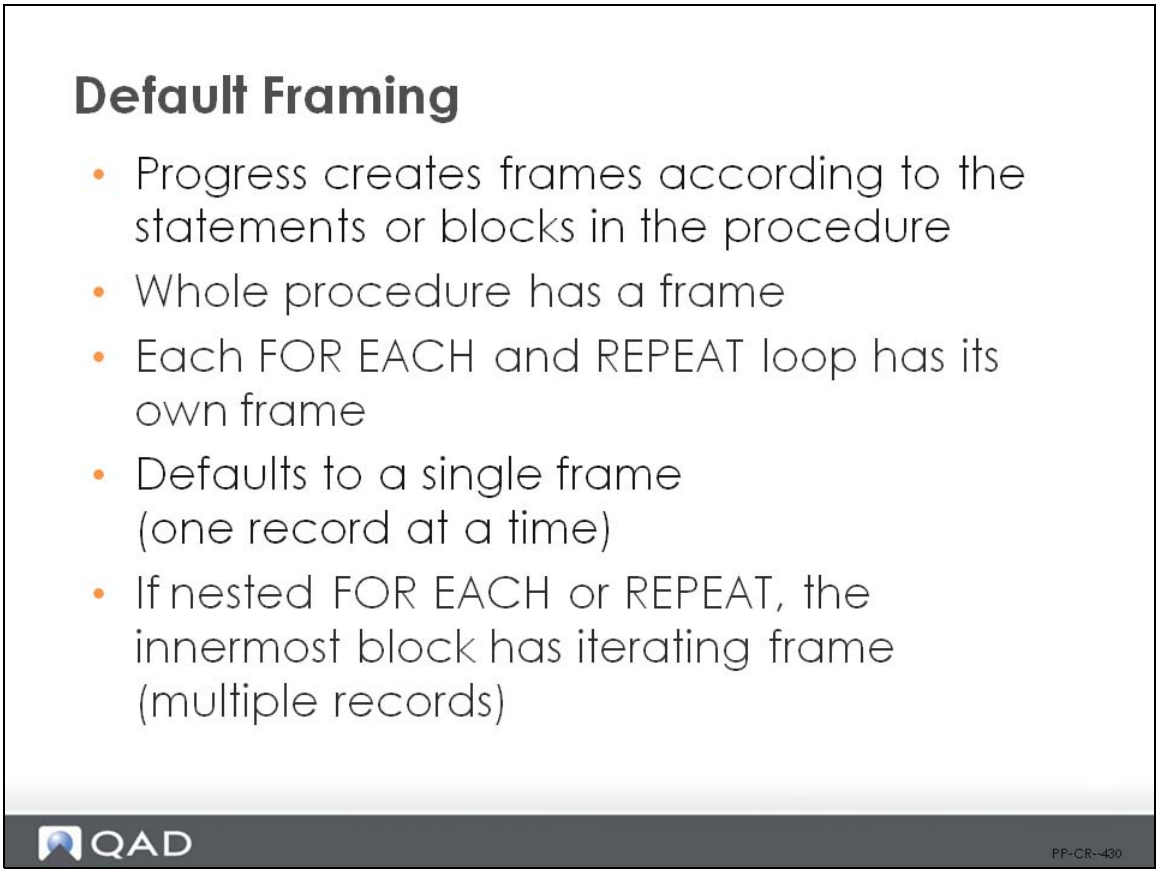
Nested Blocks Example



Progress accumulates any frame definitions that it encounters within the scope of the frame.

They are not built up as the code is processed line-by-line. Progress builds the entire frame definition when the code is compiled prior to code execution.

Default Framing



Progress creates default frames according to the statements you use in your procedure. Each frame creates a box when displayed on a terminal.

A frame is assigned to a procedure block, which is a group of statements that does not belong to a FOR EACH or REPEAT block. A procedure that does not use any FOR EACH or REPEAT statements will have at least one frame.

FOR EACH and REPEAT statements create iterating blocks with frame properties. A frame is created whenever a FOR EACH or REPEAT is used. Frames can be nested, depending on the location of the procedure, FOR EACH, and/or REPEAT block.

By default, Progress creates single frames that display one record at a time. A DOWN frame is created for the innermost iterating FOR EACH or REPEAT block. This results in a columnar display of multiple records when using FOR EACH or REPEAT.

To determine if the frame is a down frame, use this checklist:

1. Is the block an iterating block?
2. Is the default frame scoped to the block?
3. Is it the innermost iterating block? (No other block are nested inside it.)

If all answers are yes, it will be a down frame. All others will be a one-down frame.

Exercise 9-1: Framing

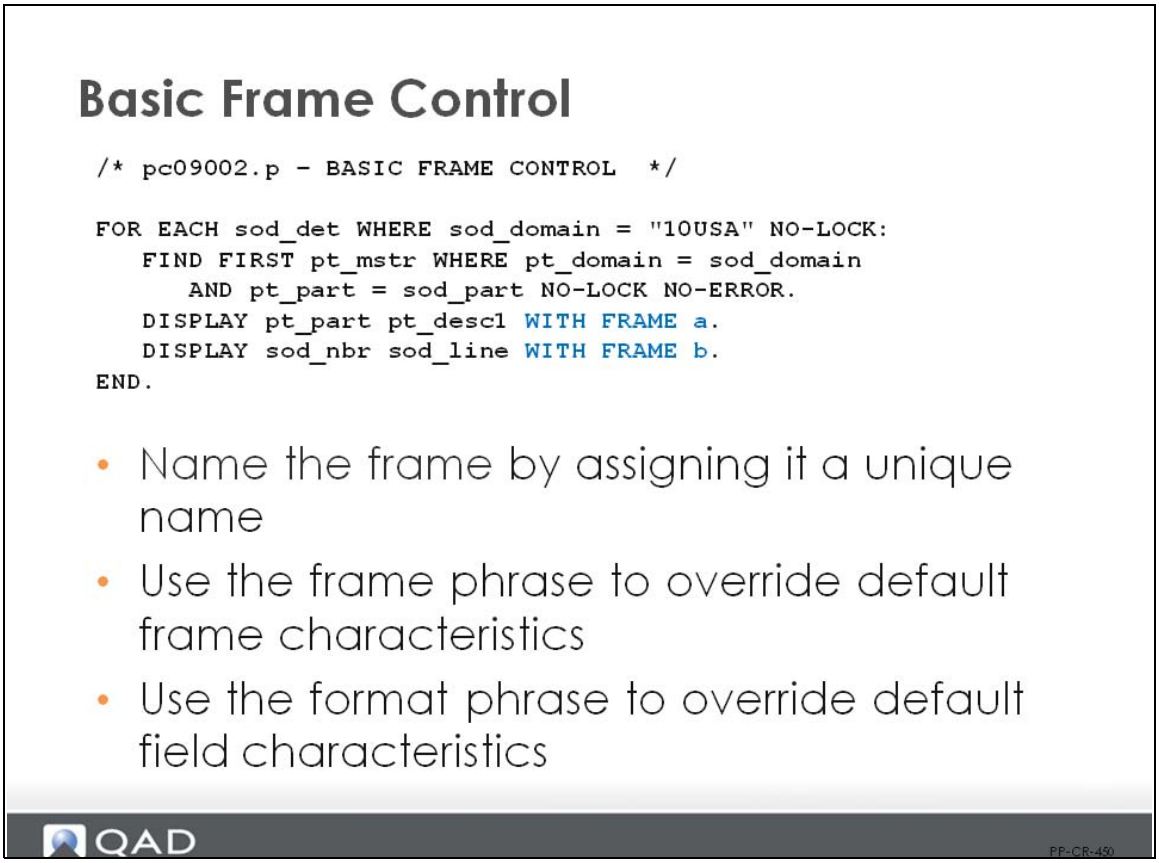
1 Review procedure `pc09001.p`.

2 How many frames would be created for the procedure?

3 Execute the procedure to verify your answer. When prompted, you can specify item number 01010 and customer 10C1001.

Note In the results, the item data and sales order data are independent; they are not related.

Basic Frame Control



One way to override the default frame characteristics is to use frame names. A frame name can be assigned within a frame phrase to create a new frame. Procedure `pc09002.p` would normally have frame for the FOR EACH block. A second frame is created because the frame name **b** is assigned to the second display statement.

New frames are defined by assigning them a unique name. Default frames are created for REPEAT, FOR EACH, and procedure blocks, but are not named. When a DO block is within a REPEAT, FOR EACH, or procedure block, its default frame comes from the closest REPEAT, FOR EACH, or procedure block. To override default frame characteristics, use a frame phrase. To override default field characteristics, use a format phrase.

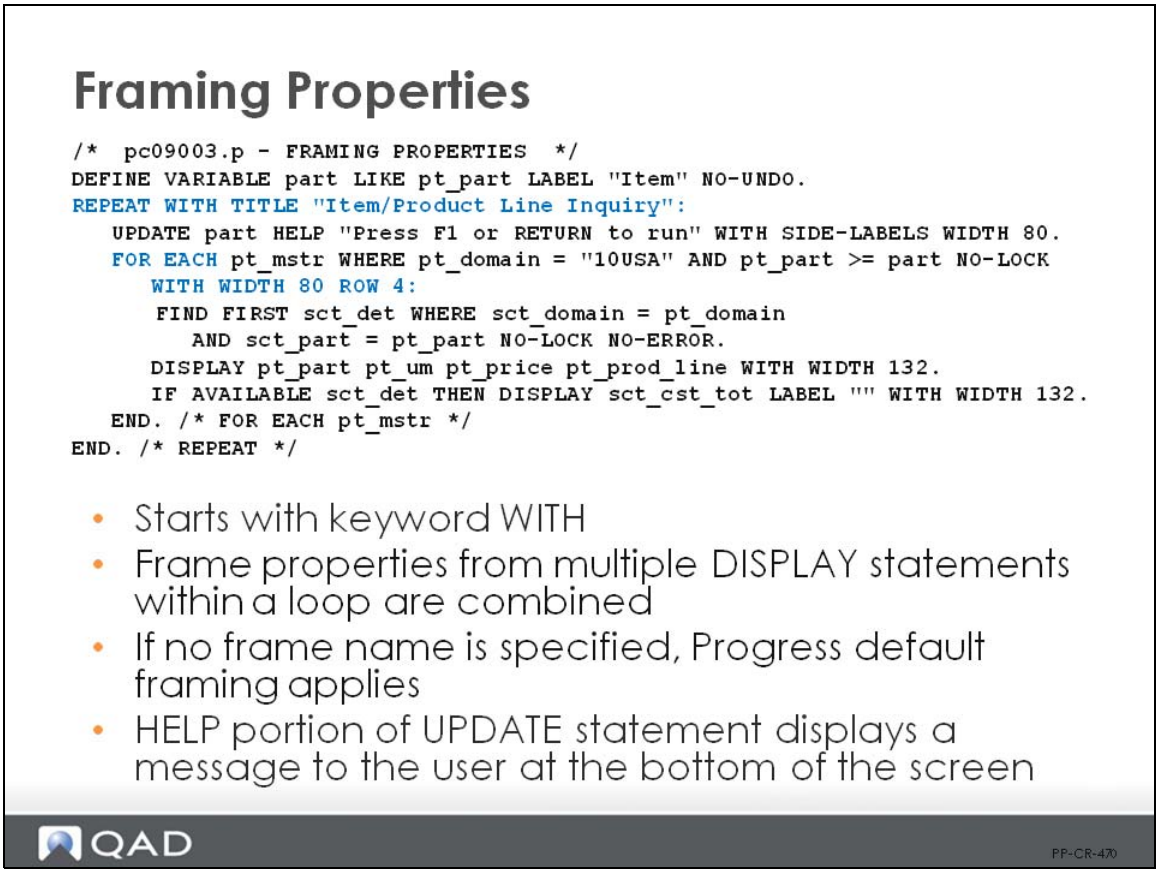
The frame phrase specifies the overall layout and processing properties of a frame when specified on block header statements (DO, FOR EACH, or REPEAT). It also specifies the default frame for data handling statements (DISPLAY, PROMPT-FOR, UPDATE) within the block. Frame phrases can also be used in individual statements to indicate the specific frame for the statement to use.

Exercise 9-2: Basic Frame Control

Modify procedure `pc09002.p`:

From: `DISPLAY pt_part pt_desc WITH FRAME a.`
To: `DISPLAY pt_part pt_desc.`
From: `DISPLAY sod_nbr sod_line WITH FRAME b.`
To: `DISPLAY sod_nbr sod_line.`

Framing Properties



Framing characteristics are defined at the end of a data handling statement starting with the key word WITH. Frame properties from multiple DISPLAY statements within a loop are combined. A frame is centered with a title, even though the CENTERED and TITLE options appear on different DISPLAY statements.

If no frame name is specified, Progress default framing applies. When nested FOR EACH statements are used, each one should use a different frame.

The HELP portion of the UPDATE statement acts like a message at the bottom of the screen and directs the user to what action to perform.

Exercise 9-3: Framing Properties

1 What is the effect of specifying ROW 4?

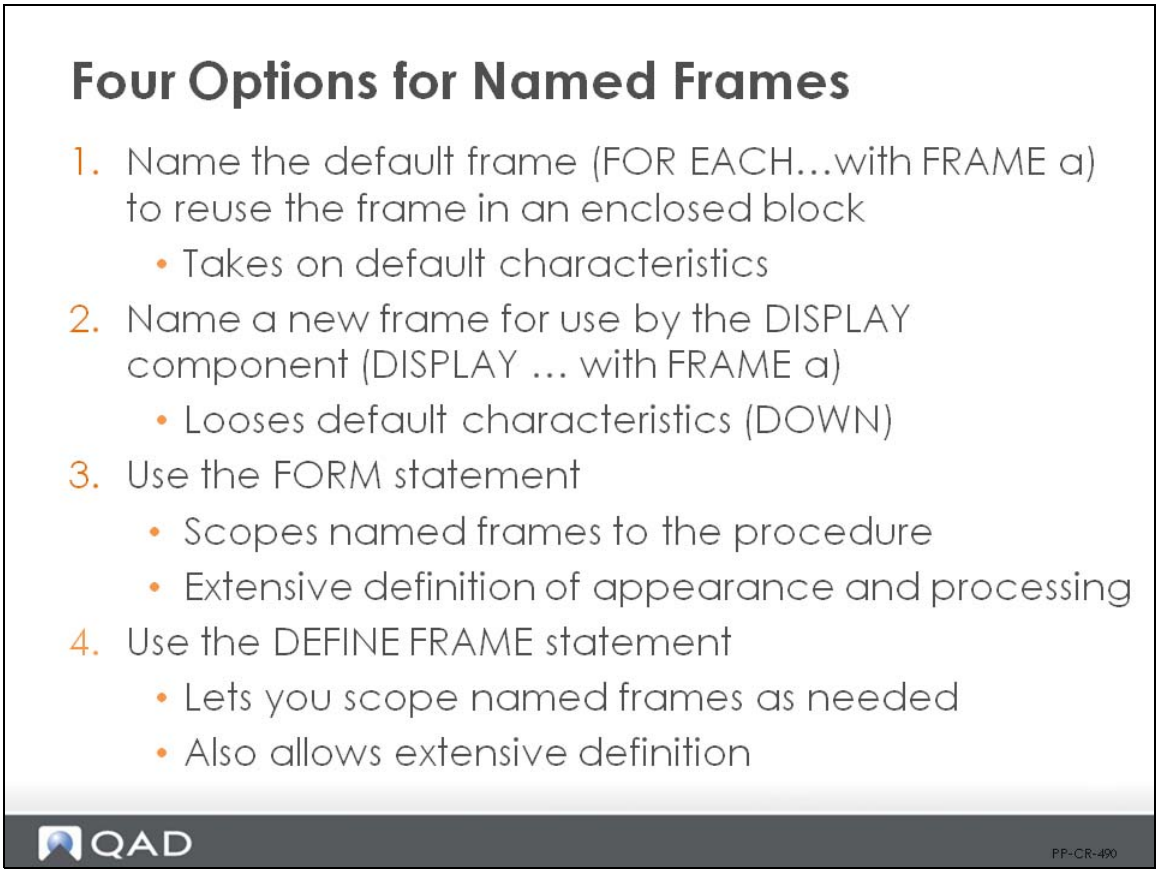
2 Modify procedure `pc09003.p`:

From:
```
DISPLAY pt_part pt_um pt_price pt_prod_line.
```

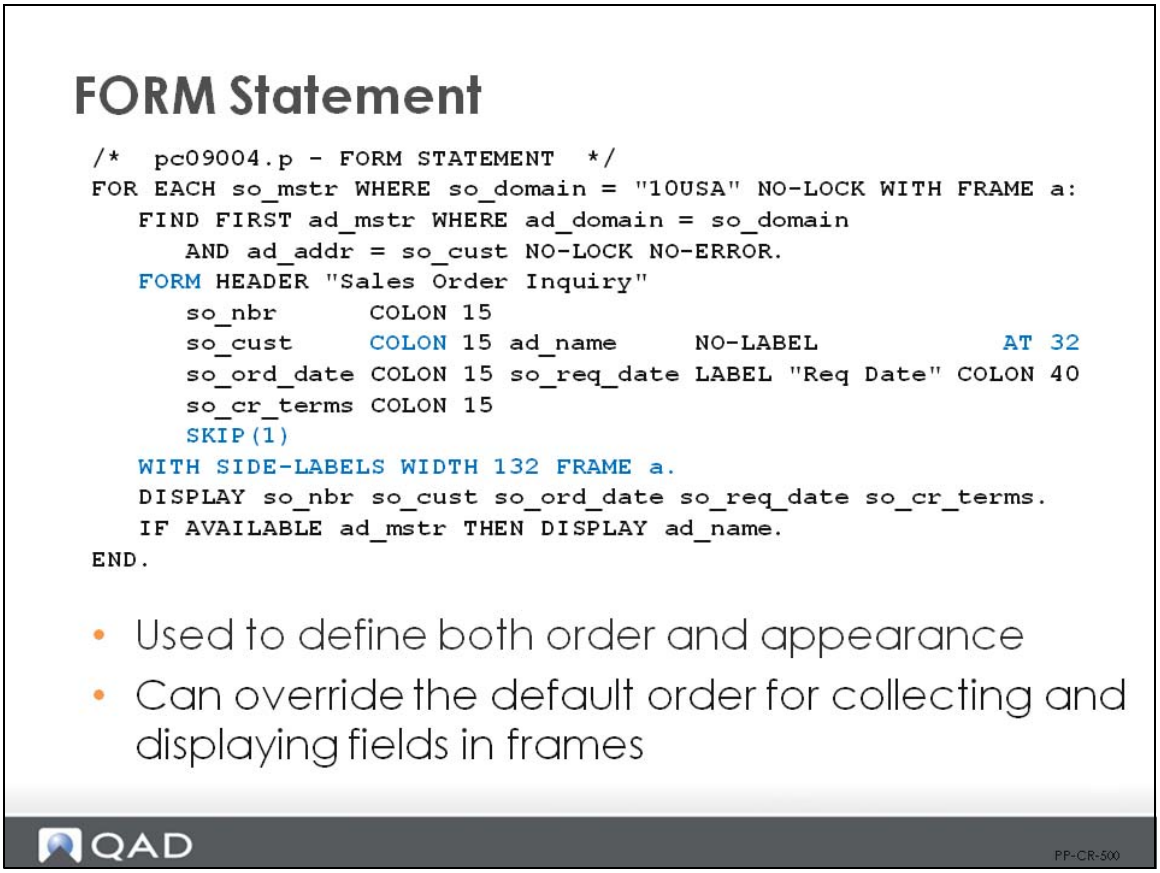
To:
```
DISPLAY pt_part pt_um pt_price pt_prod_line
WITH TITLE COLOR NORMAL
"Item /Product Line Inquiry ".
```

Hint. Use item 01010 when you run the program.

Four Options for Named Frames



FORM Statement



The FORM statement is used to specify field and frame display characteristics in a single statement. Use it to define the order and appearance of fields for a frame at one time. The FORM statement overrides the default frame characteristics and can be used to override the default order in which fields are collected and displayed into frames.

When you use a data-handling statement (DISPLAY, PROMPT-FOR, or UPDATE), you can name the frame to use or use the default frame.

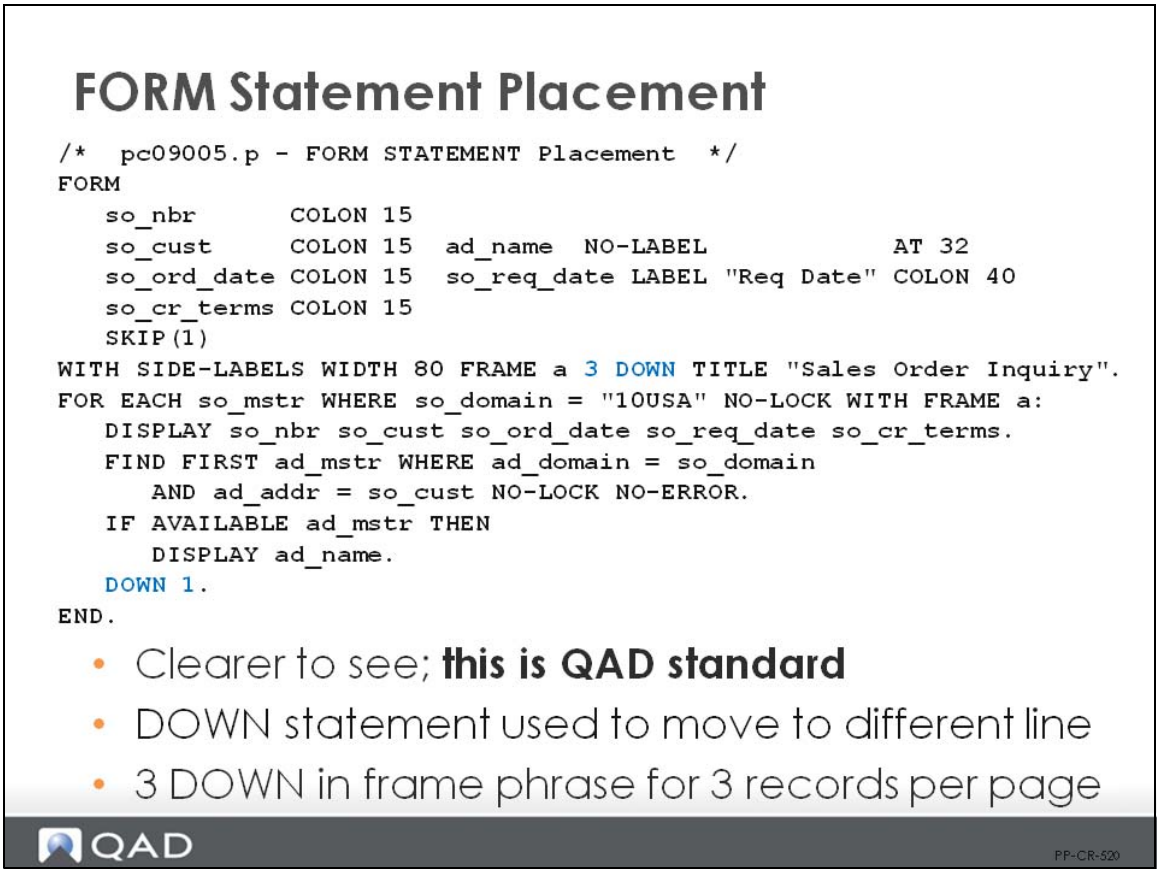
When you specify a frame on the FOR EACH statement, all of the statements in the procedure use the FORM definition of the frame.

Note Although you can embed a FORM statement inside any block of code, QAD does not recommend this approach for managing a form definition. Instead, the form definition should appear outside the code block as illustrated in `pc09005.p` on the next page.

Exercise 9-4: FORM

1. Modify procedure `pc09004.p` and add so_ship to the DISPLAY statement.
2. Modify this procedure so that it also displays the following sales order detail information: line number, item number, item price, order quantity, and quantity shipped.
3. Notice the difference between HEADER and TITLE.

FORM Statement Placement



This program illustrates how to segregate FORM statements at the beginning of a procedure.

The DOWN statement is used to move to a different line.

The example `pc09005.p` uses the frame phrase to specify 3 DOWN and the frame name.

Exercise 9-5: FORM Placement

1 Modify procedure `pc09005.p`:

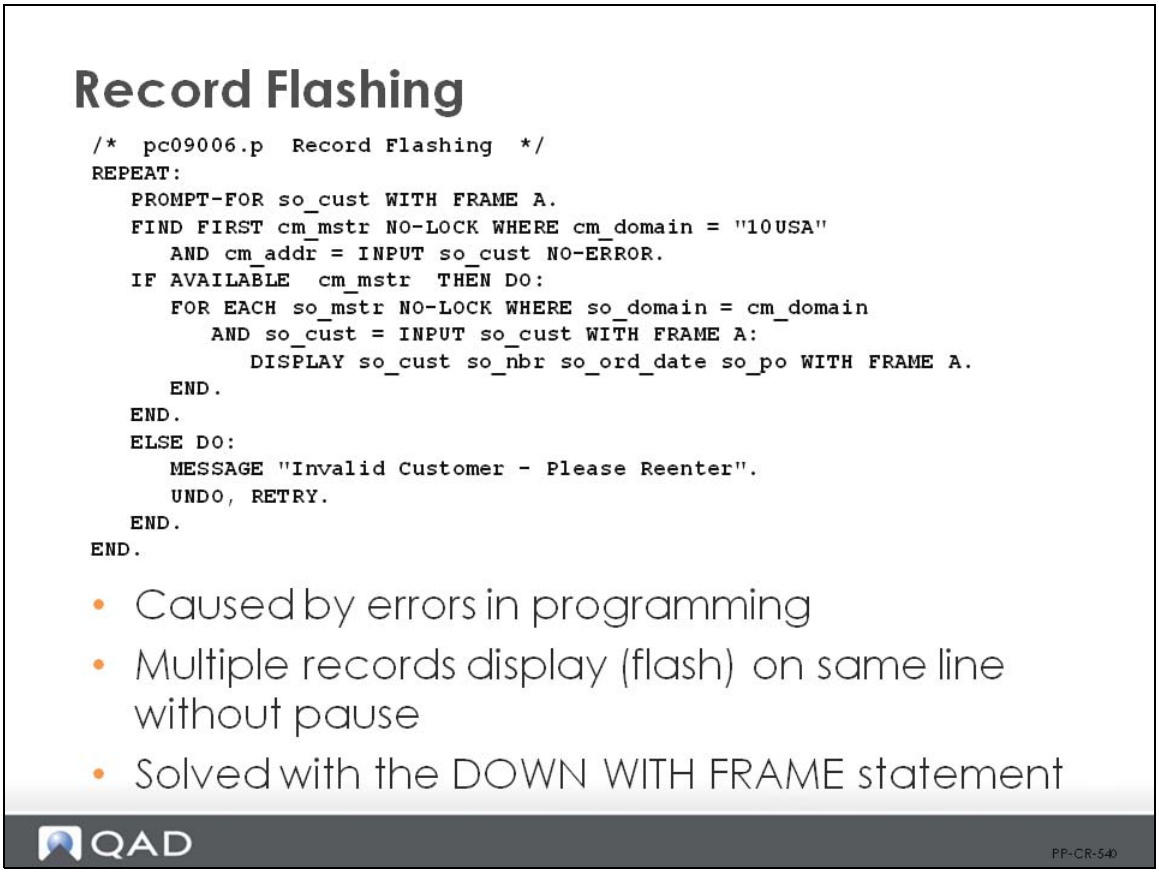
From: `WITH SIDE-LABELS WIDTH 80 FRAME a 3 DOWN.`
To: `WITH SIDE-LABELS WIDTH 80 FRAME a.`

2 Comment out the DOWN 1 statement

3 Rename the frame for the FOR EACH statement. Does this produce a better result?

4 Describe the easiest way to avoid the problems illustrated by these changes.

Record Flashing



Errors in programming can result in the display multiple records on the same line without a pause between each new record. This causes multiple records to flash very quickly in the same place.

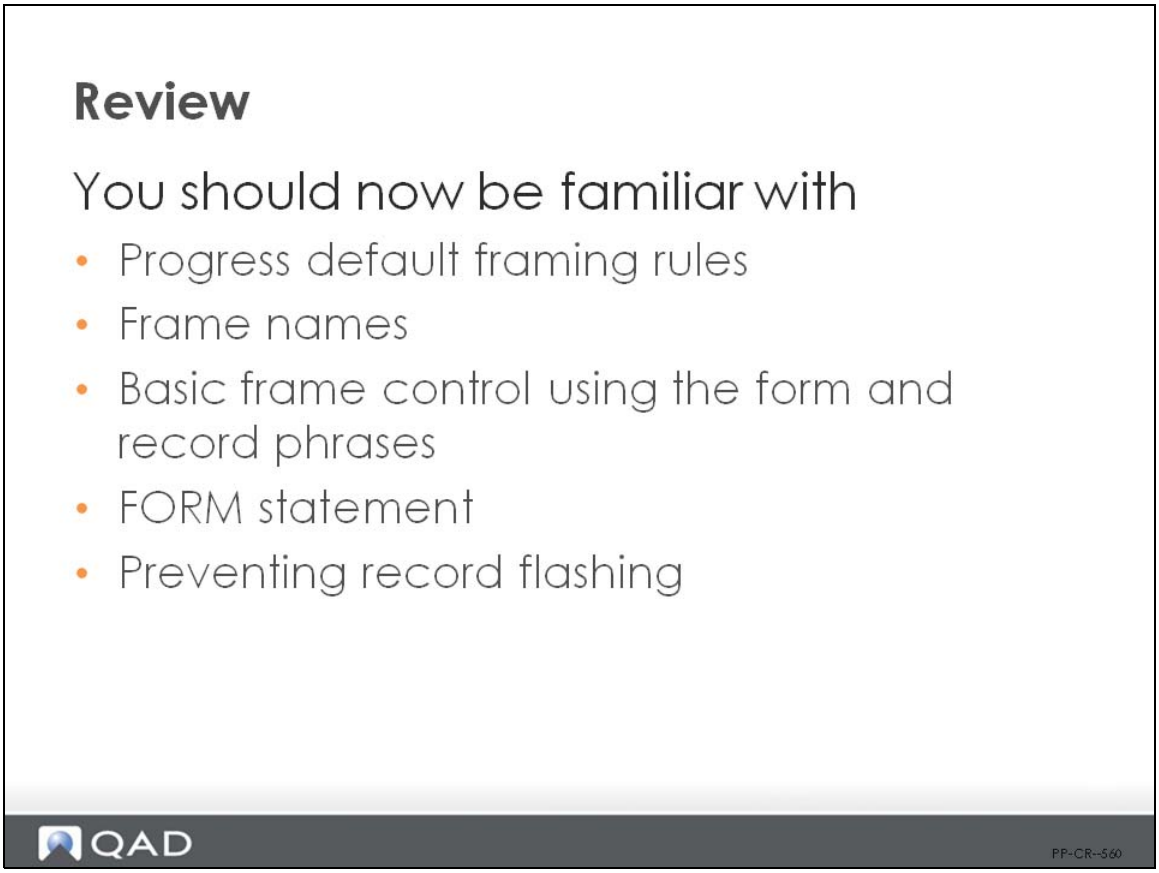
Record flashing is frequently caused by the inappropriate use of frame phrases and display-handling statements. It is most easily avoided by using the default Progress framing and adding frame control only as required.

Exercise 9-6: Record Flashing

Revise procedure `pc09006.p` to prevent the records from flashing.

Hint. Use customer 10C1001 when testing the program.

Review



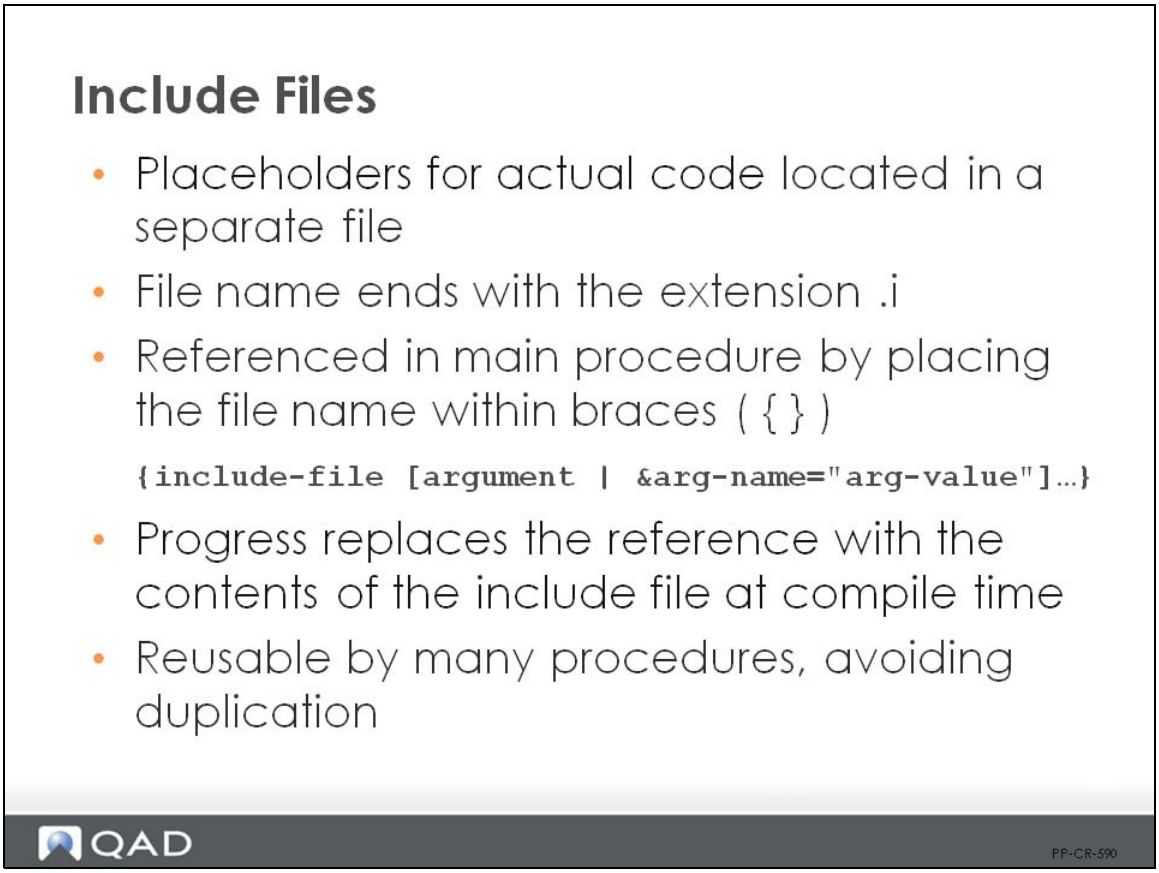
In this lesson, you learned how frames are created and how to control frame handling. You should now be able to evaluate a simple procedure and determine its default framing.

- Default frames are assigned to procedure blocks as well as FOR EACH and REPEAT statements.
- Frames can be nested. The innermost iterating block creates a DOWN frame.
- You can create new frames by using the frame phrase to assign a frame name.
- Frame properties are additive within Progress procedures. Progress evaluates and combines all of the frame specifications for a frame.
- You can use the FORM statement to combine field and frame specifications within a single statement.
- The inappropriate use of frame control can result in record flashing.

Lesson 10: Using Include Files

Upon completion of this lesson, you will be familiar with using standard QAD EA include files to create consistent reports.

Include Files

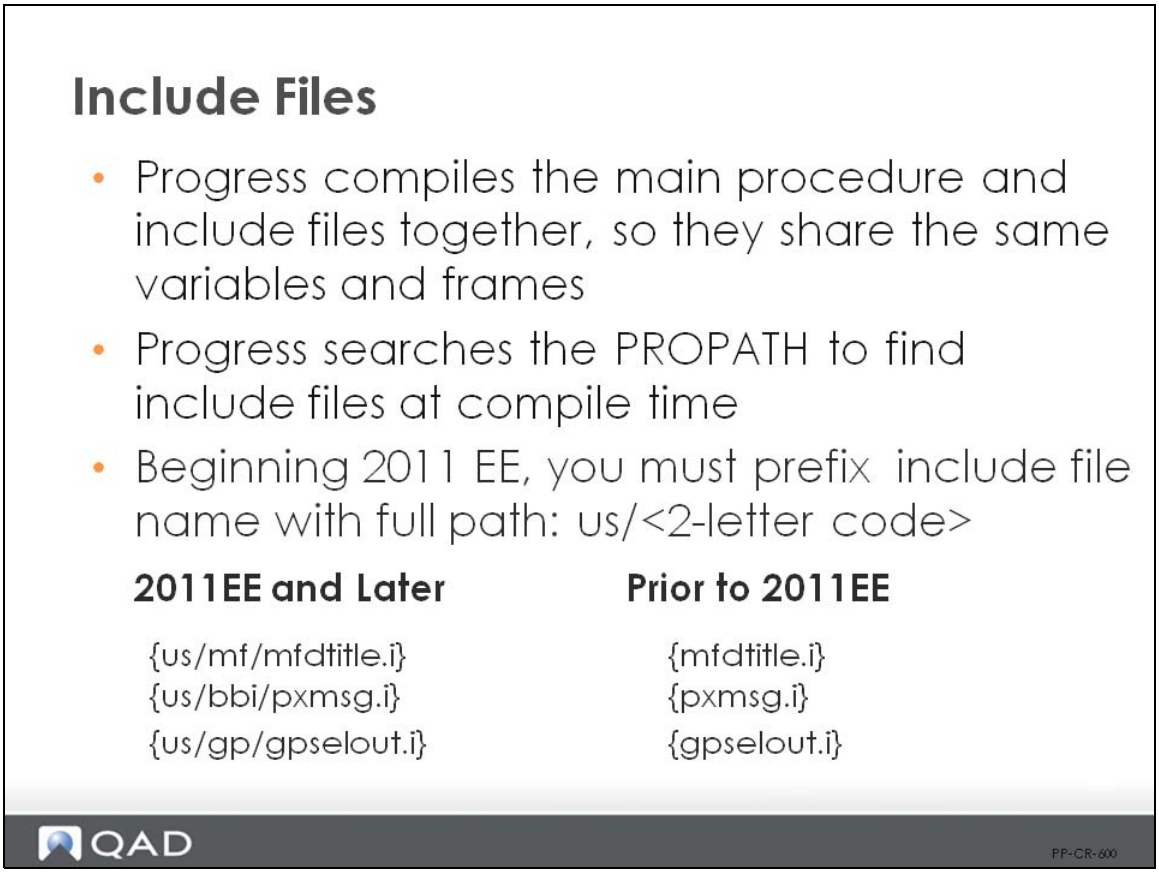


An include file is a portion of a procedure or a complete procedure that is placed into a separate file. The use of include files is one method for reuse of code since it allow several procedures to use the same code to perform the same functions.

Use include files to avoid duplicating a common set of statements within one or more procedures. An include file is referenced within a procedure by placing the table name within braces ({ }).

Progress retrieves the contents of include files when programs are compiled. Although it is not necessary for include files to end with the `.i` extension, this convention is suggested by Progress Software Corporation.

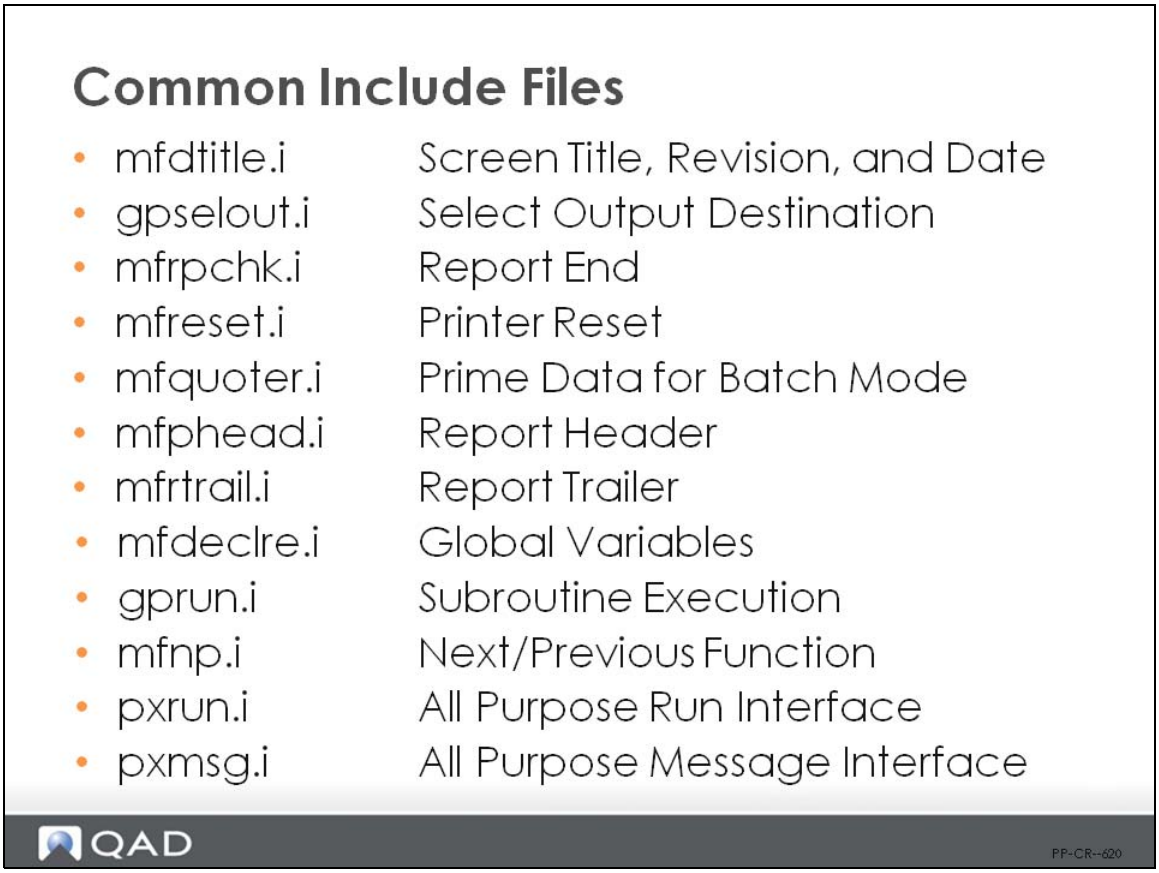
Include Files



Progress retrieves the statements in a file and compiles them as part of the main procedure. It looks in the PROPATH, which is a character string that contains the list of directories that Progress searches for procedures and files. See “Manipulating the PROPATH” on page 206 for details.

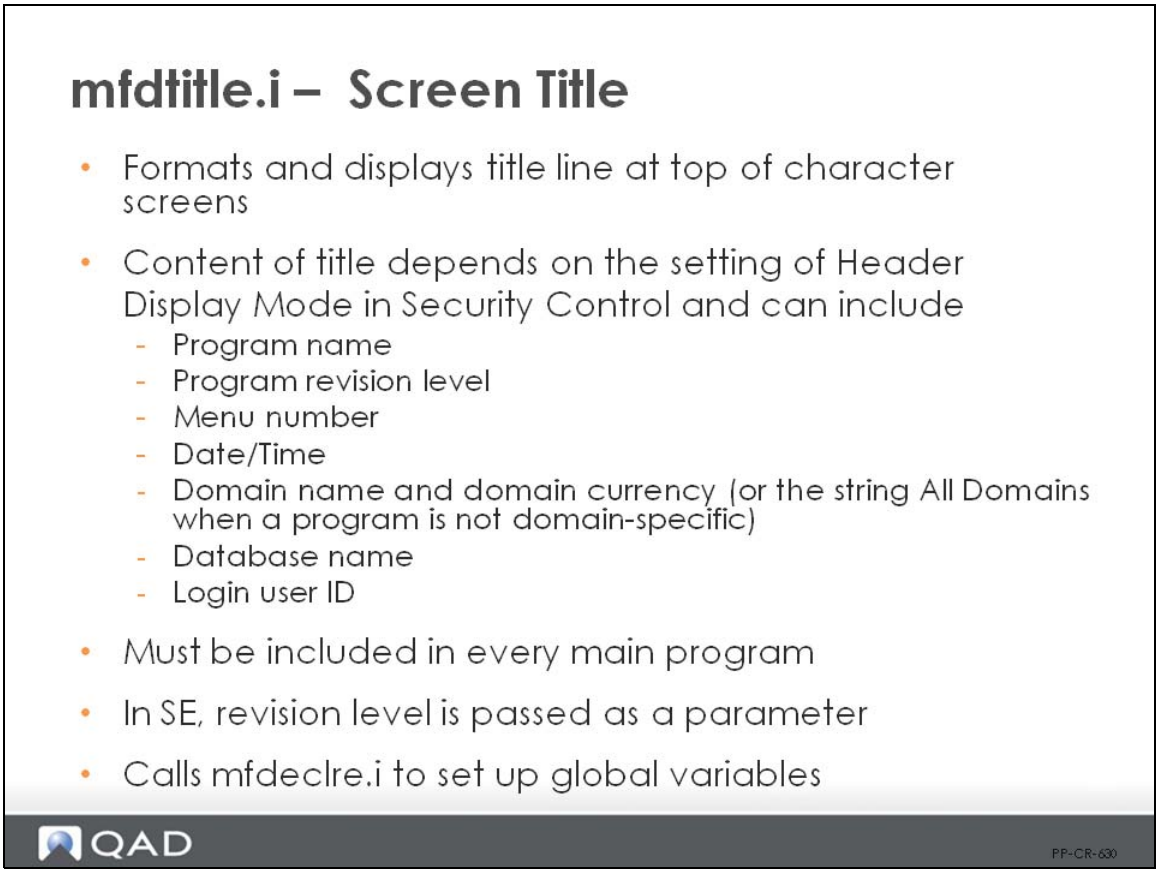
Note Beginning with QAD 2011, include files must be prefixed with a full path. See “QAD Enterprise Applications Directories” on page 30.

Common Include Files



The first seven include files on this list are used for managing various aspects of simple reports and inquiries. If you are planning to use the Reporting Framework for designing your reports, you do not need to use these include files, since the Report Designer builds reports differently. This information is provided for students who want to write their own ad-hoc reports and take advantage of standard QAD processing features.

mfdtitle.i – Screen Title

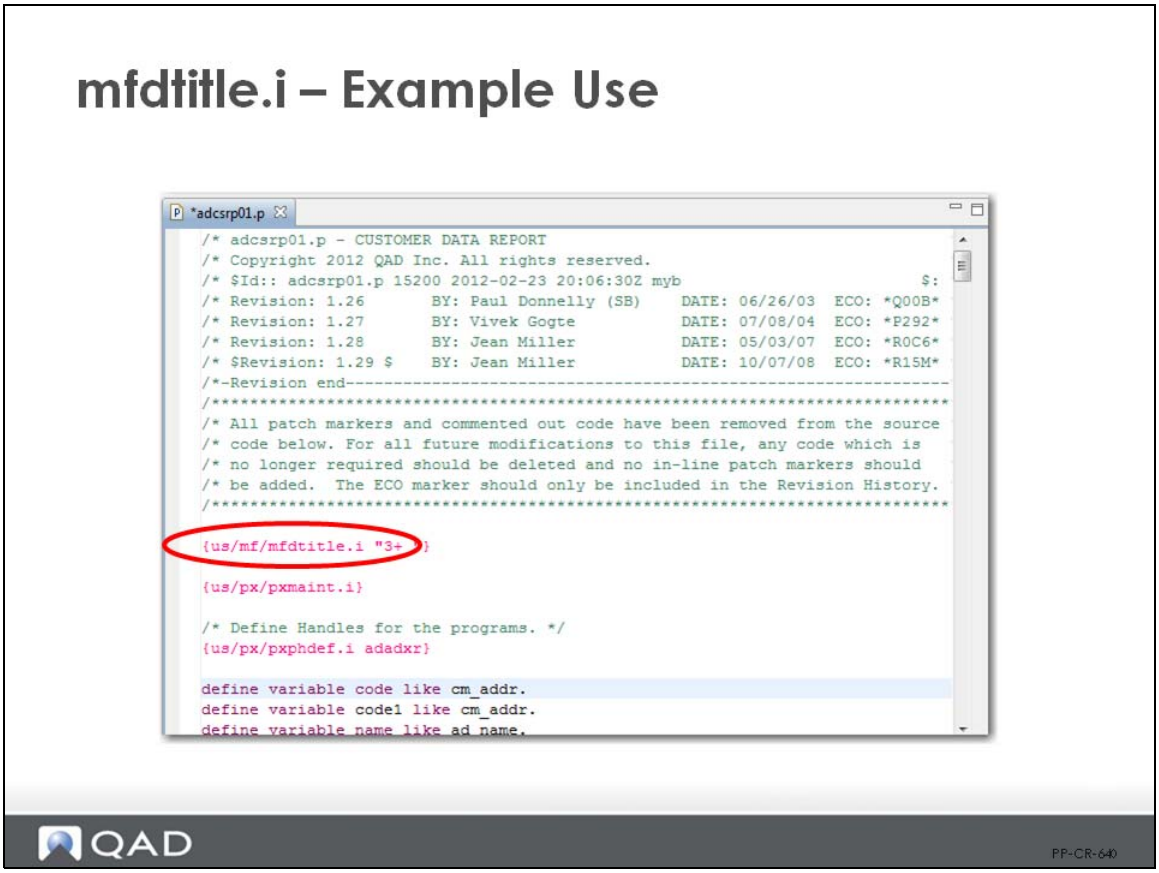


You can display standardized titles on the input frame by using the include file `mfdtitle.i`. In SE, this procedure passes the revision level to `mfdtitle.i` by putting the revision level in quotes (“a”). The title is obtained from the menu system.

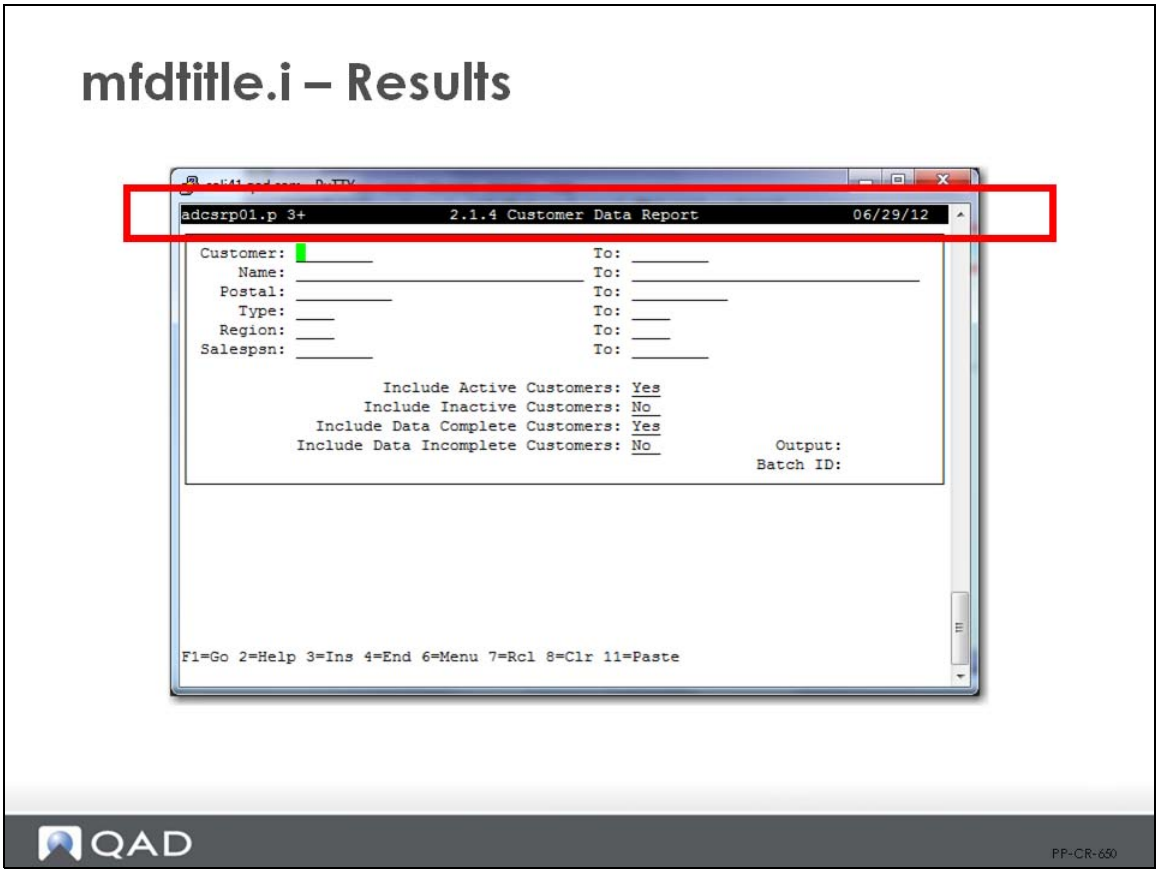
Use `mfdtitle.i` to display the program name in the window title bar, QAD EA menu bar, and toolbar icons. To conform to other standard include fields, all input fields used for selection criteria should be displayed within frame a.

For standardized inquiries, use the frame phrase options NO-UNDERLINE WIDTH 80 for the input frame. Display frames should also use WIDTH 80.

mfdtitle.i – Example Use

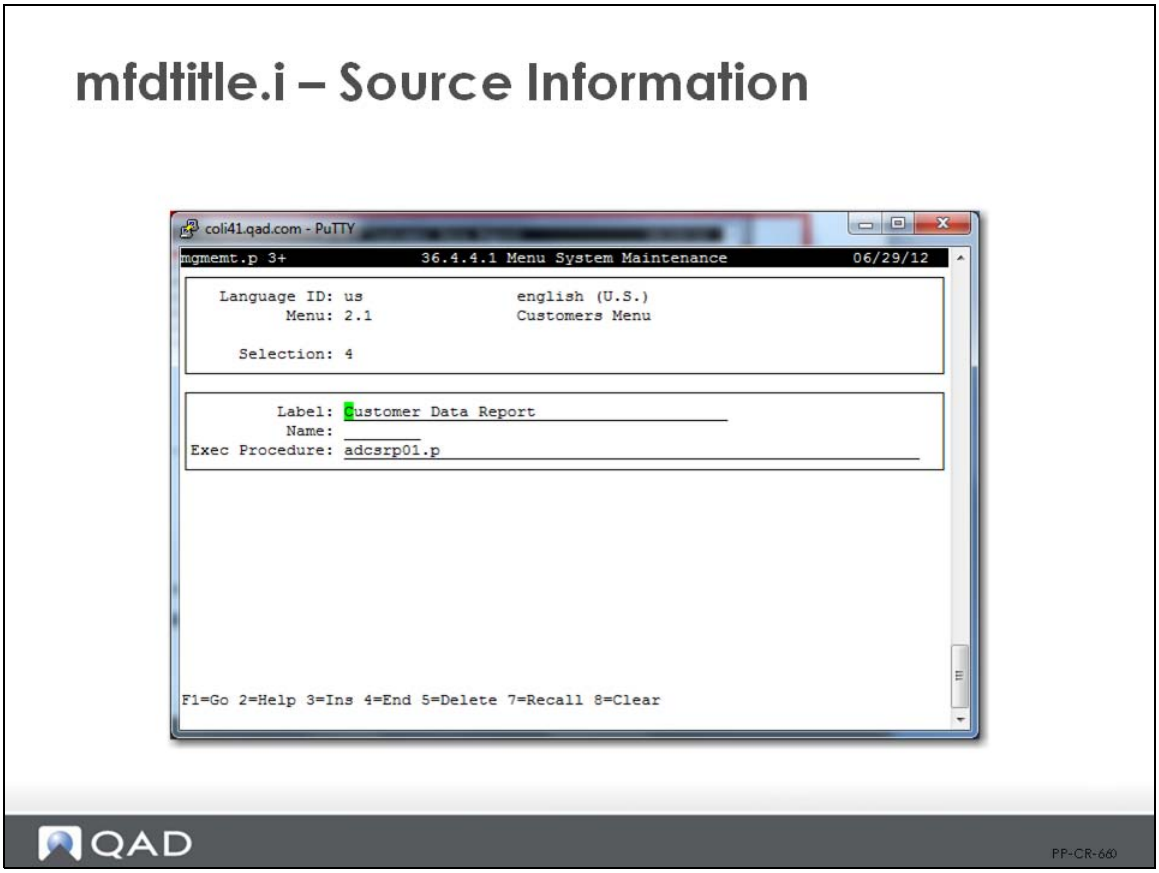


mfdtitle.i – Results



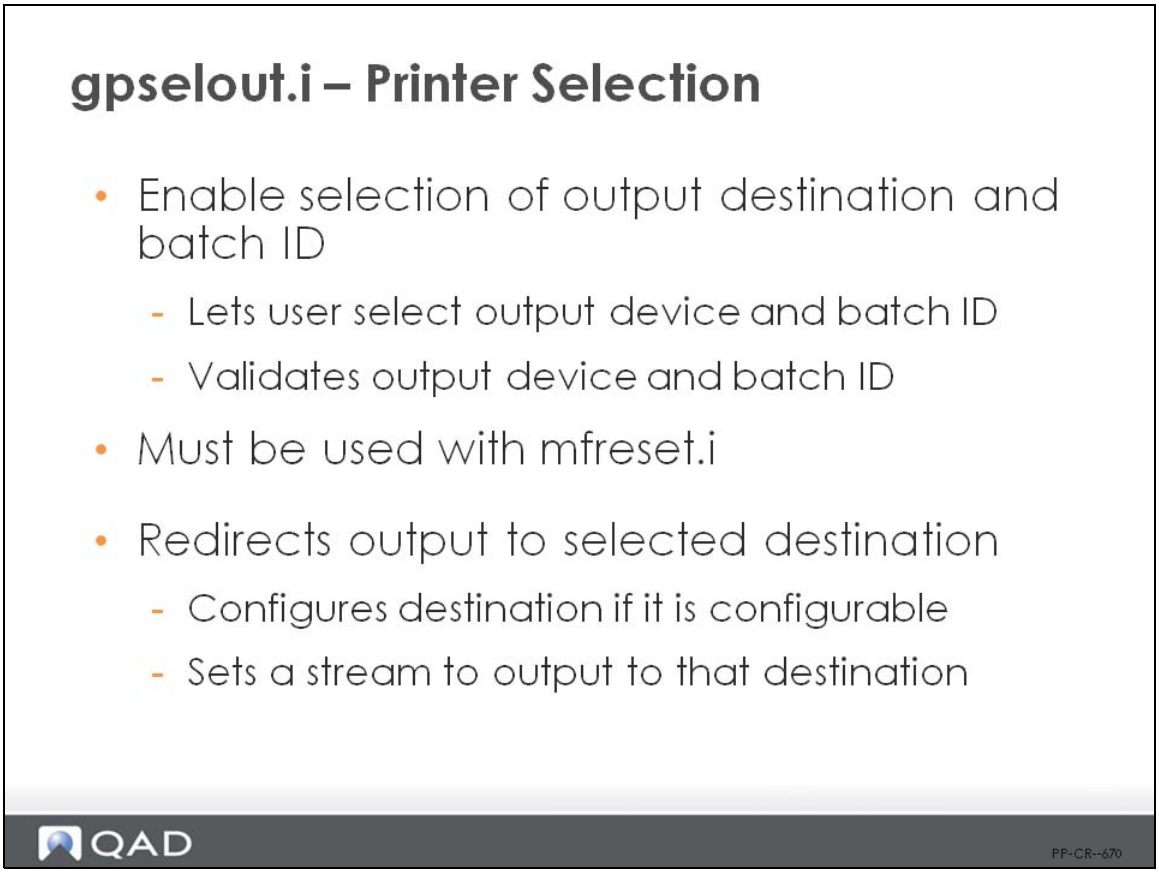
This screen shows how the title of a report appears as formatted by `mfdtitle.i`.

mfdtitle.i – Source Information



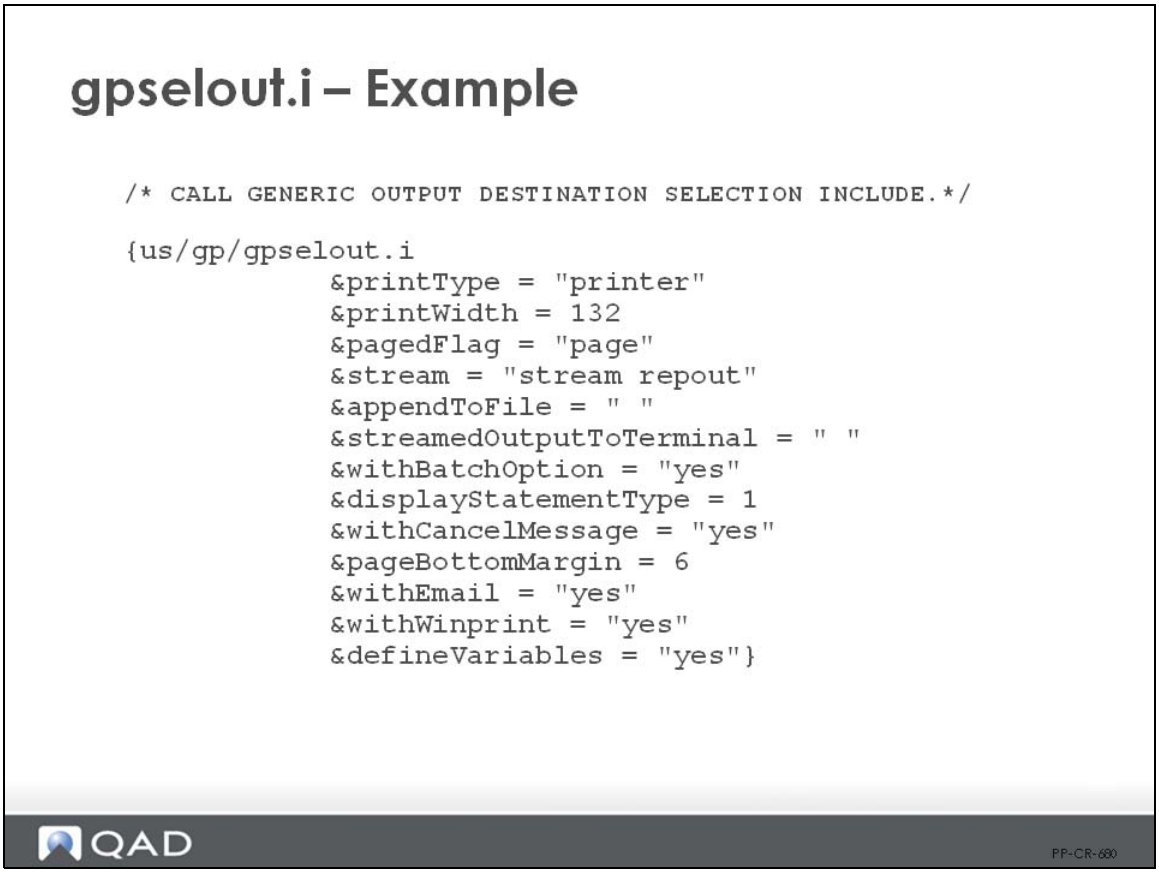
Some of the information defined in the title bar is obtained from Menu System Maintenance. This table manages the program label displayed on the screen and the menu number position.

gpselout.i – Printer Selection

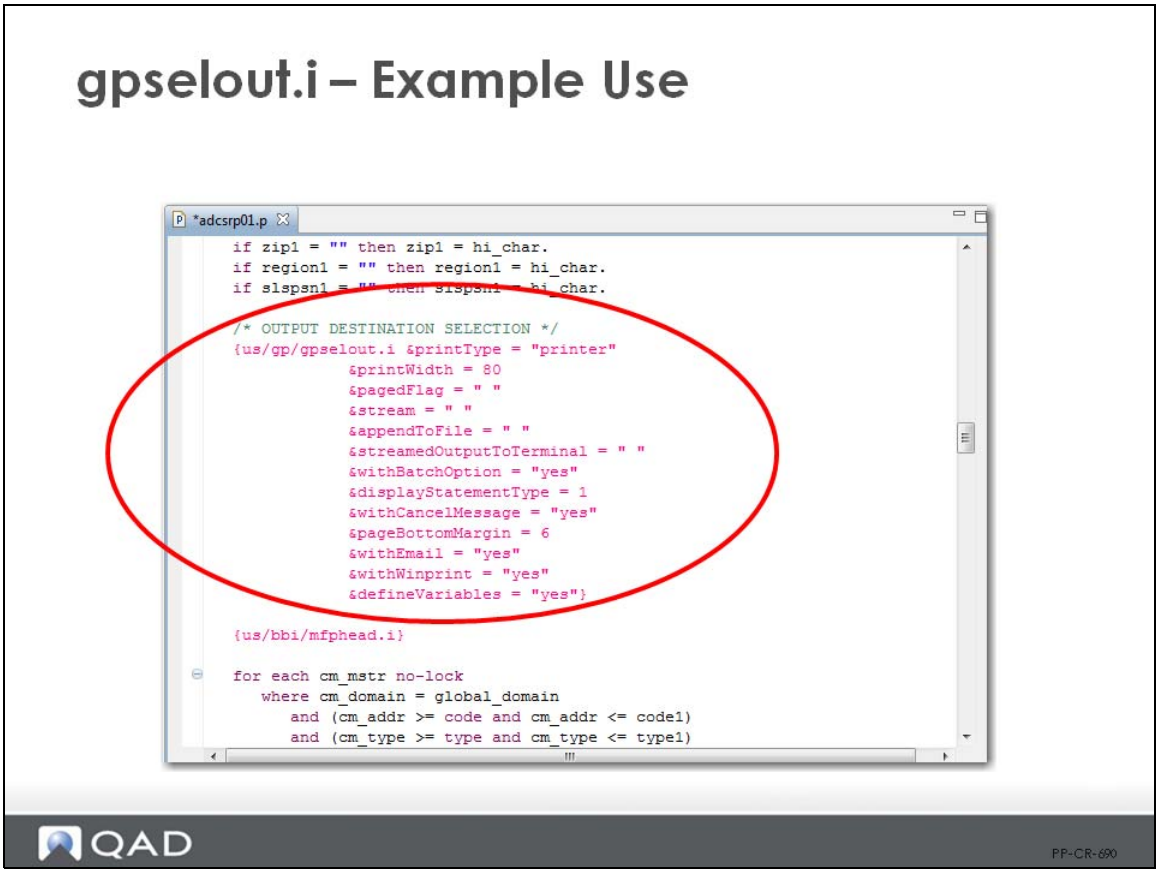


Use `gpselout.i` to prompt the user for an output device and batch ID for running reports in the background.

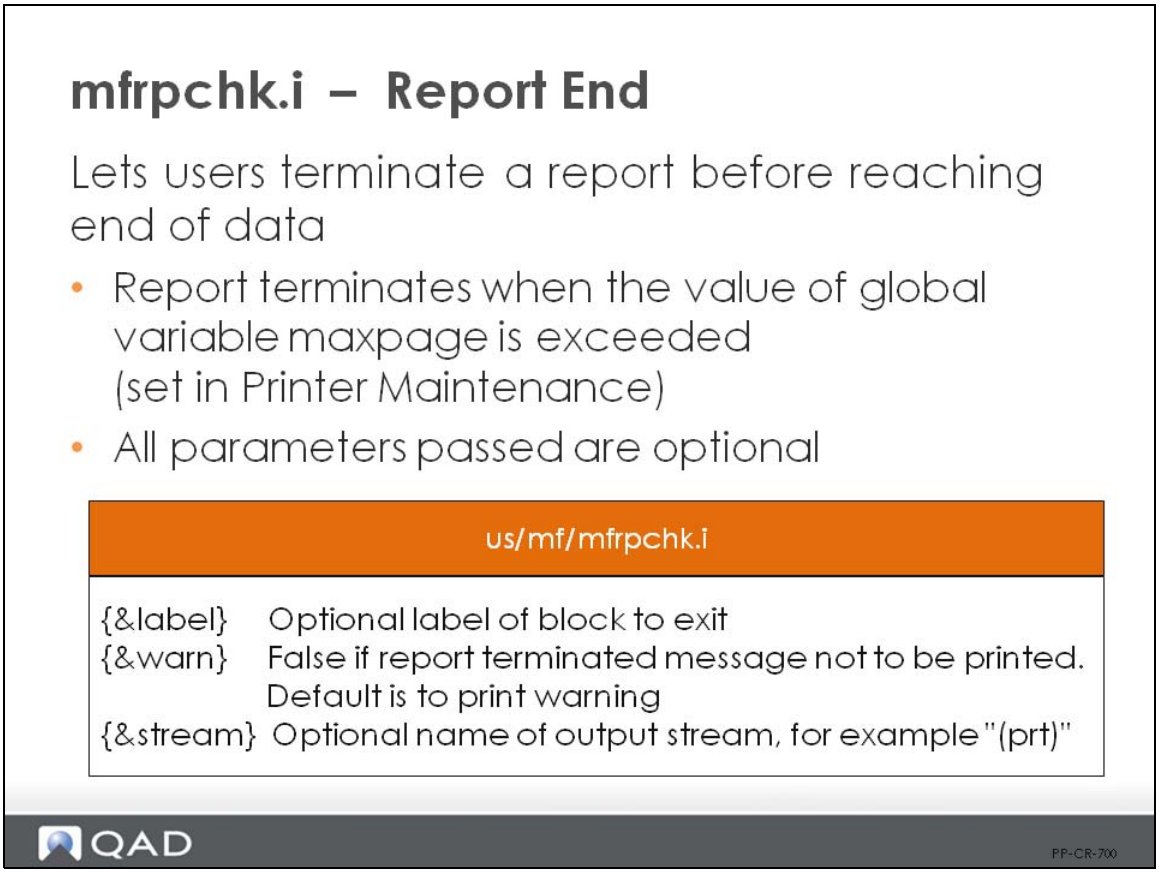
gpselout.i – Example



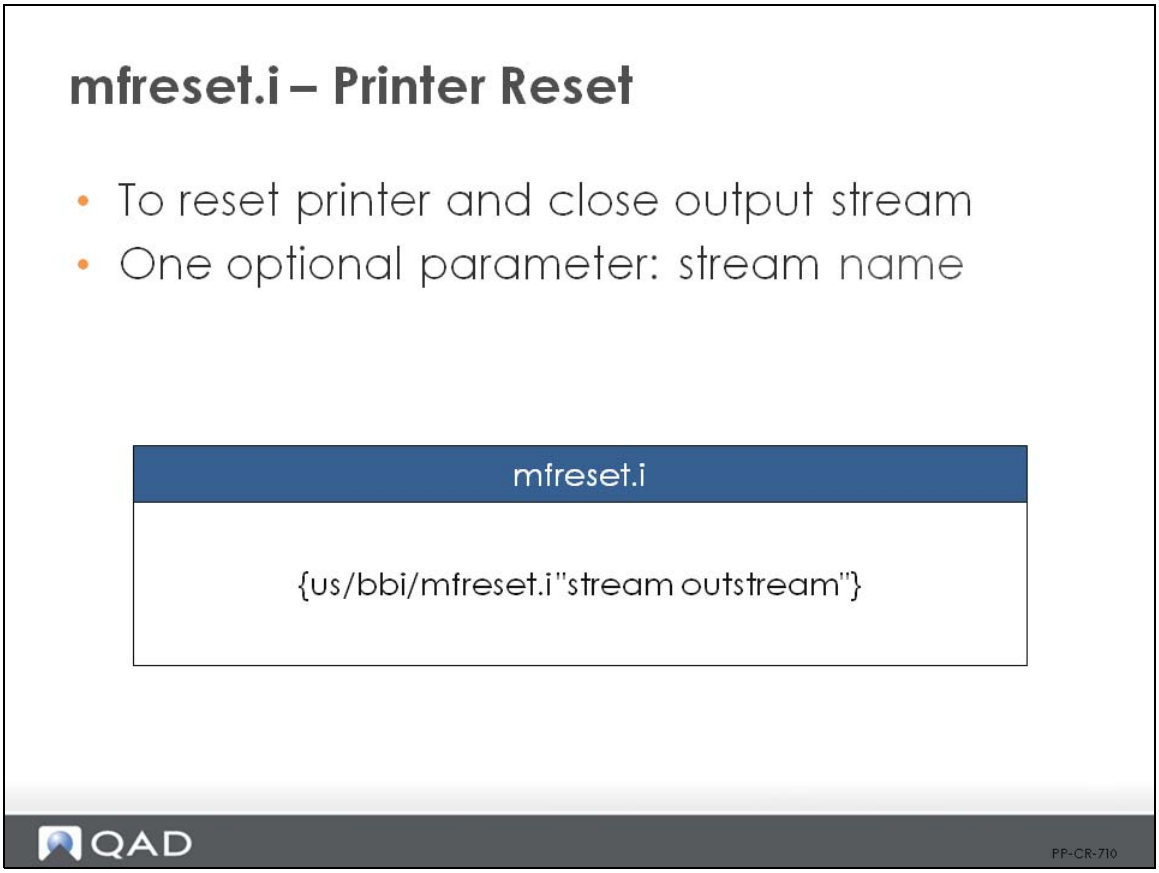
gpselout.i – Example Use



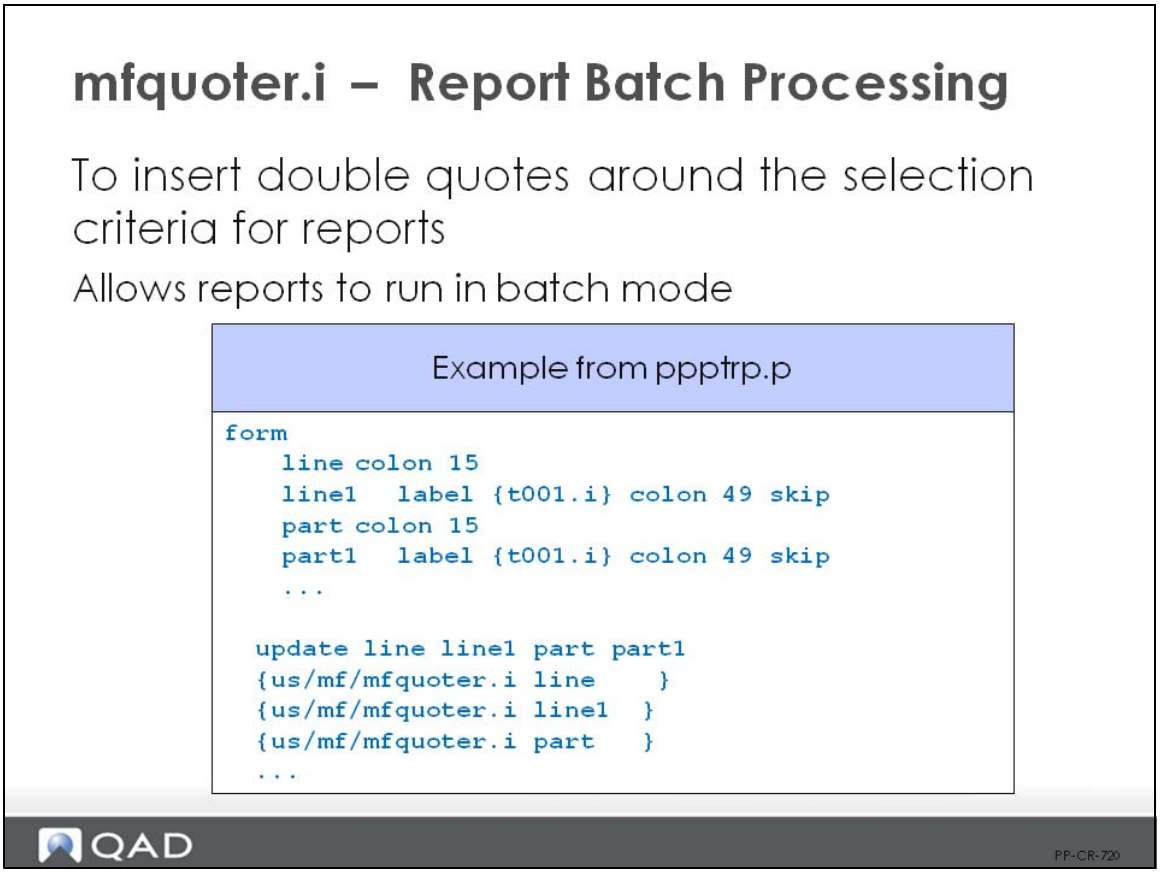
mfrpchk.i – Report End



mfreset.i – Printer Reset



mfquoter.i – Report Batch Processing



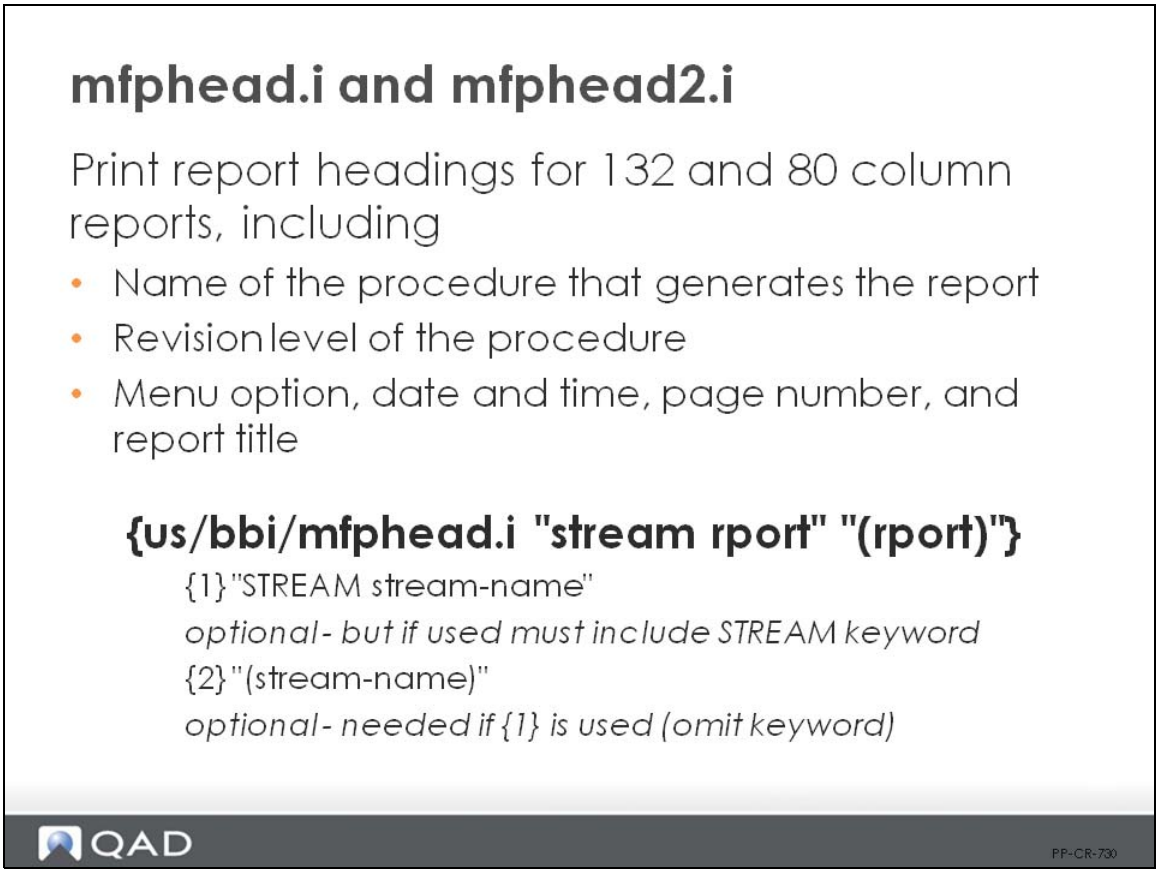
The purpose of the include file `mfquoter.i` is to insert double quotes around the selection criteria for reports. By doing so, it allows you to run reports batch mode.

Because `mfquoter.i` has only one parameter, multiple `mfquoter.i` statements are required, one for each selection criteria.

Notice that `mfquoter.i` is included after the statements updating the form's selection criteria.

It must use the same sequence as the variables in the selection criteria.

mfphead.i and mfphead2.i



Report Header

Report Header

Item Data Report - 6/29/2012 3...

QAD

Item Data Report

10USA

Prod Line: 10 Finished Goods

Item Number	Description	UM	Item	Drawing Brk Cat	Group	Type	Status	Added	Dsgn
01010	Medical Ultrasound	EA	D	F-10000-A23	Medical	FINGOOD	ACTIVE	01/15/02	PRODM
01011	Supplies Kit	EA		SUPPLIES	Medical	KIT	ACTIVE	01/30/02	PRODM
01020	Implantable Ultrasound	EA	A	F-20000-J87	Medical	FINGOOD	ACTIVE	05/06/09	PRODM
01030	Consumer Ultrasound	EA	A	F-30000-X22	CPG	FINGOOD	ACTIVE	01/01/10	PRODM
01040	Industrial Ultrasound	EA	CC	F-40000-B62	IND	FINGOOD	ACTIVE	06/22/04	PRODM
01040-001	Industrial Ultrasound TO	EA	CC	F-40000-B62	IND	FINGOOD	ACTIVE	01/06/11	PRODM

Each report printout has a header

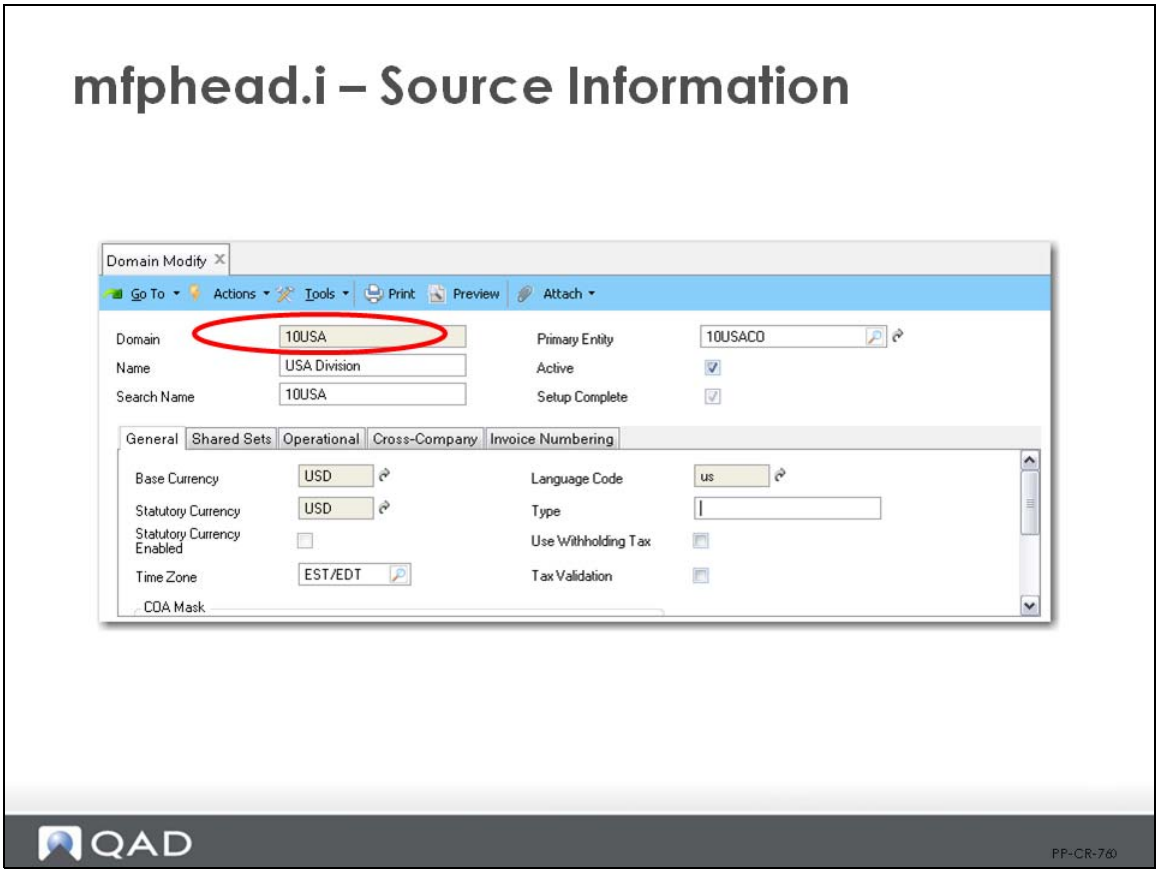
- mfphead.i 132 column report output
- mfphead2.i 80 column report output

QAD

PP-CR-740

mfphead.i – Example Use

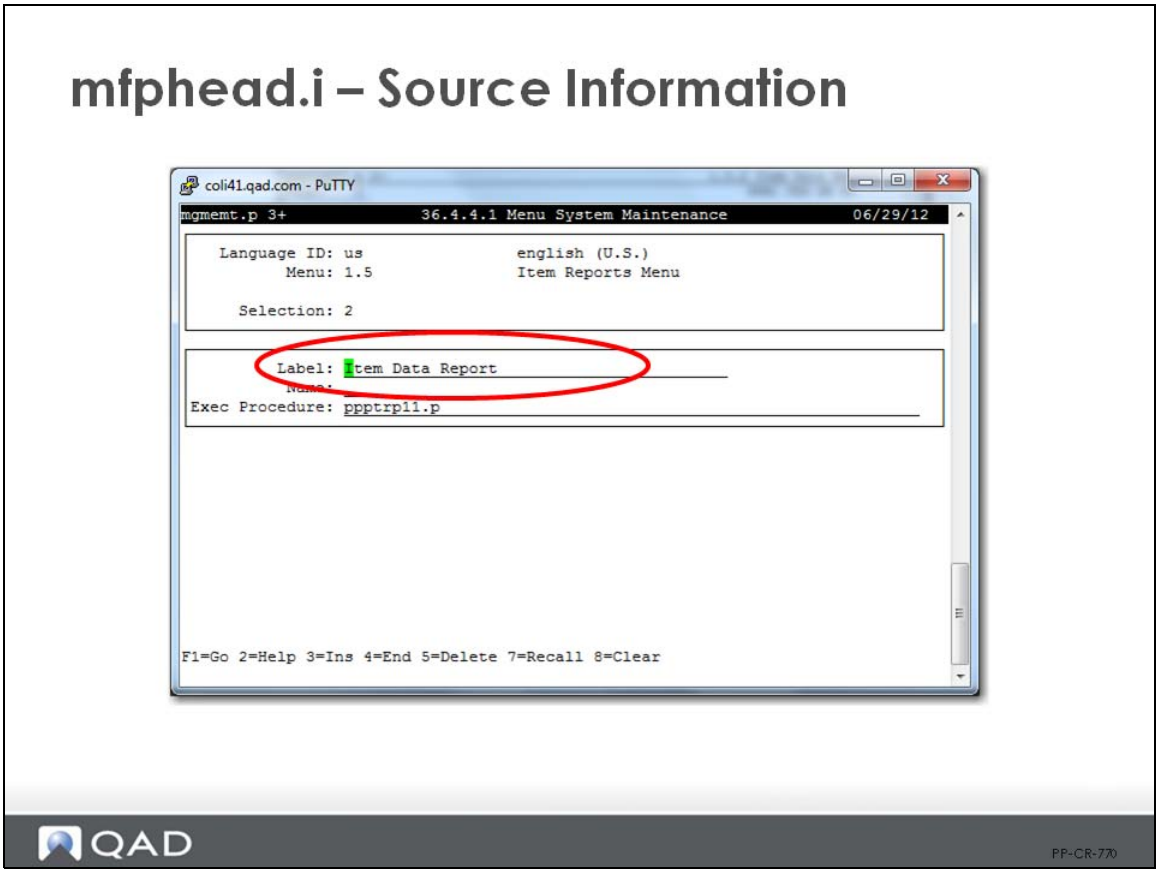
mfphead.i – Source Information



In QAD Enterprise Edition, the company name is the name of the domain defined in Domain Create.

In QAD Standard Edition, the company name that displays on reports is defined with the special setting of ~reports in Company Address Maintenance. This is not applicable to QAD Enterprise Edition.

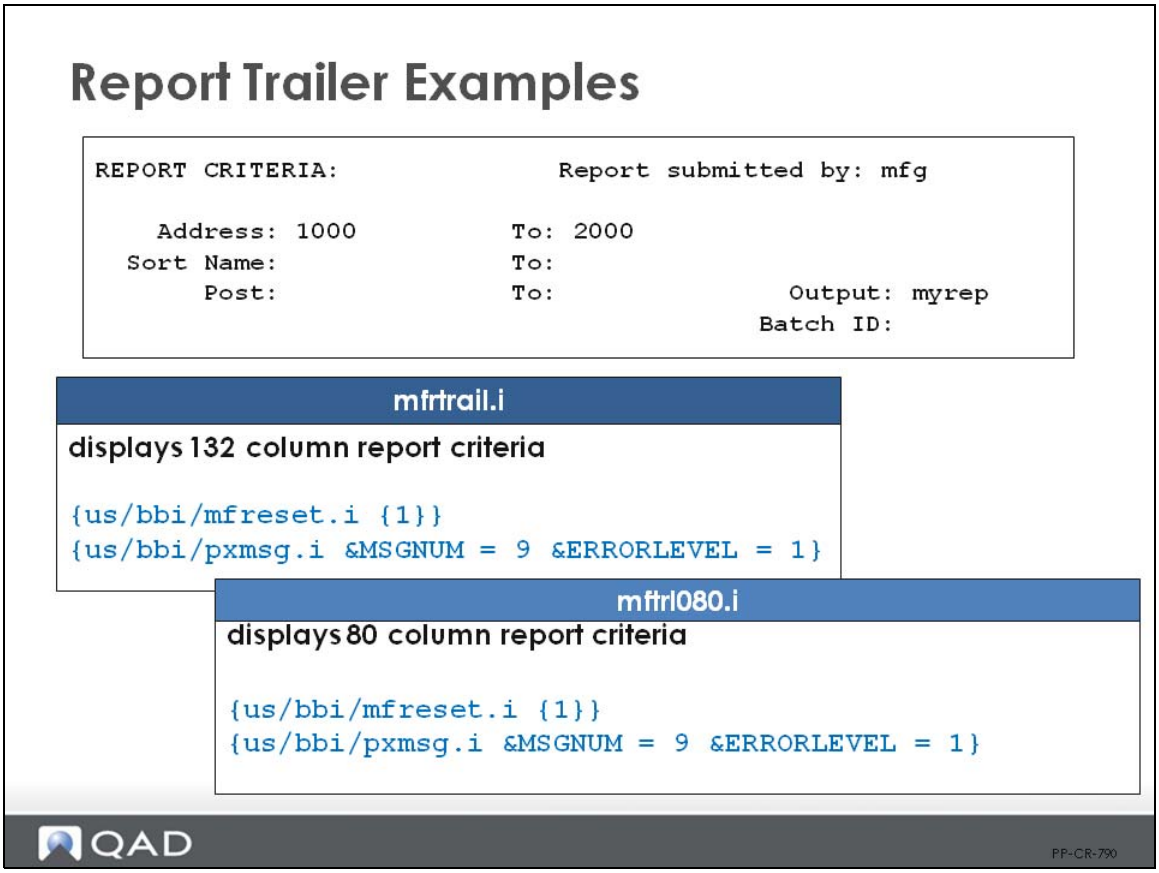
mfphead.i – Source Information



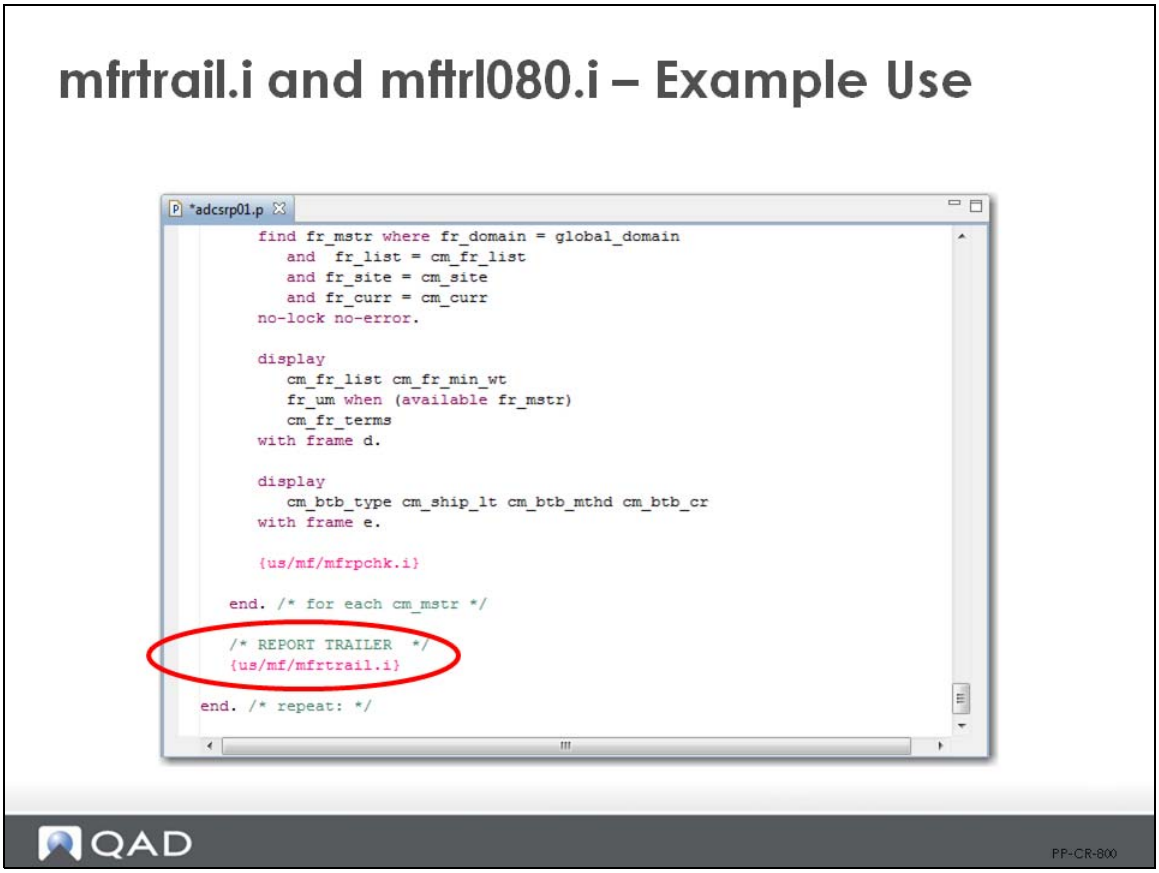
The title of the report is defined in Menu System Maintenance.

mfrtrail.i – Report Trailer

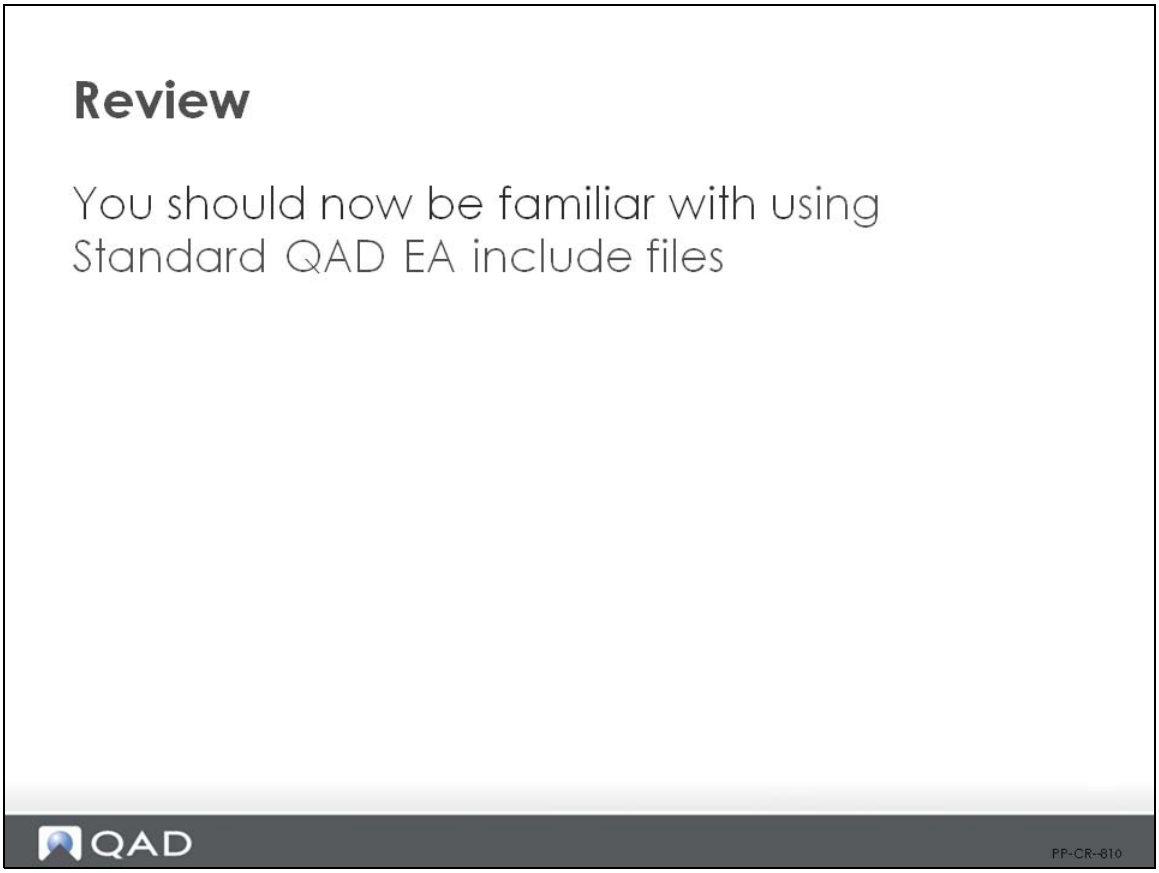
Report Trailer Examples



mfrtrail.i and mftrl080.i – Example Use



Review

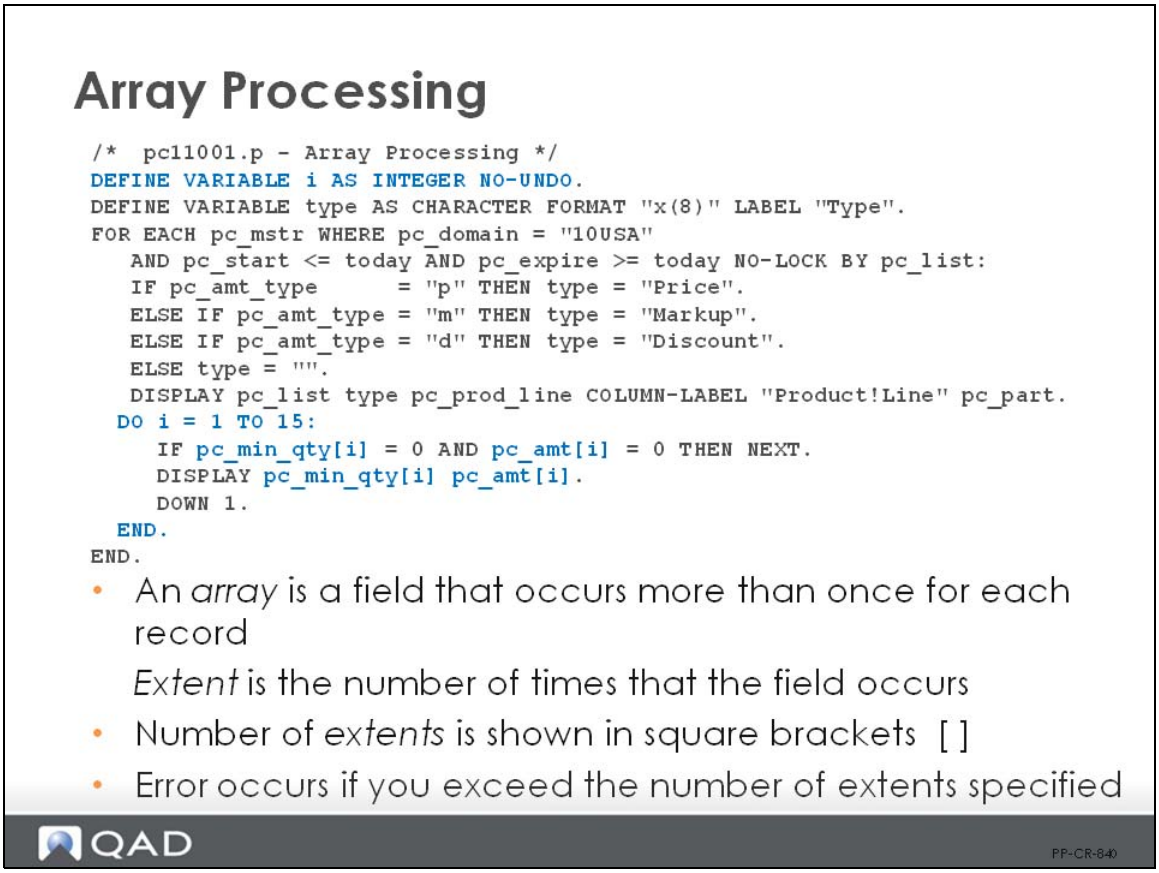

Review
You should now be familiar with using Standard QAD EA include files
QAD
PP-CR-810

Lesson 11: Arrays, Temp-Tables, Buffers, and Other Useful Functions

Upon completion of this lesson, you will be familiar with the following topics:

- Arrays, how to identify them, and how to retrieve information from array fields
- Internal procedures and user-defined functions
- Shared variables and parameters
- Temporary tables and buffers
- Some useful Progress functions such as STRING, SUBSTRING, and TRIM
- Creating user-defined functions

Array Processing



An array is a field that occurs more than once for each record in a table. The Price List table (pc_mstr) contains price information stored in an array. Each page of comments is also stored in an array. In the *Database Definitions* manual, array fields have extents shown after the field name. The number of extents a field has is shown in brackets ([ ]). The extent is the number of times that a field occurs for each record in that table.

You can use the DO statement to repeat a block of commands within a procedure. This is especially useful for array handling.

The example shown here displays price list information from the pc_mstr table. The data dictionary field definition for pc_min_qty and pc_amt fields specifies 15 extents. This indicates that 15 values for each of those fields are available for each price list. The DO statement in the procedure `pc11001.p` specifies that the block iterates 15 times, once for each value of the variable i starting at 1 and going to 15.

To access a unique array item, you must specify the occurrence you want to access. This is done by putting the occurrence number (or extent number) in square brackets [ ] following the field name. For example, pc_amt[4] represents the fourth price list amount.

You can use a variable within brackets after an array field name. The current value of the variable is used as the array extent. The variable must be defined as an INTEGER data type and must be set to a value that is within the bounds of the array. For example, in the procedure `pc11001.p`, trying to access pc_amt[i] with the value of i set to 16 would be invalid because there are only 15 values, not 16.

Exercise 11-1: Arrays

Note The Price Master table is not included in the training database, so you can only review the code in `pc11001.p` to see how arrays are used; you cannot compile and run it.

1. Compare the output from procedure `pc11001.p` with the standard procedure for Price List Maintenance.

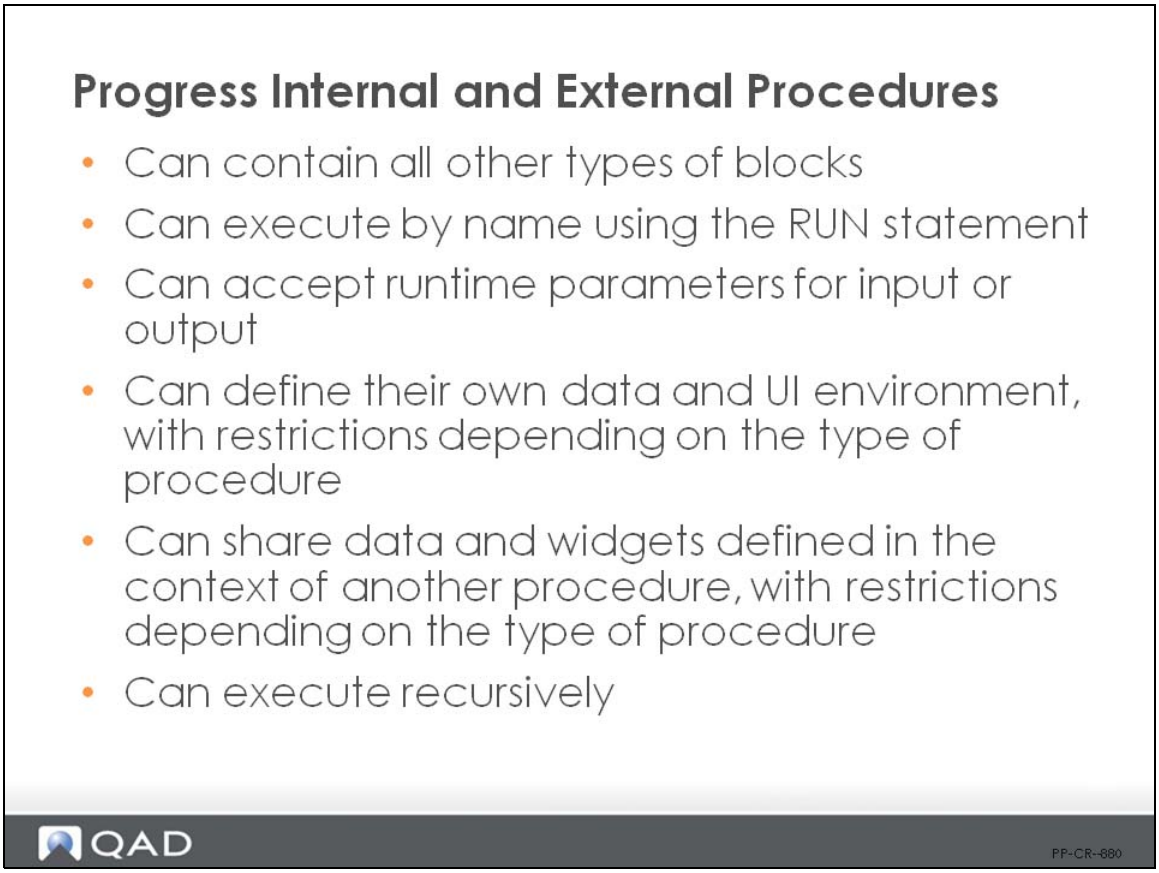
   Hint. Go into QAD EA and run menu option 1.10.2.1 (`ppvnpcmt.p`). Log in as user demo, password qad.

2. Modify the procedure `pc11001.p`. Use the comment delimiters (/*) and (*/) to remove the statement:

   ```
   IF pc_min_qty[i] = 0 AND pc_amt[i] = 0 THEN NEXT.
   ```

3. Use comment delimiters to delete "DOWN 1" and replace DO with REPEAT.

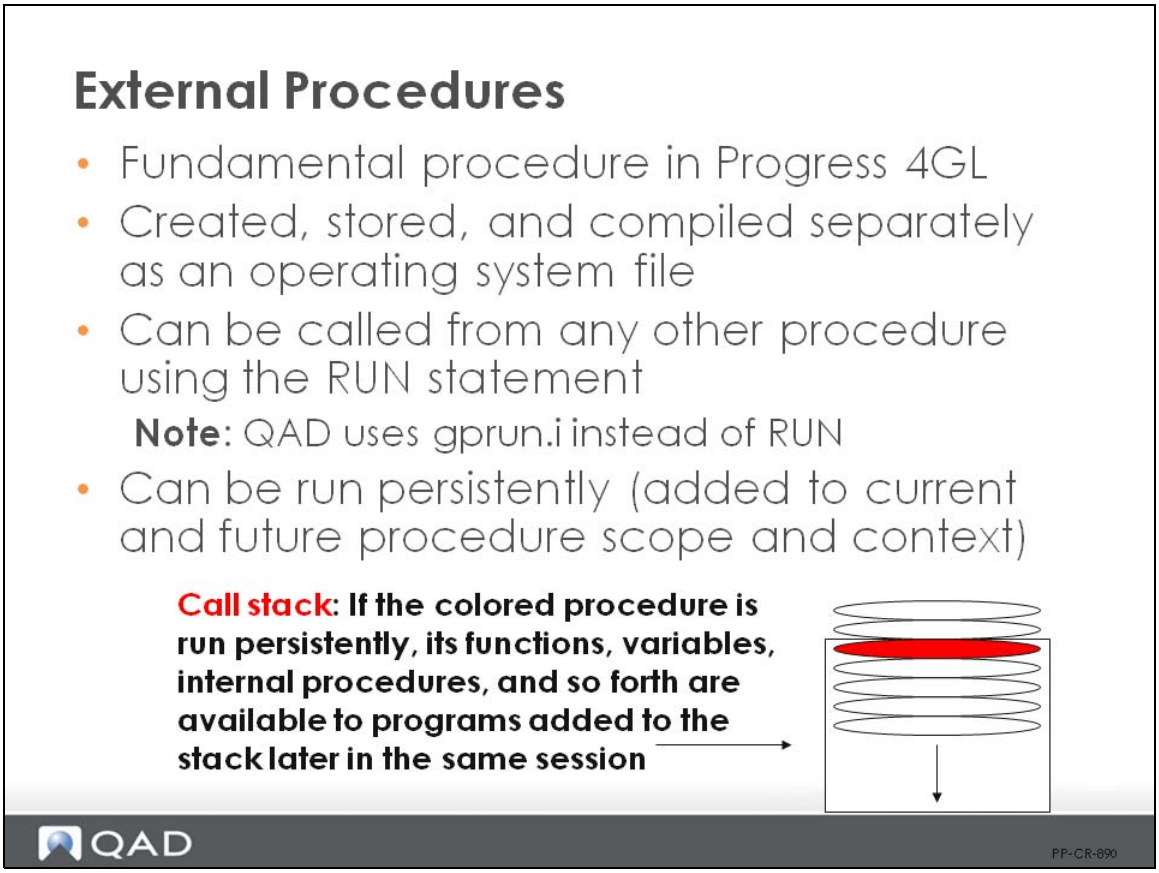
Progress Internal and External Procedures



A separate source procedure is known as an external procedure.

You can define one or more named entry points within an external procedure file, and these are called internal procedures. You run internal procedures in almost exactly the same way that you run external procedures, except that there is no .p file name extension on the internal procedure.

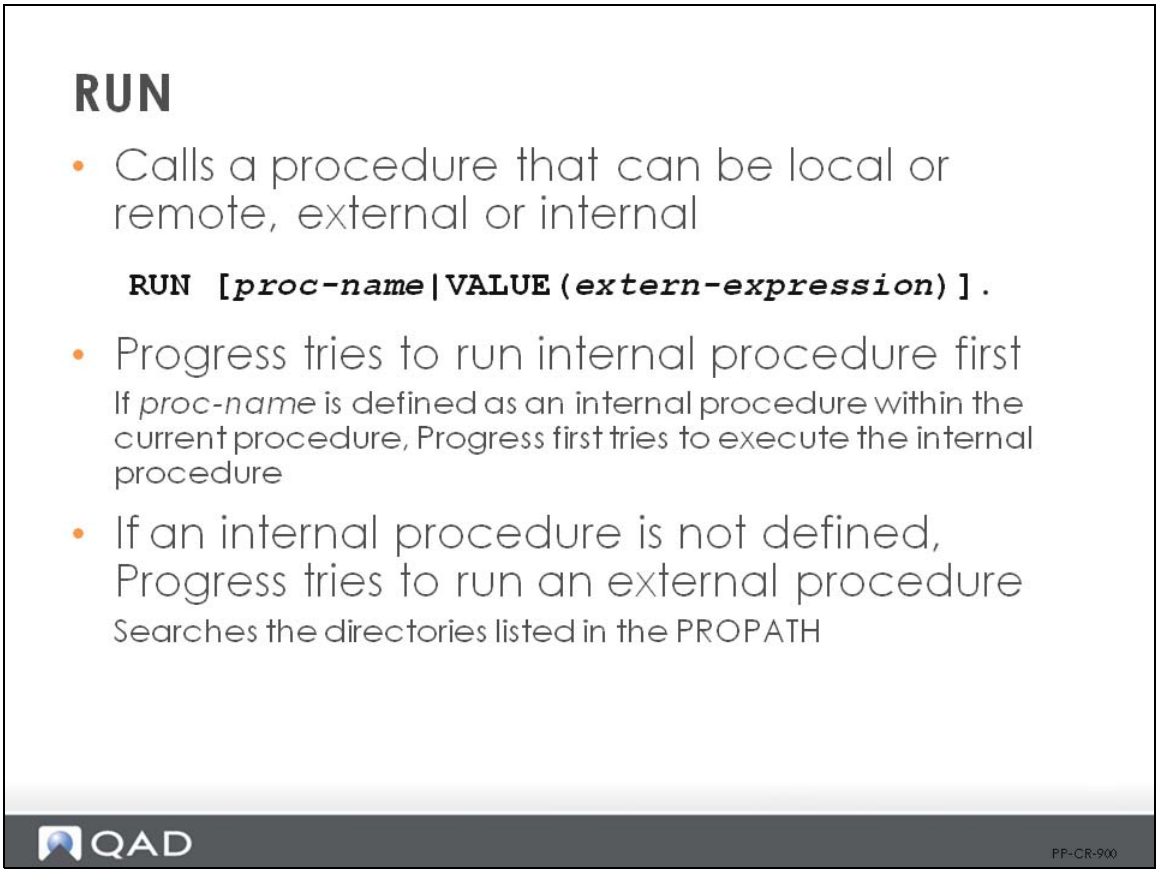
External Procedures



Scope versus context:

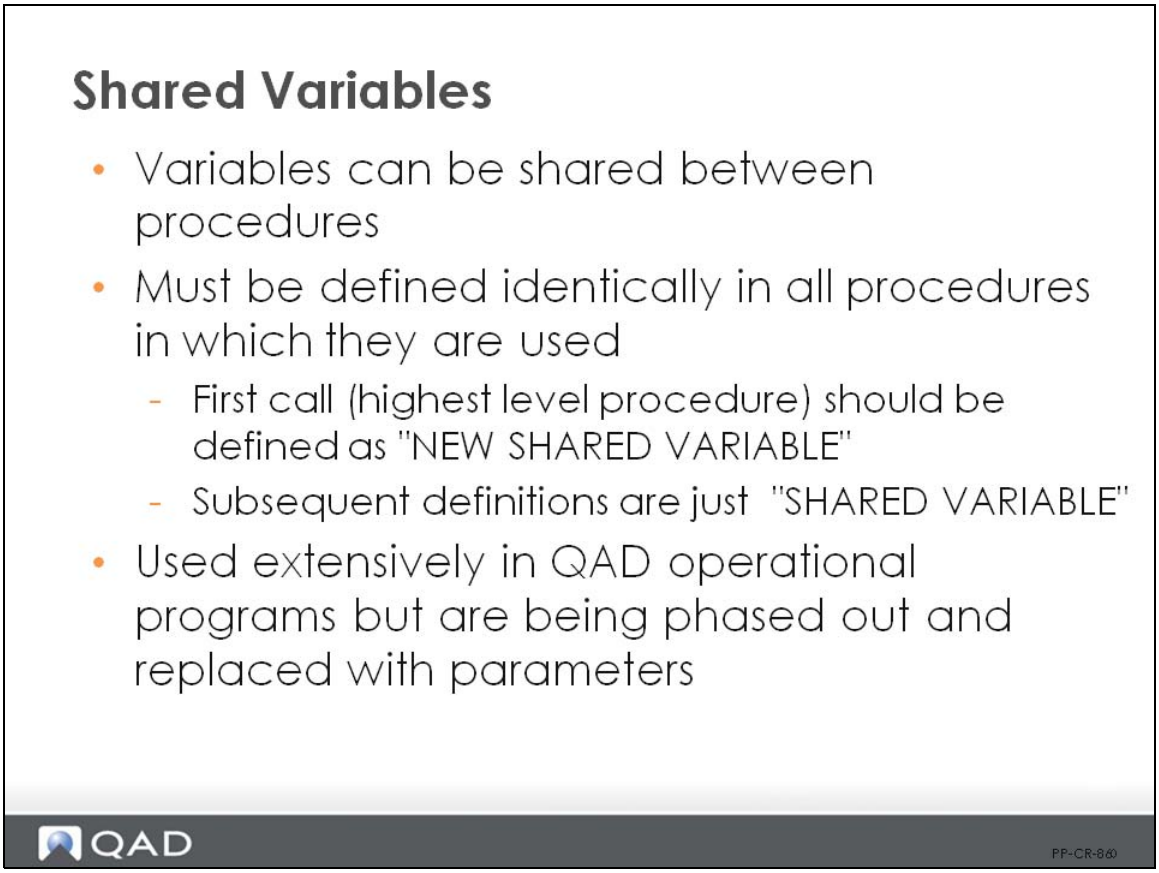
- Context is the application environment in which the procedure executes.
- Scope is the duration that a specific context is available.

RUN



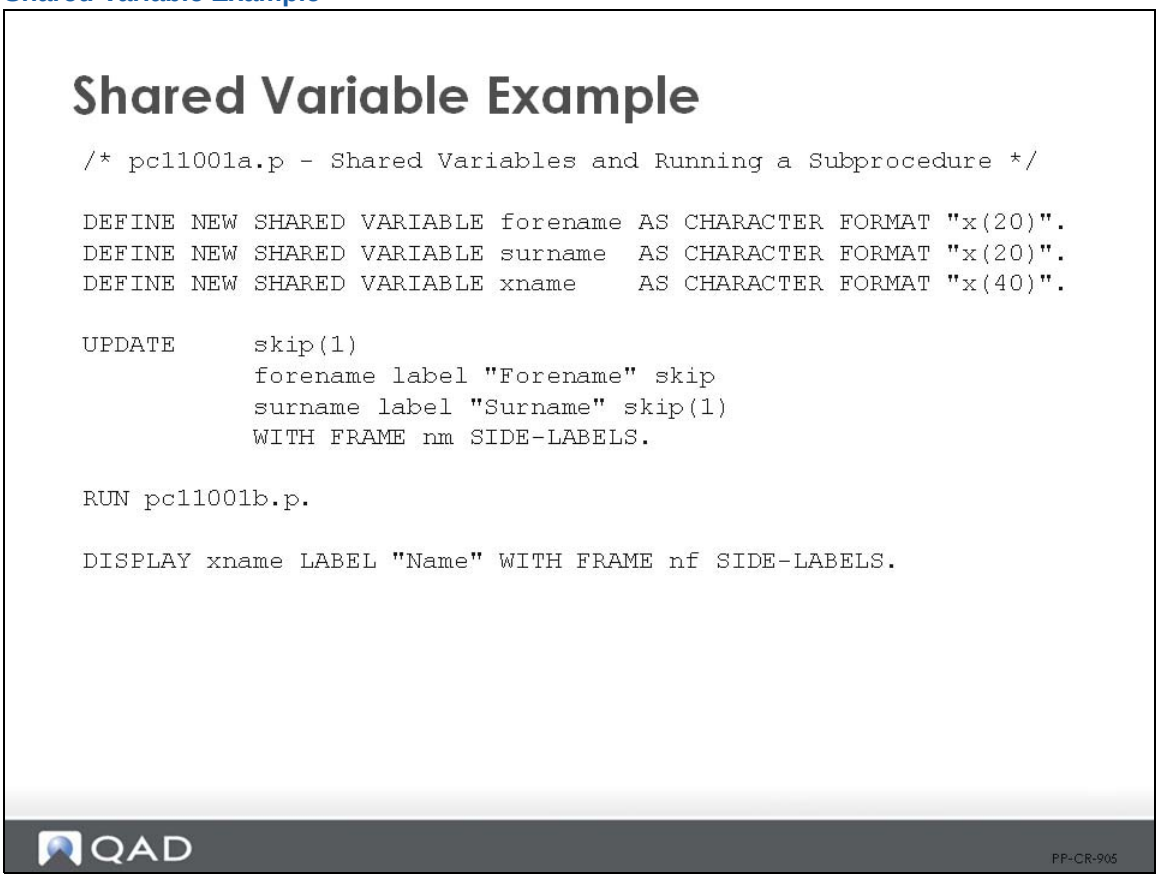
Note QAD uses the include file `gprun.i` instead of the Progress RUN statement when calling external procedures.

Shared Variables



Variables can be shared over several procedures. A shared variable must be defined identically in all procedures that use it. The initial call of a shared variable should be defined as a NEW SHARED VARIABLE. Subsequent definitions should be as a SHARED VARIABLE.

Shared Variable Example



Shared variables are used when values need to be passed between procedures. The example shows one way you can use Progress procedures to call other procedures. This particular method is not actually used in QAD application code where `gprun.i` is used to call other programs.

Note While instances of shared variables and temp-tables exist in QAD applications, the current coding standard is to avoid the use of both shared variables and temp-tables.

Exercise 11-2: Shared Variables

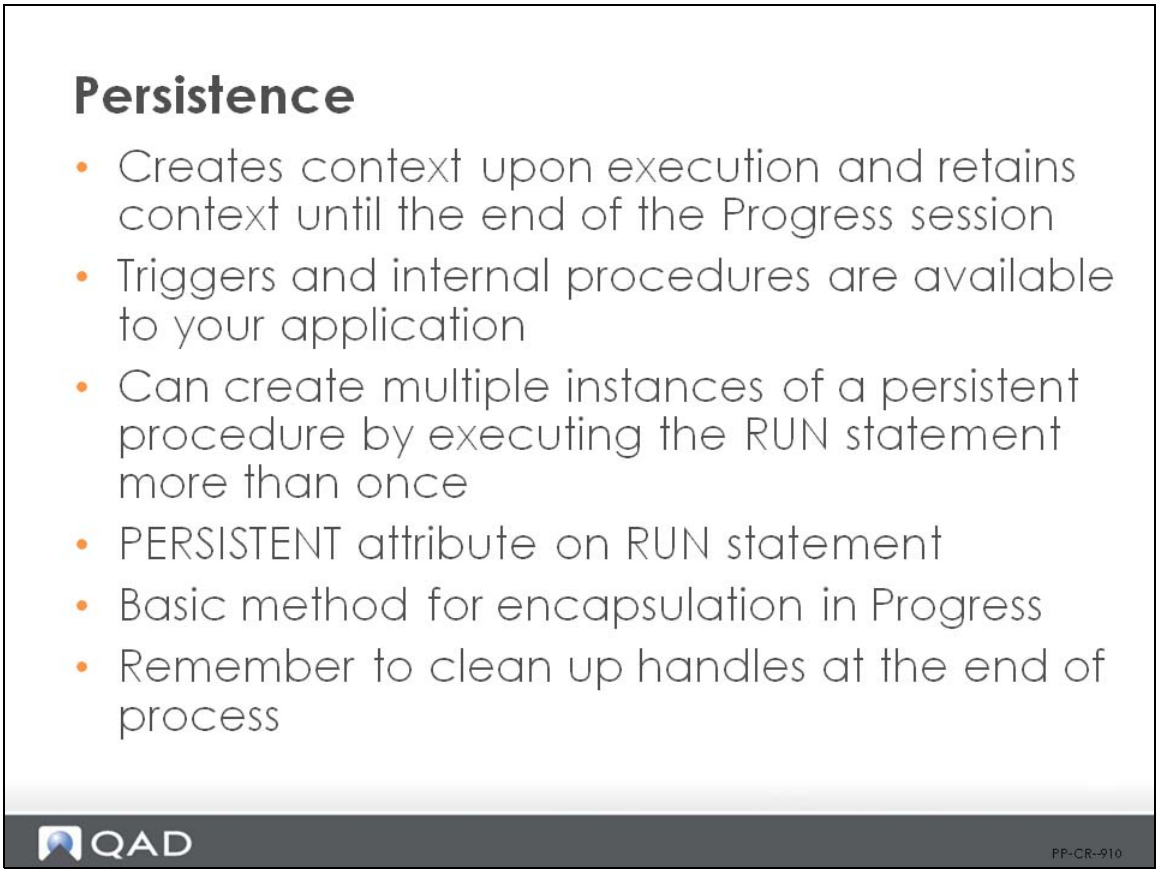
1 Create a new procedure called `main1.p` that:

- Asks the user to UPDATE an integer type variable.
- Then displays the value of the variable.
- Then call subprocedure `upd2.p`.
- After running `upd2.p`, redisplay the value of the variable in a different frame.

2 Create a new program called `upd2.p` that:

- Takes in a numeric variable passed to it.
- Adds the value of ten to the variable.

Persistence



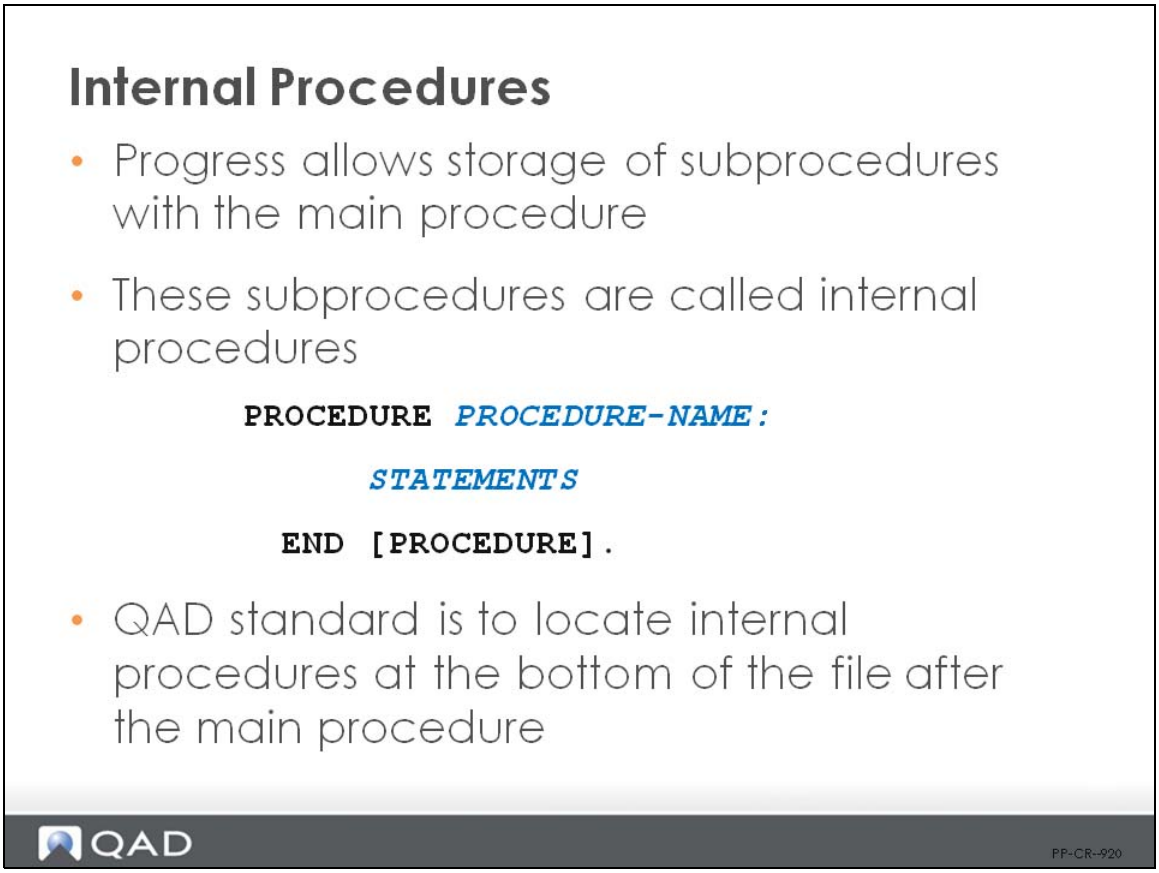
The syntax for running a program persistently is:

```
RUN filename PERSISTENT [SET handle] [run-options]
```

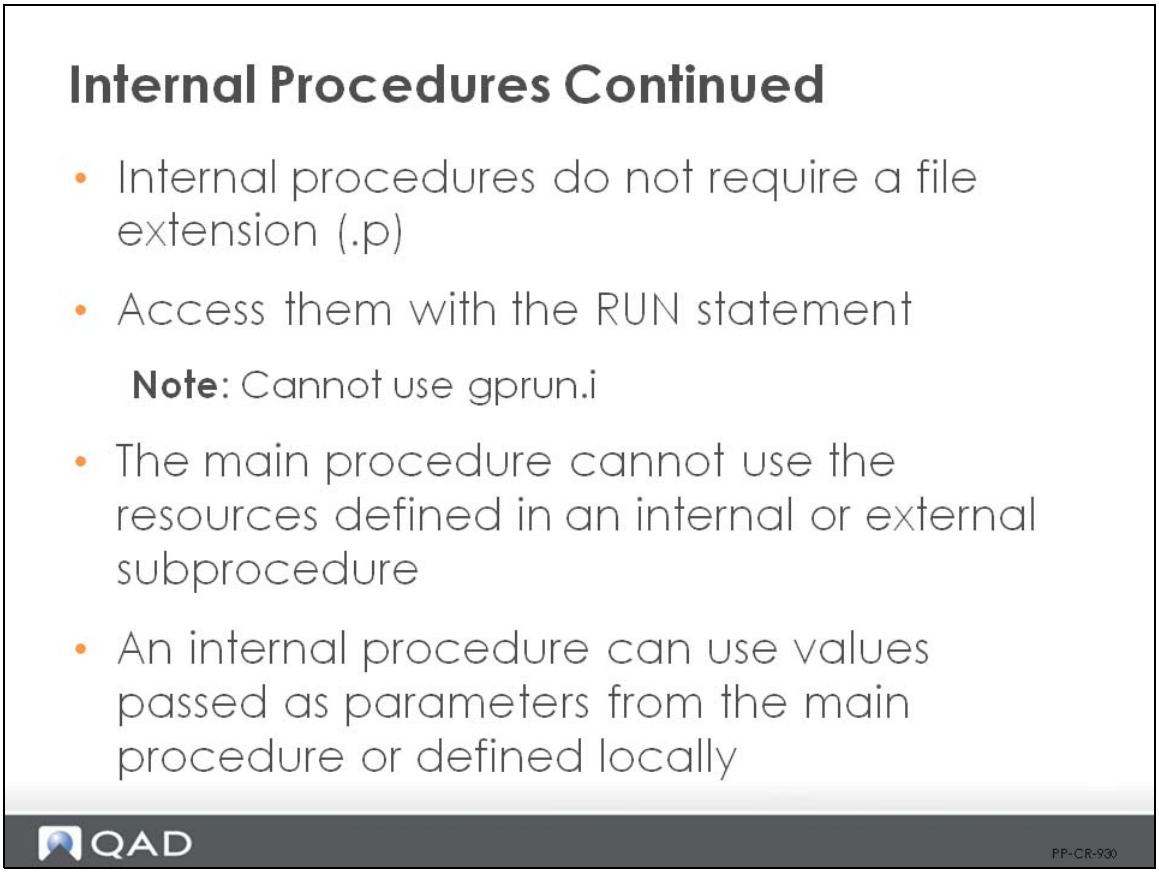
- Filename and run options are the same as for external procedures.
- Handle is a field or variable of HANDLE data type that identifies the current context of the procedure. You can use the handle to invoke triggers or internal procedures in the persistent procedure. See “Handles in Progress” on page 290 for more details.

You pass parameters to a persistent procedure in the same way as with standard external procedures.

Internal Procedures



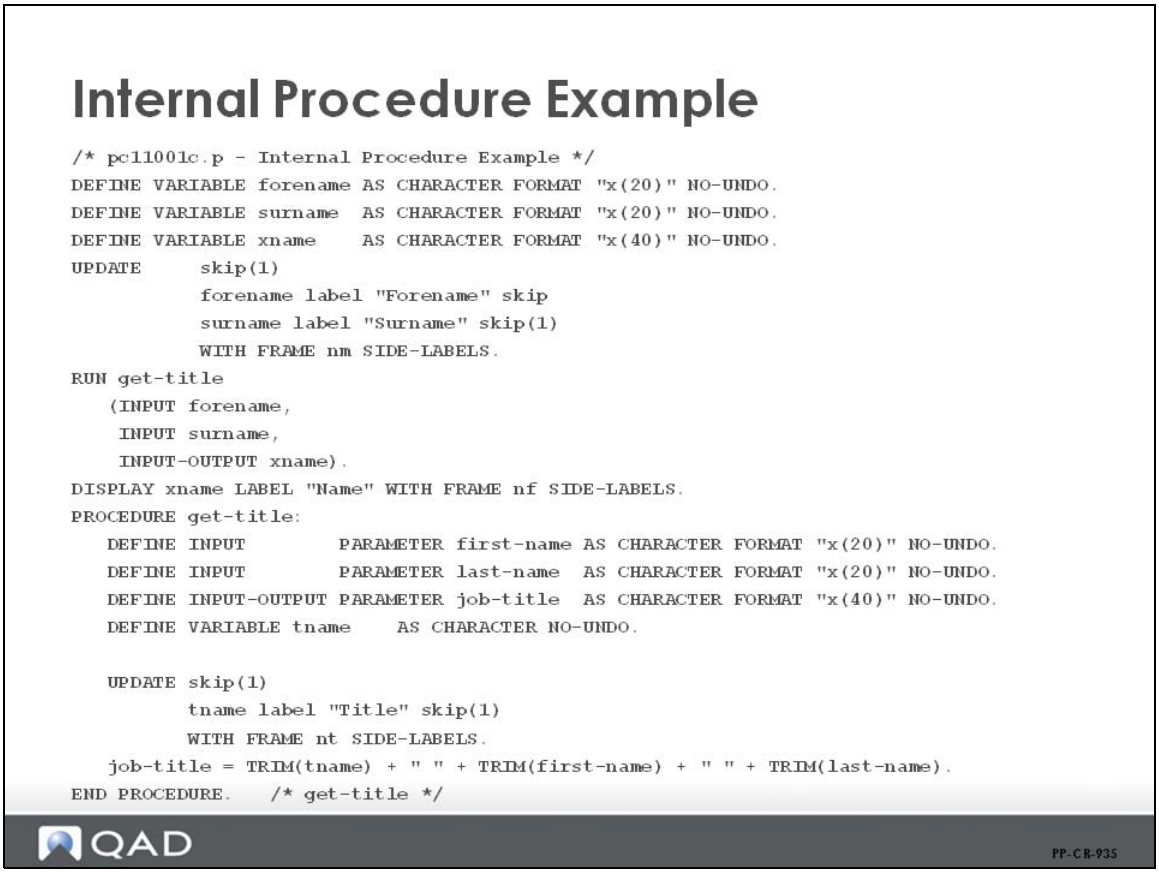
Internal Procedures Continued



Use the Progress RUN statement to execute an internal procedure. The include file `gprun.i` is only used with external procedures.

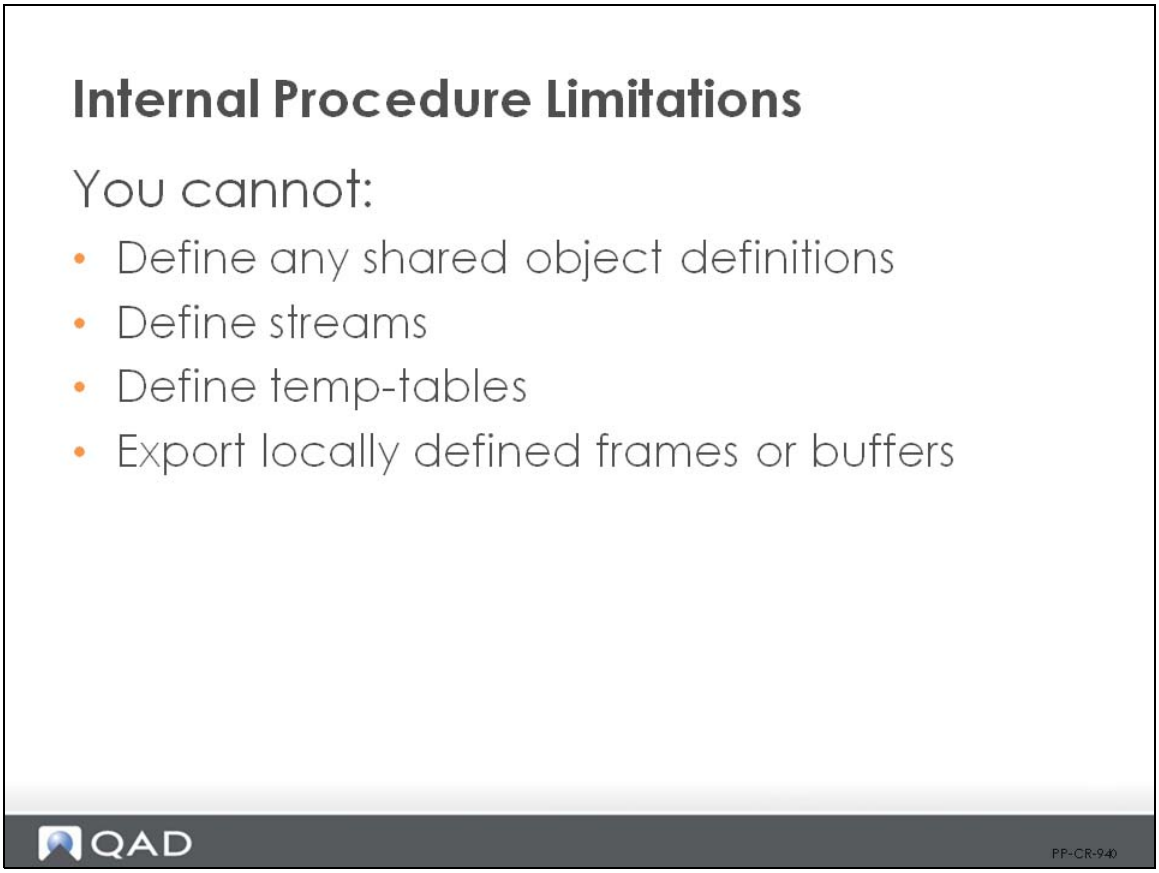
Anything referenced inside an internal procedure must be passed to it through a parameter or defined locally within the procedure. An internal procedure should not directly use resources defined in the main procedure.

Internal Procedures Example

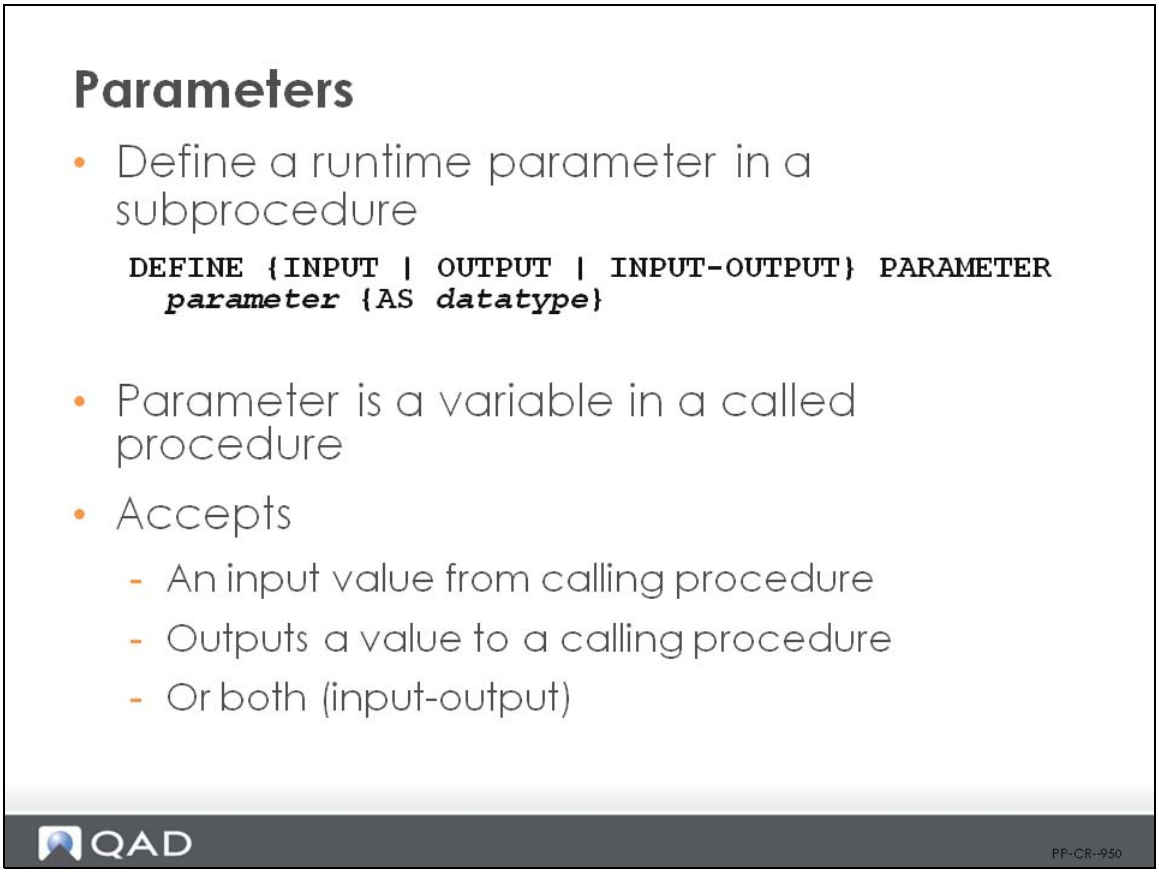


Program `pc11001c.p` shows how you can combine the two previous programs (`pc11001a.p` and `pc1100b.p`) into a single program using an internal procedure. This approach eliminates the use of shared variables using parameters (which are discussed next).

Internal Procedure Limitations

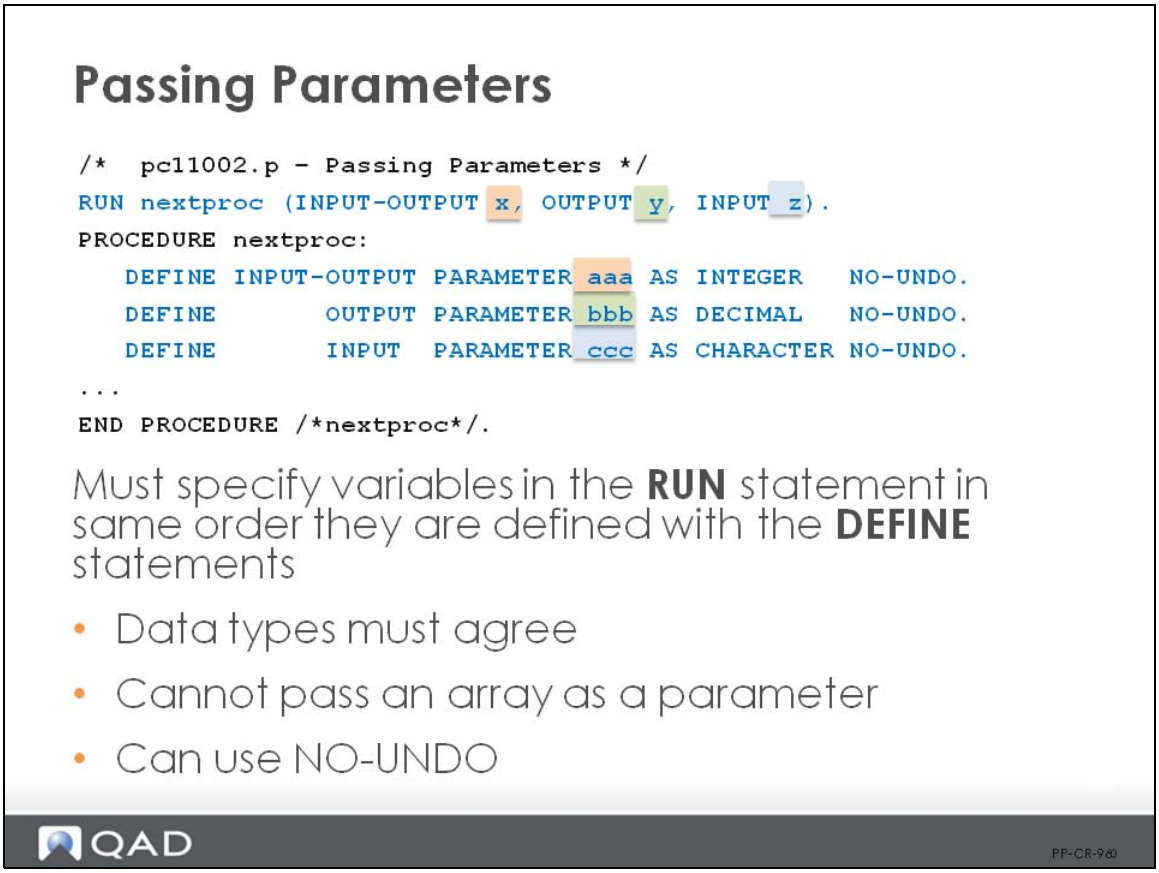


Parameters



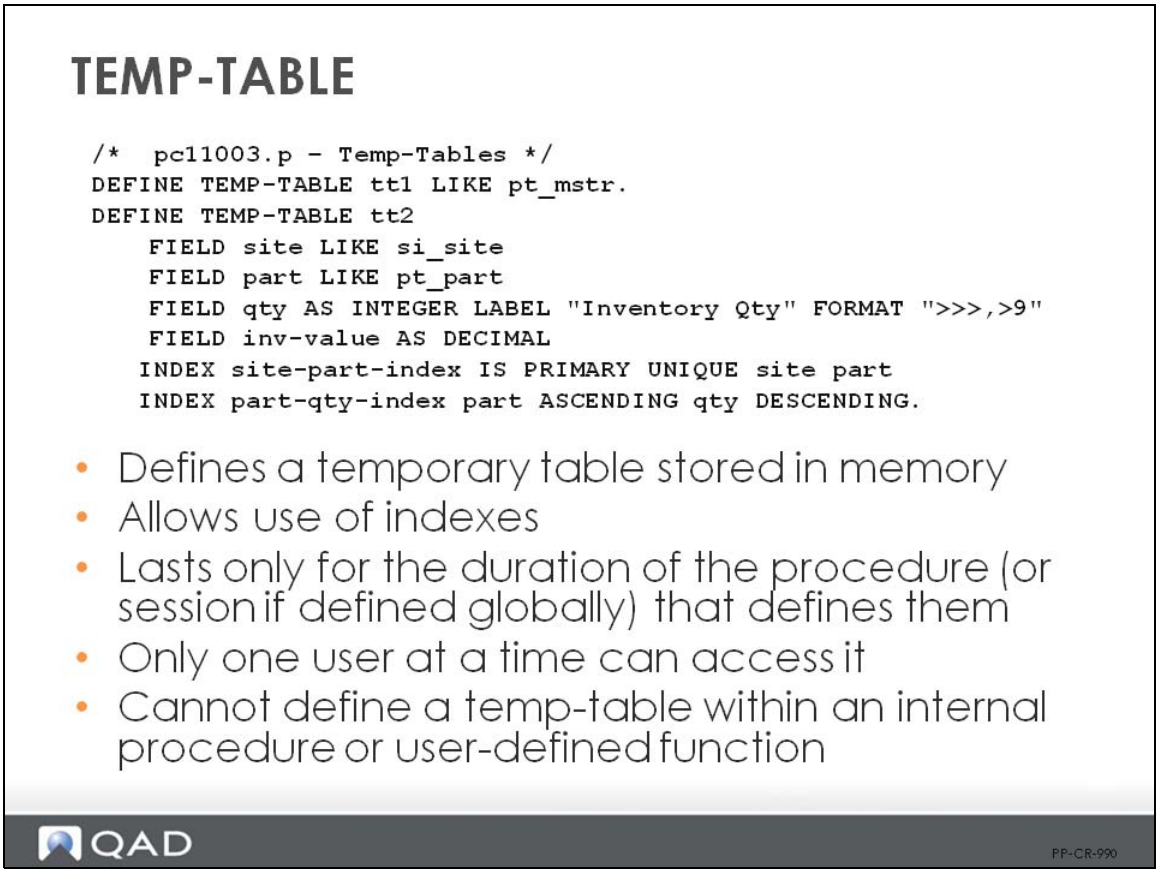
Parameters, not shared variables, are the preferred way to pass data between programs and procedures.

Passing Parameters



- INPUT-OUTPUT passes information to the called procedure and then passes back the updated information from the procedure to the calling procedure.
- OUTPUT requests output from the called procedure.
- INPUT provides input to the called procedure.

TEMP-TABLE



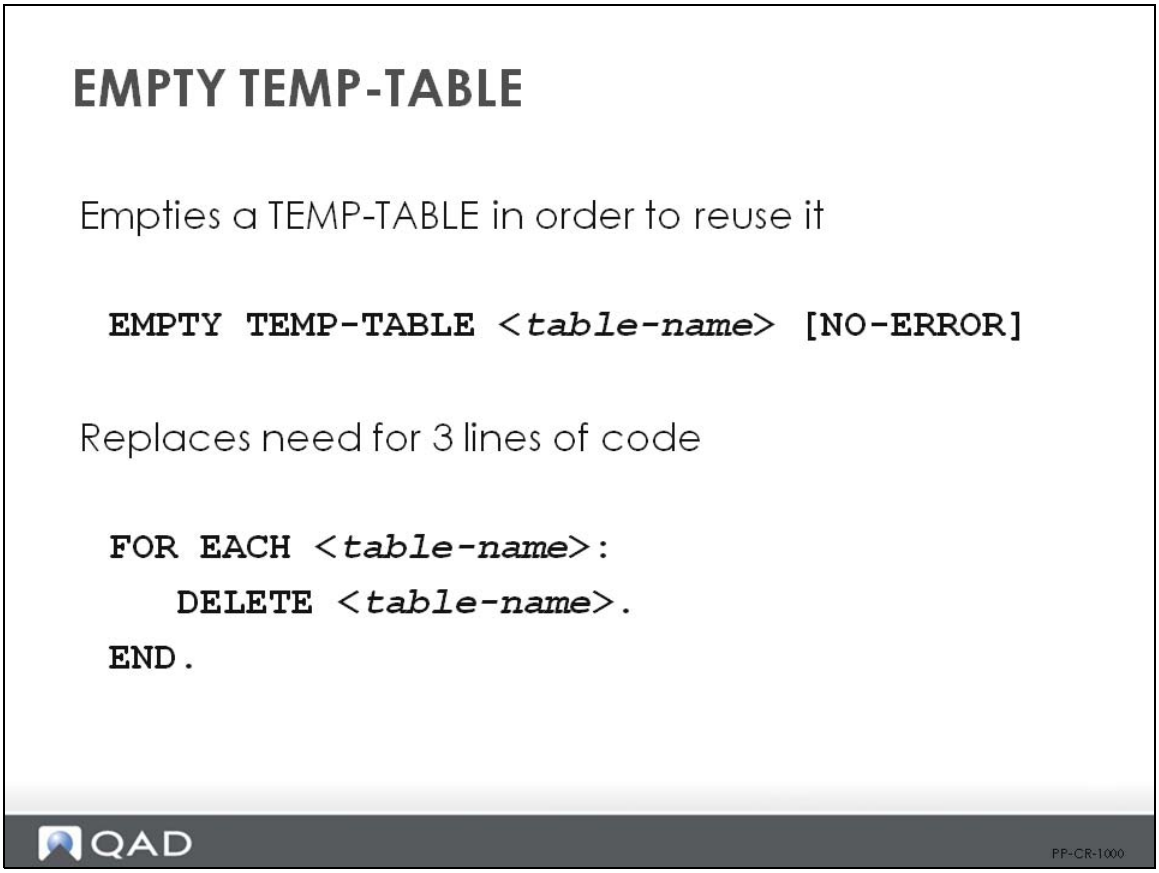
Progress allows for the creation of temporary database tables that can be used to hold work file and summary information. Temporary tables are stored on disk in a temporary database and are available for the duration of the procedure that defines them. They do not affect the QAD EA database in any way.

A temporary table is set up using the DEFINE TEMP-TABLE statement, followed by the temporary table name, which includes a list of fields that the table will contain along with any indexes.

Once defined, a temporary table can be treated like any other table contained in the QAD EA database. One difference is that the NO-LOCK and EXCLUSIVE-LOCK options are not required when accessing records of the temporary table because the table is local to the procedure that created it.

If you need to share a temporary table among procedures, pass it as a parameter.

EMPTY TEMP-TABLE



The EMPTY TEMP-TABLE command is a newer, more efficient command that combines three lines of code. This command only empties outside of transaction scope, whether the transaction includes TEMP-TABLE records or not.

It can be invoked as a statement:

```
EMPTY TEMP-TABLE <table-name> [NO-ERROR]
```

Or as a method (not used in operational code):

```
<tt-buffer-hand>:EMPTY-TEMP-TABLE()
```

Exercise 11-3: TEMP-TABLE

1 Create a procedure that uses a temporary table to accumulate Total Customer Balance by State:

- Prompt the user to choose a state.
- Then show the total balance or an error message.

Hint. Use cm_mstr and ad_mstr; fields ad_state and cm_balance. Remember to set the domain to "10USA" in the WHERE clauses.

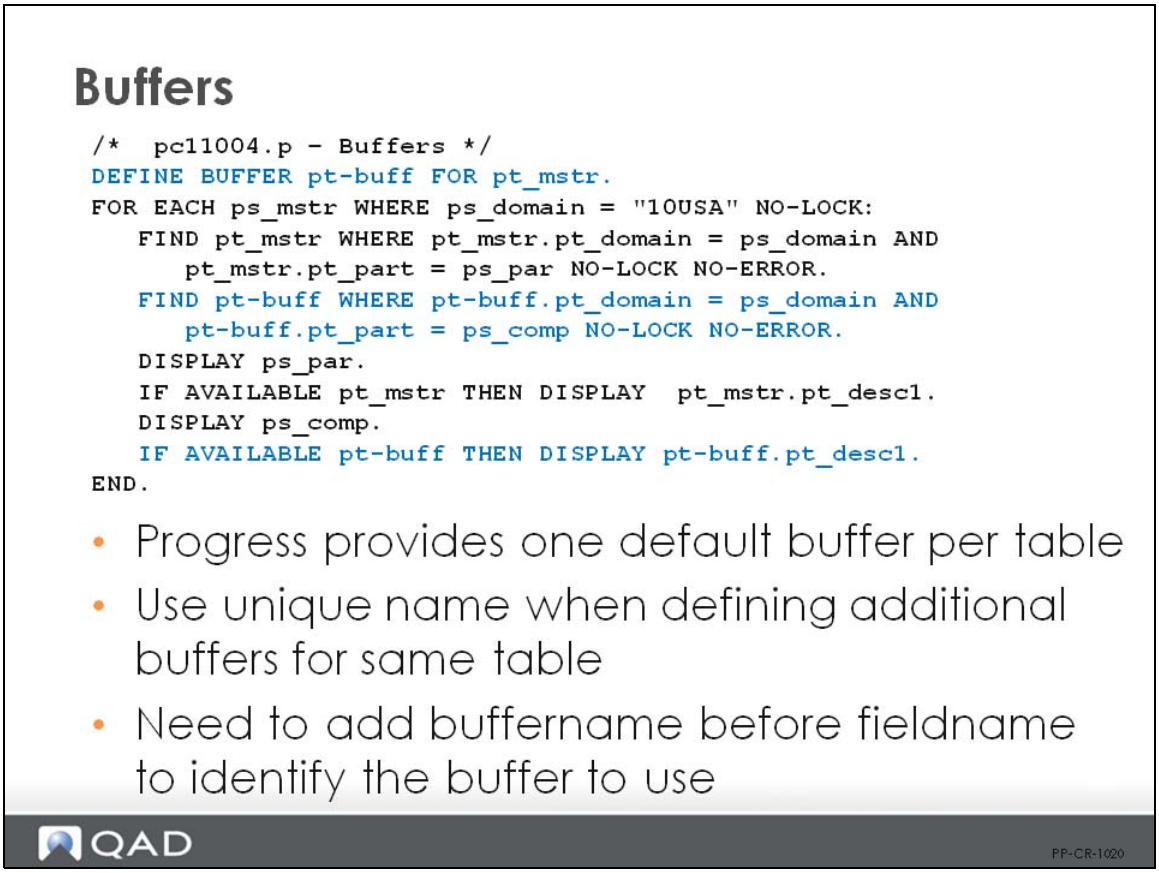
2 Review the following code sample while doing this exercise.

```
DEFINE VARIABLE whatstate AS CHARACTER NO-UNDO.
DEFINE TEMP-TABLE tt_summary NO-UNDO
   FIELD tt_state AS CHARACTER
   FIELD tt_totbal AS DECIMAL
INDEX index1 AS PRIMARY tt_state.

FOR EACH cm_mstr WHERE cm_domain = "10USA" NO-LOCK,
EACH ad_mstr NO-LOCK WHERE ad_domain = cm_domain AND ad_addr = cm_addr
   FIND tt_summary WHERE tt_state = ad_state NO-ERROR.
      IF NOT AVAILABLE tt_summary THEN DO:
         CREATE tt_summary.
         ASSIGN tt_state = ad_state.
      END.
    tt_totbal = tt_totbal + cm_balance.
END.

REPEAT:
        UPDATE whatstate.
        FIND tt_summary WHERE tt_state = whatstate NO-ERROR.
        IF AVAILABLE  tt_summary THEN DISPLAY tt_totbal WITH 1 DOWN.
        ELSE DISPLAY "Error" @ tt_totbal.
END.
```

Buffers

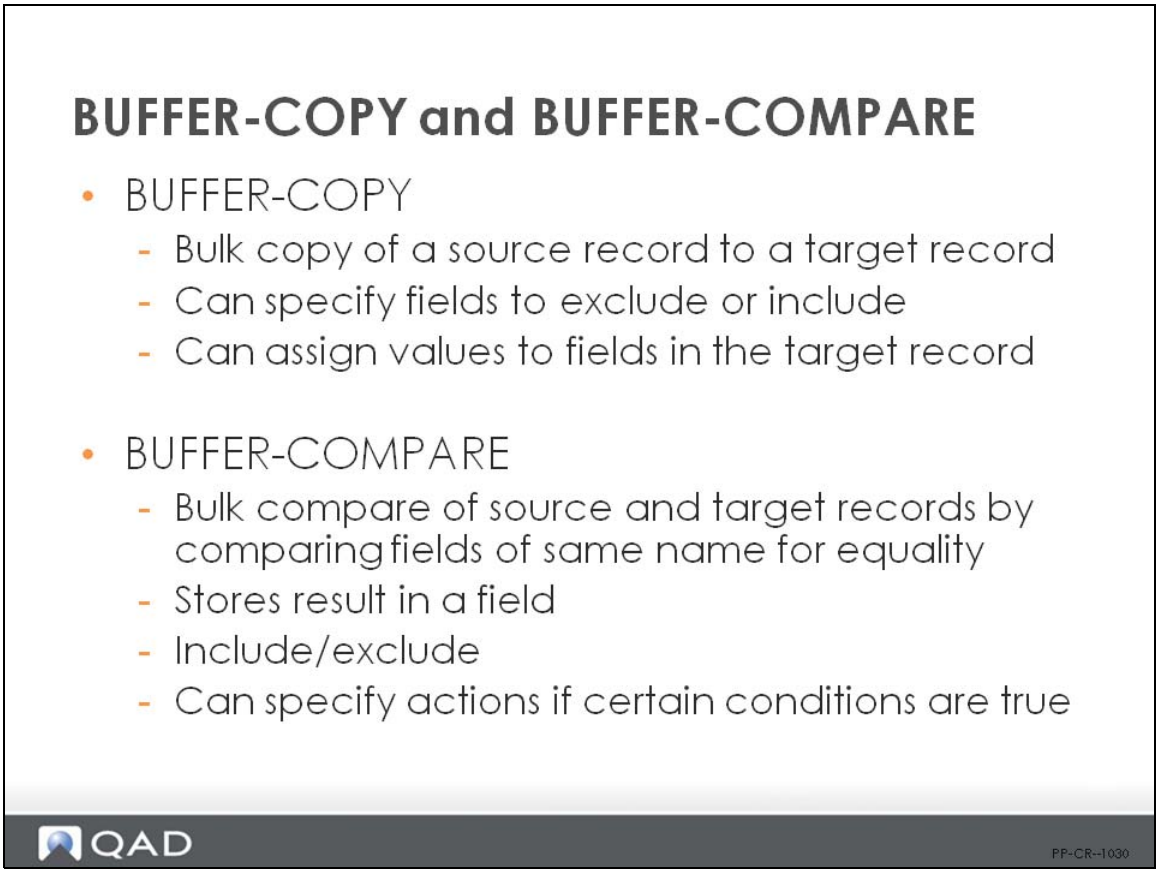


Progress provides you with one buffer for each table that you use in a procedure. This buffer is referenced by its table name in the database (for example, pt_mstr). Progress uses that buffer to store one record at a time from the table as the records are needed during the procedure.

If you need more than one record at a time from a table, you can define additional buffers for the table using the DEFINE BUFFER statement followed by the buffer name.

If you need to share buffers among procedures, pass them as parameters.

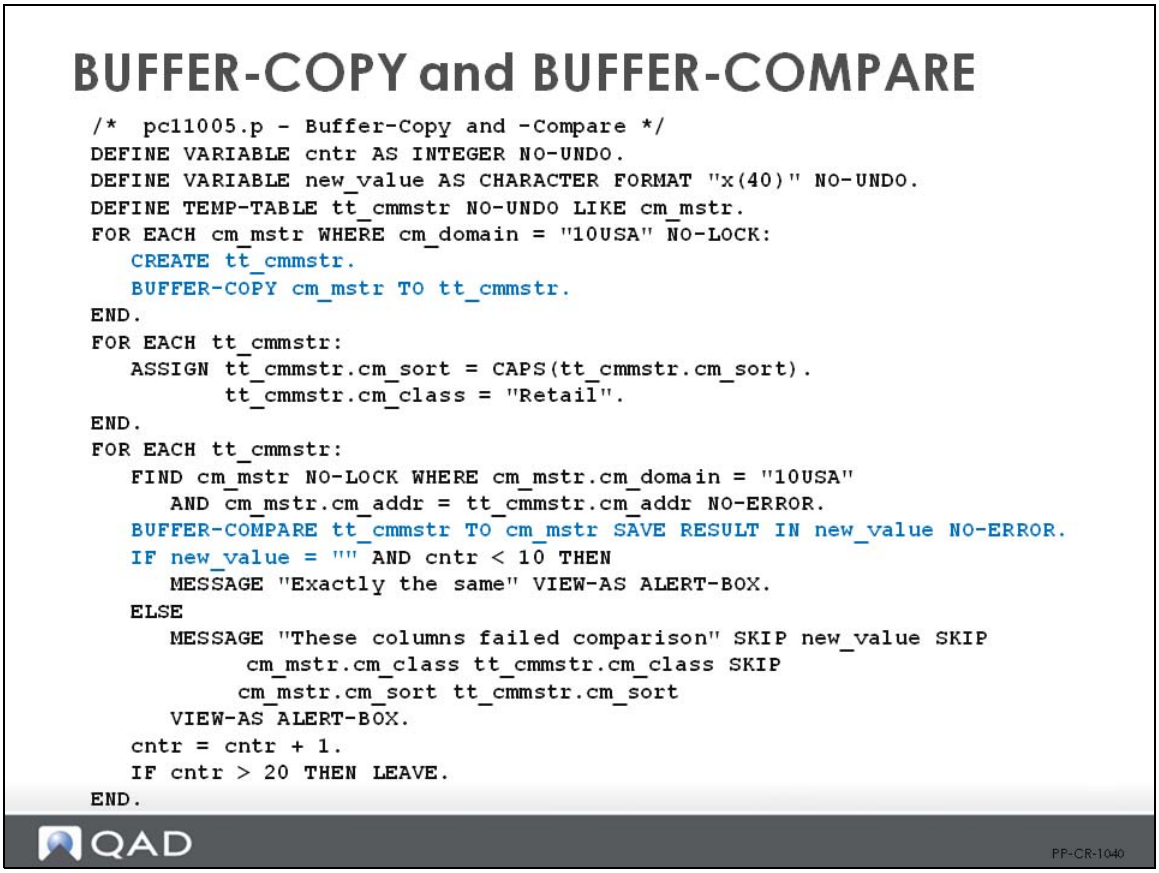
BUFFER-COPY and BUFFER-COMPARE



BUFFER-COPY performs a bulk copy of a source record to a target record by copying each source field to the target field of the same name. You can specify a list of fields to exclude from the bulk copy, or a list of fields to include in the bulk copy.

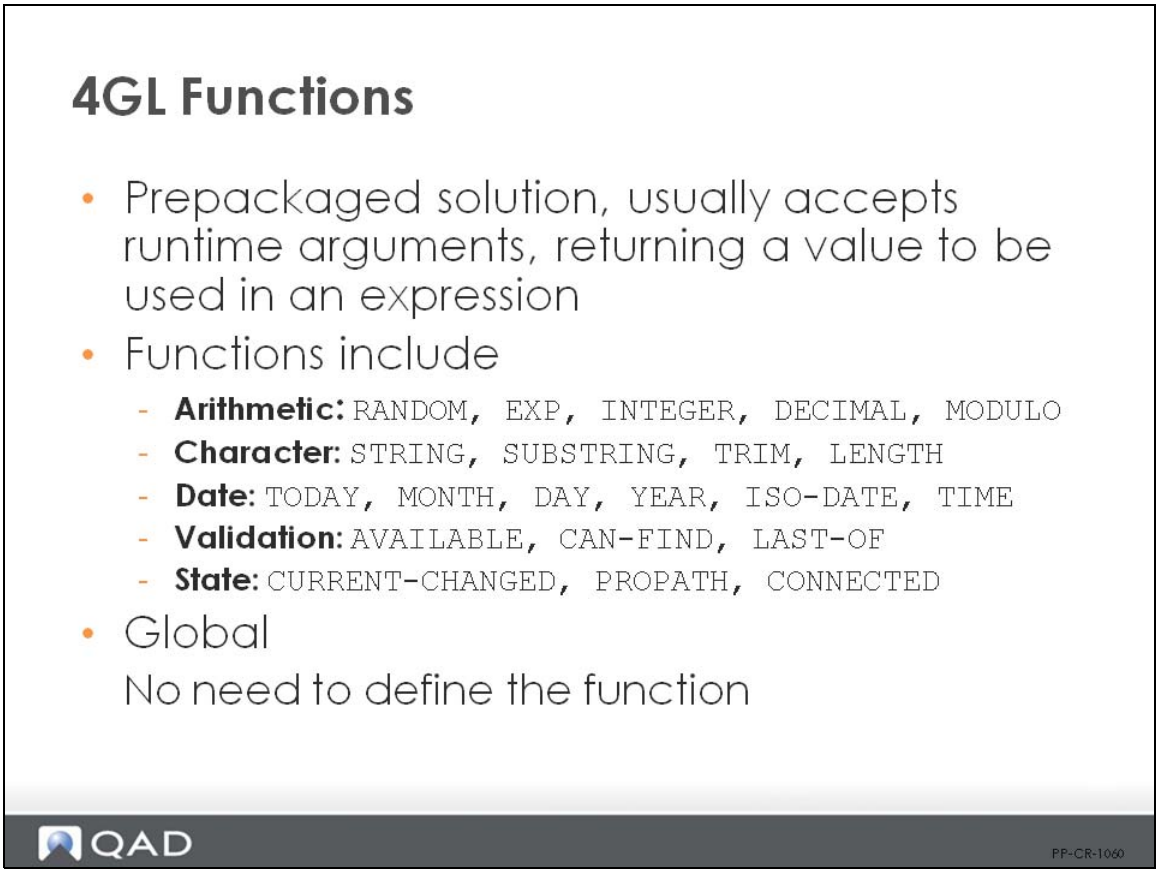
BUFFER-COMPARE performs a bulk comparison of two records (source and target) by comparing source and target fields of the same name for equality and storing the result in a field. You can specify a list of fields to exclude, or a list of fields to include.

BUFFER-COPY and BUFFER-COMPARE

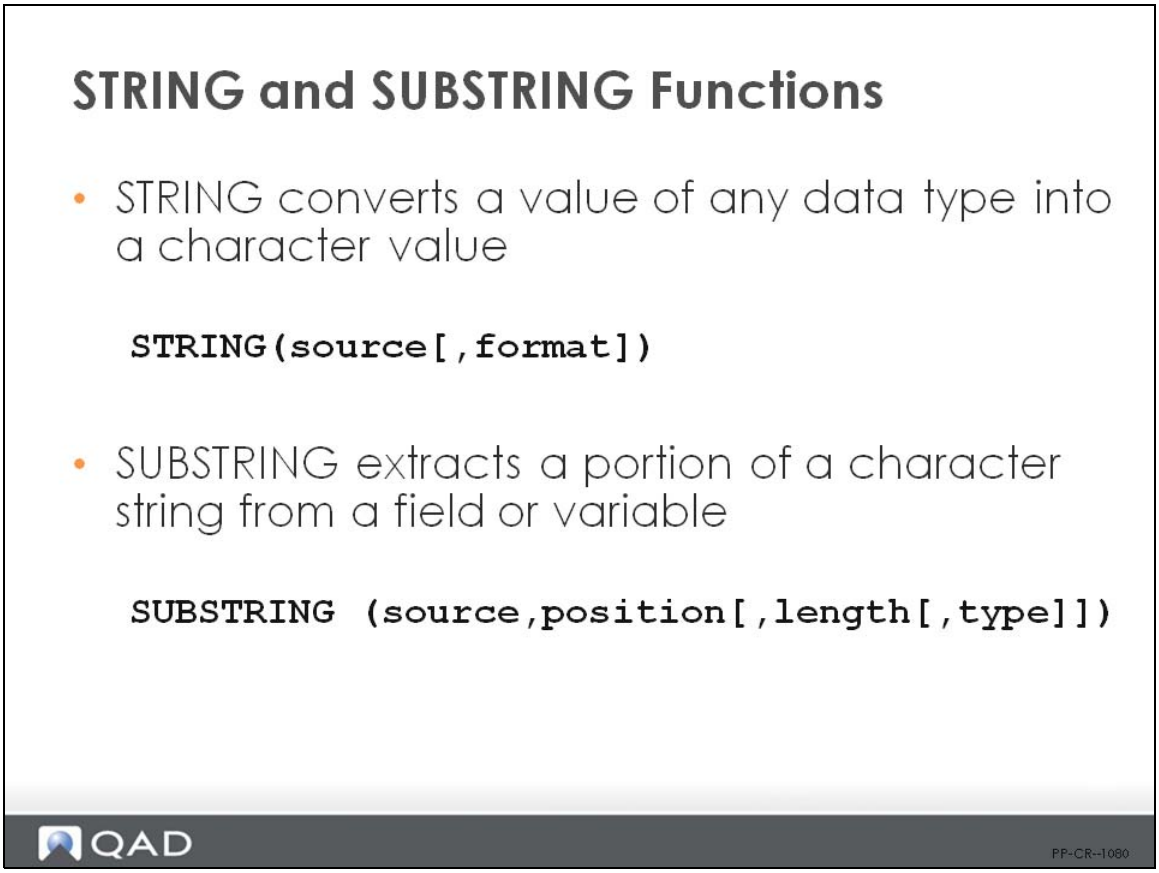


This slide illustrates a code sample using BUFFER-COPY and BUFFER-COMPARE.

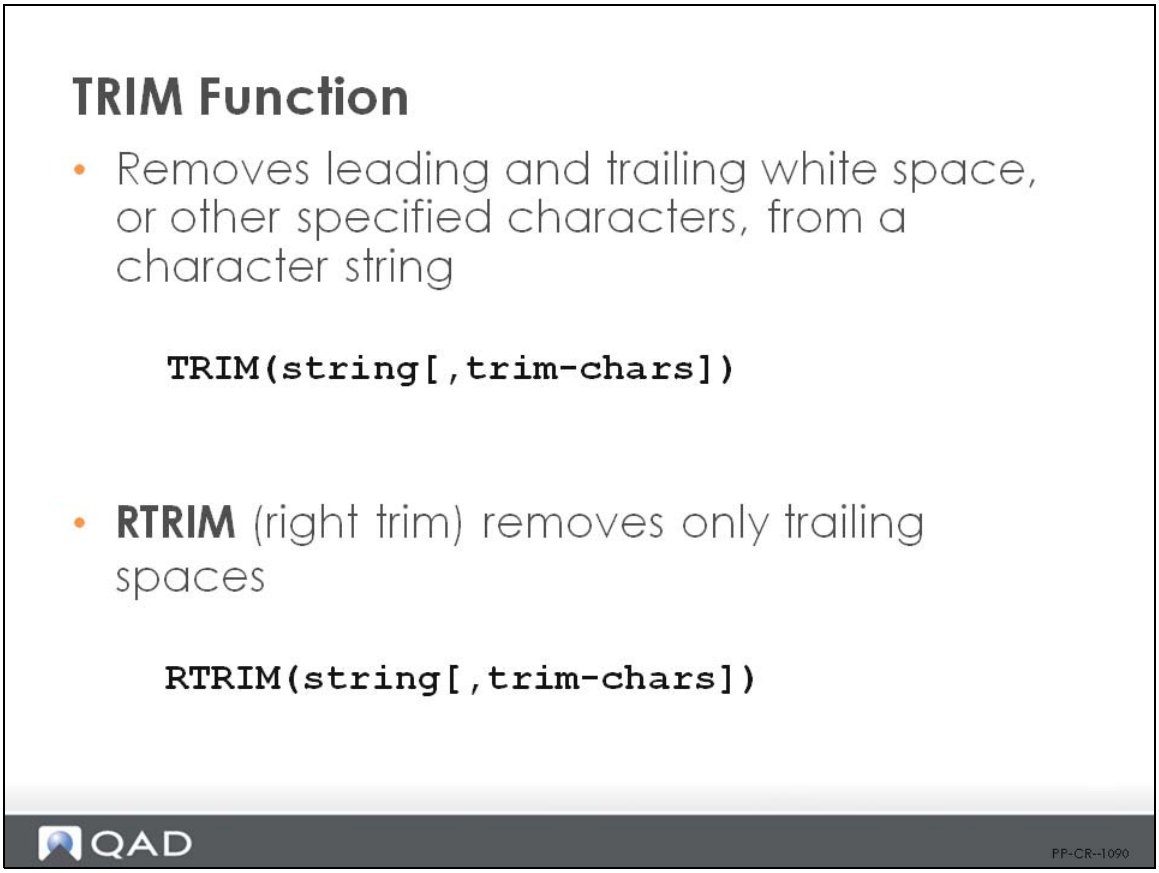
4GL Functions



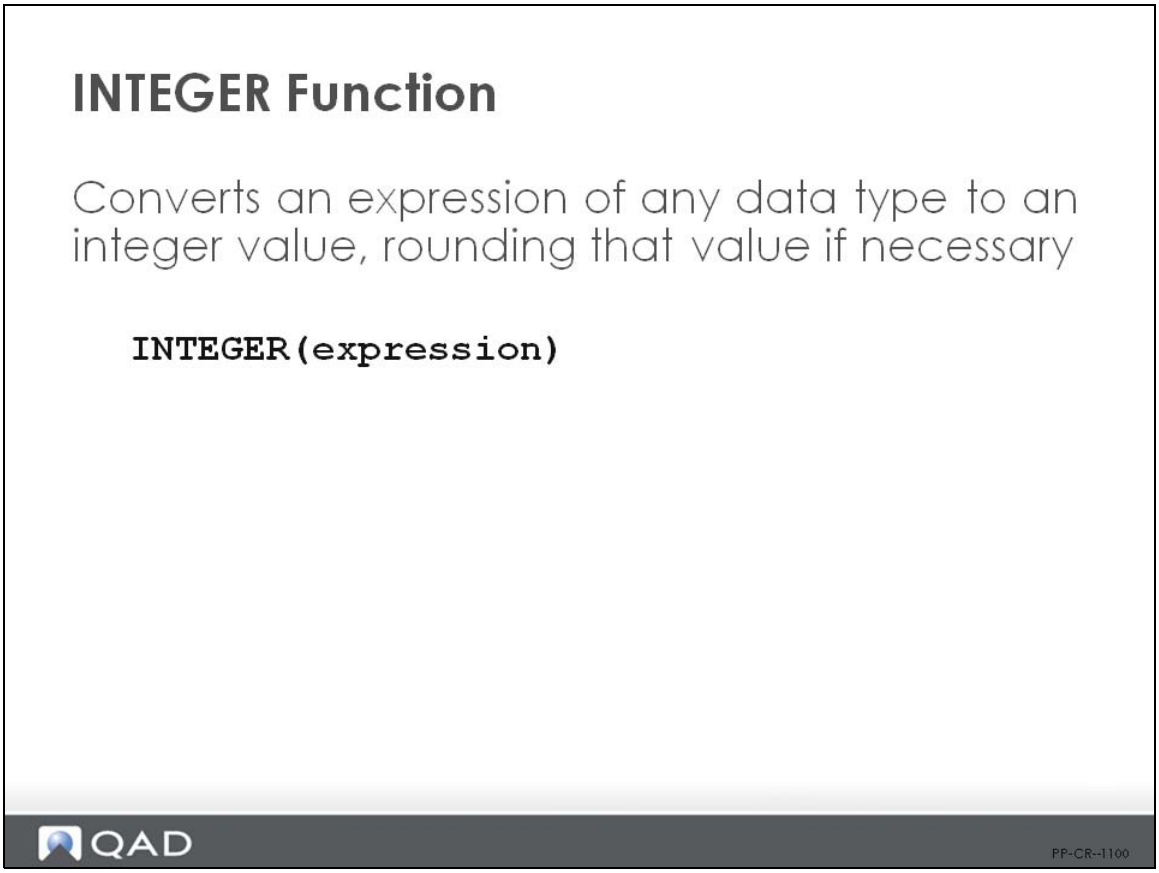
STRING and SUBSTRING Functions



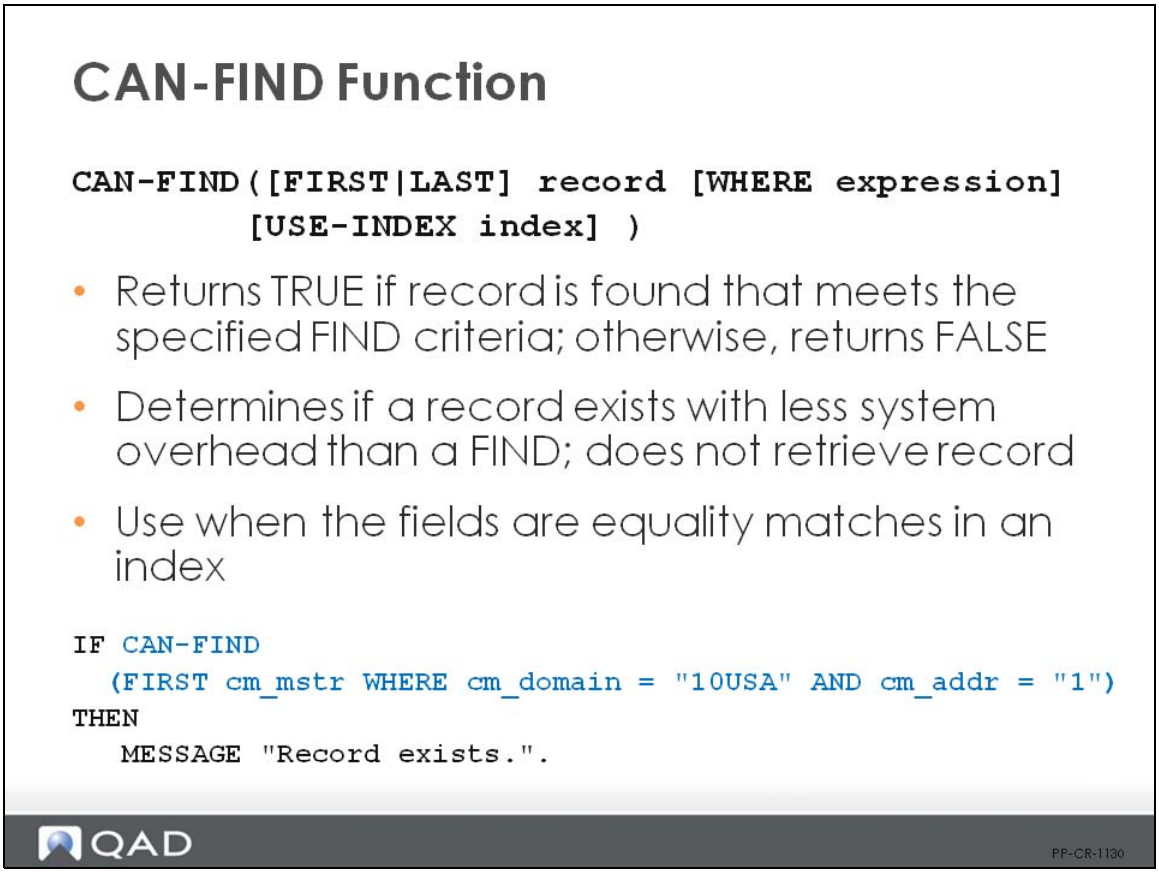
TRIM Function



INTEGER Function



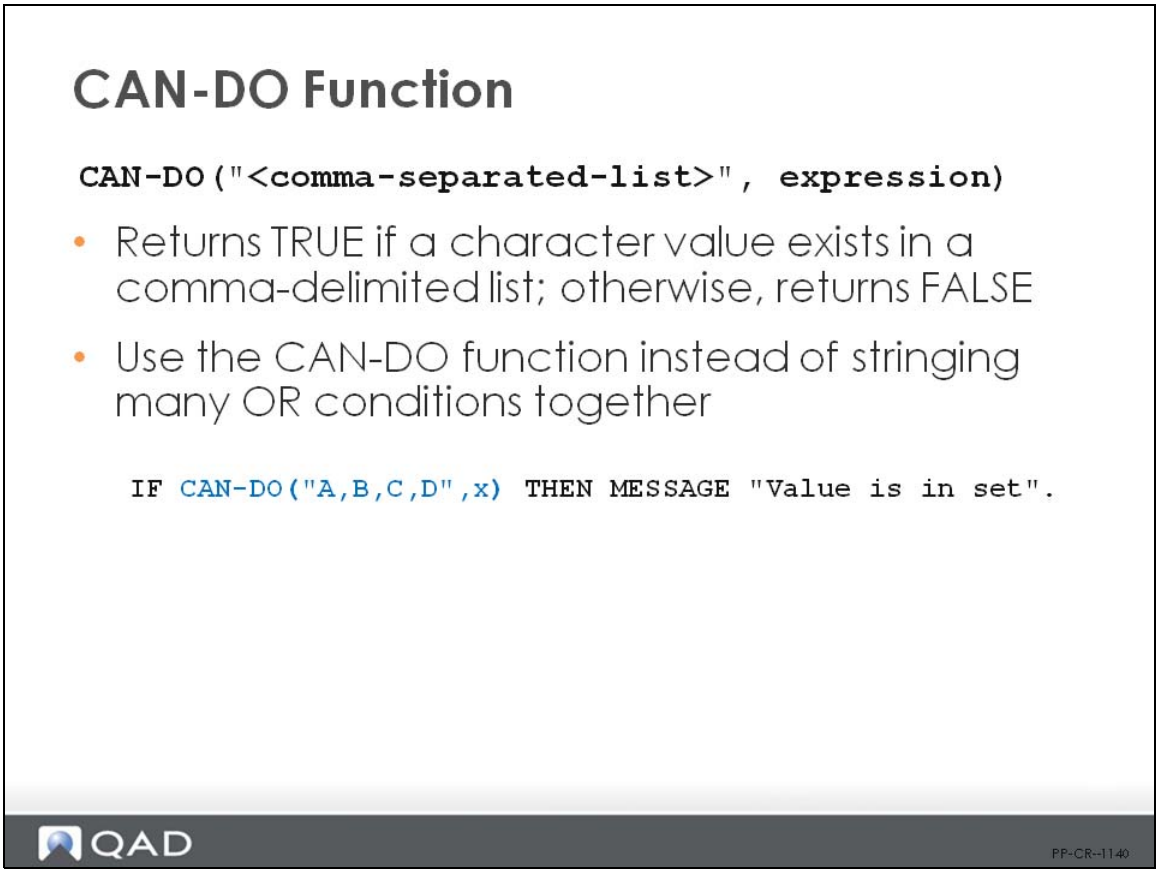
CAN-FIND Function



The CAN-FIND function does not actually pull the record into the record buffer. It is merely a way to test for existence of the record. For this reason, a NO-LOCK statement is not necessary with the CAN-FIND function.

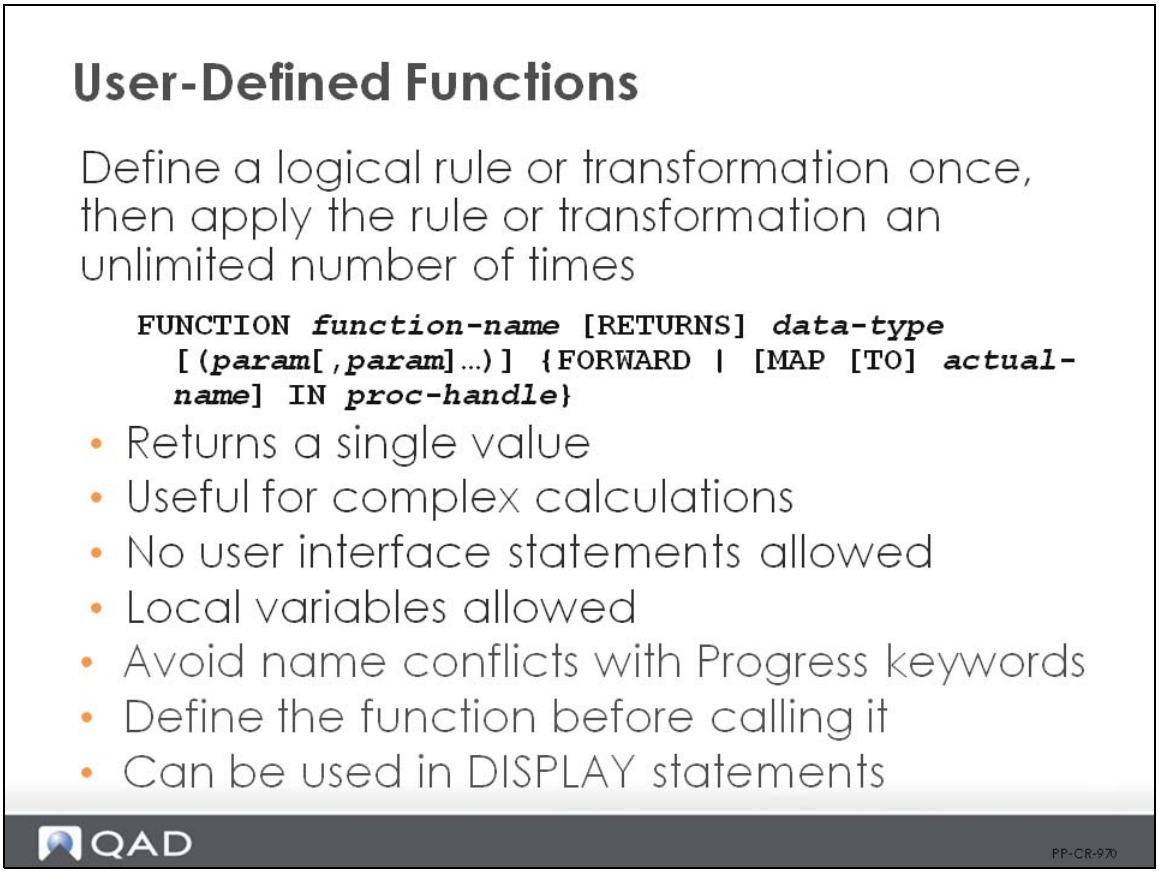
It is preferable to use the CAN-FIND function instead of the FIND statement when you only need to know if the record exists and do not need to access its contents.

CAN-DO Function



If the current value of x is present in the list of arguments (A, B, C, and D), TRUE is returned.

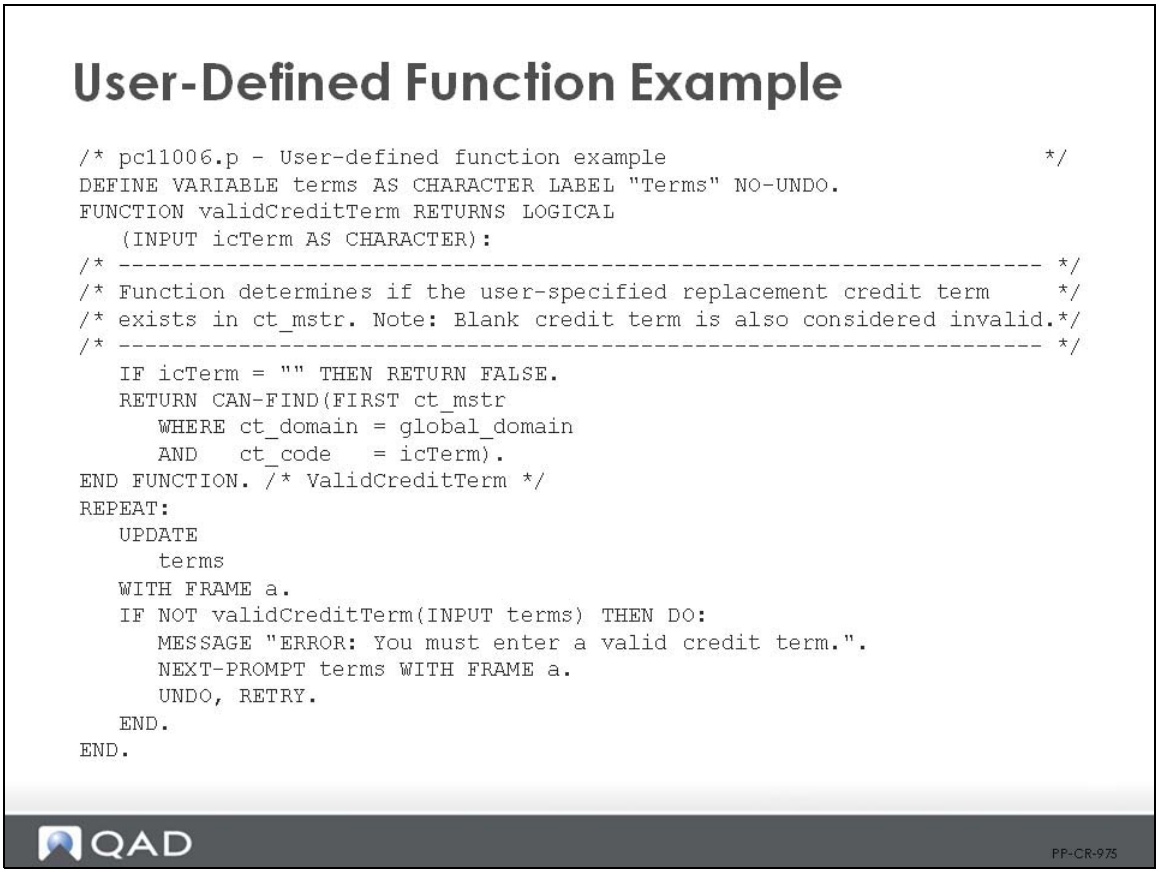
User-Defined Functions



Using a user-defined program, you can define a logical rule or transformation once, then apply the rule or transformation an unlimited number of times.

Do not name your function with the same name as a Progress keyword or it will override the Progress function.

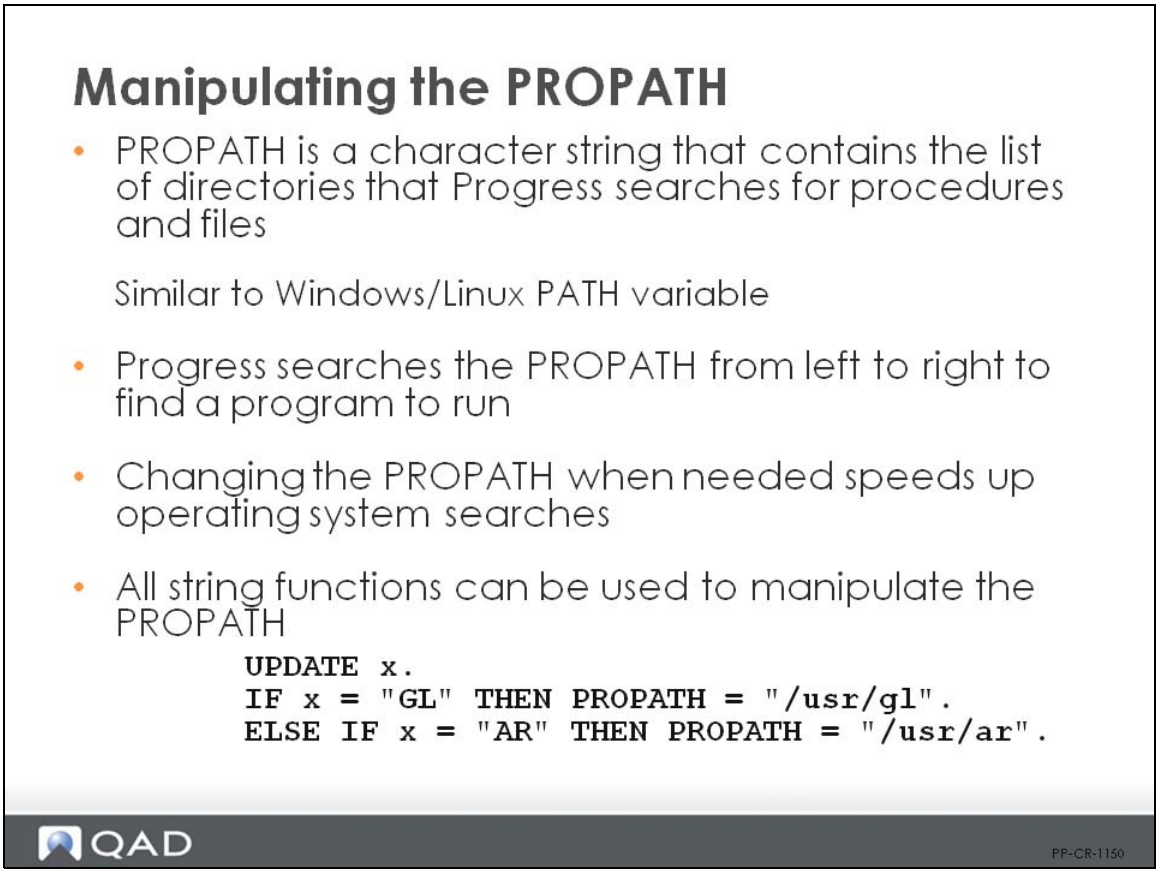
User-Defined Function Example



Program `pc11006.p` illustrates a user-defined function that takes a credit terms code as input and determines whether it is defined in the database. If the credit terms code is blank or cannot be found in the ct_mstr (Credit Terms Master) table, the function returns a false value, indicating the credit terms code is invalid. Otherwise, the function returns a true value, indicating the credit terms code is valid.

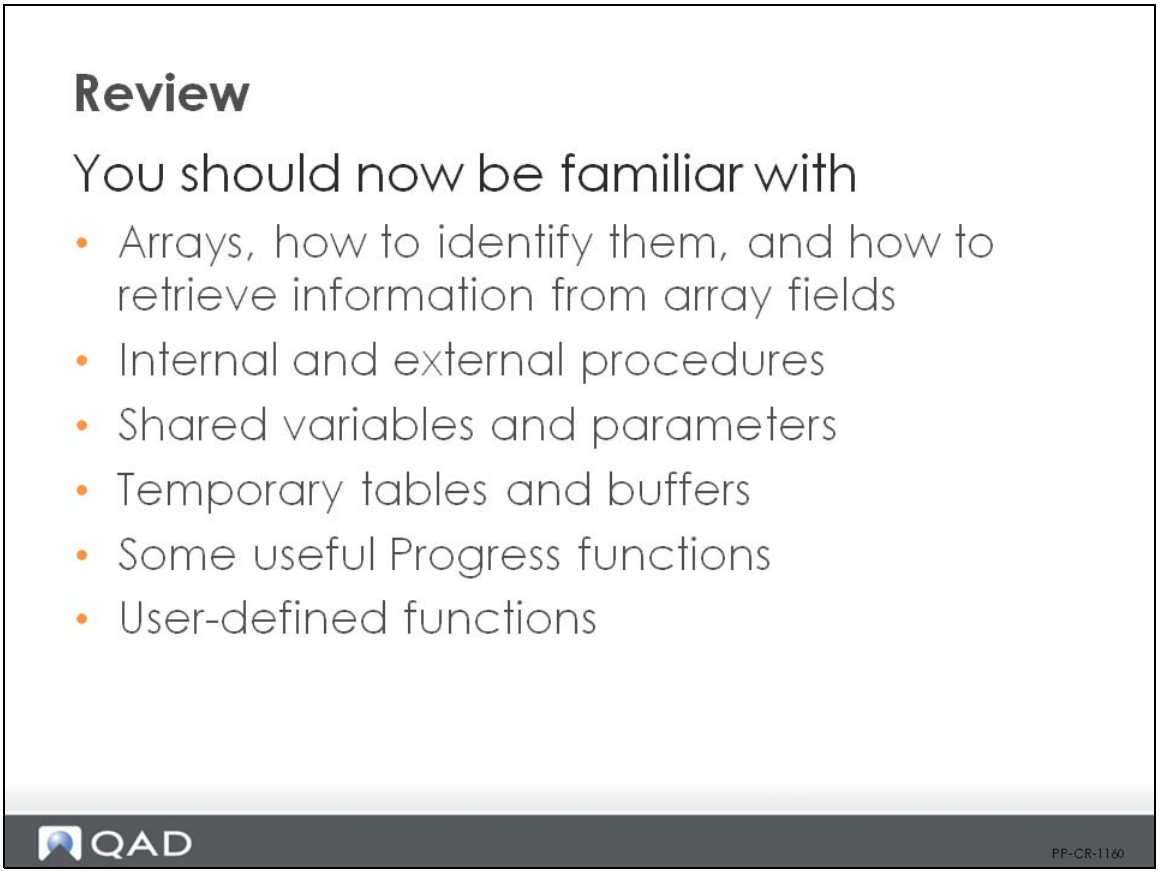
The function is defined before it is used in the program. The program calls a user-defined function typically through an IF statement, passing the value to be evaluated inside parentheses, and acting upon the returned result. The passed value is treated as an input parameter to the function. All functions have an implied output parameter defined by the RETURNS statement of the function definition. This eliminates the need for an explicit OUTPUT parameter in the function call.

Manipulating the PROPATH



In an Enterprise Edition database, each domain can have a different PROPATH. This feature supports unique code execution for each domain.

Review



In this lesson, you learned:

- That array fields occur more than once for each record in a file and can be displayed by using the DO or REPEAT statements
- The use of shared variables, procedure calling, and passing parameters
- The use of temporary tables to create temporary work files and buffers to access more than one record at a time in a program
- How to use built-in Progress functions to return and manipulate values

Chapter 3

Advanced Topics

Agenda for Section 3

This section of the class consists of four lessons:

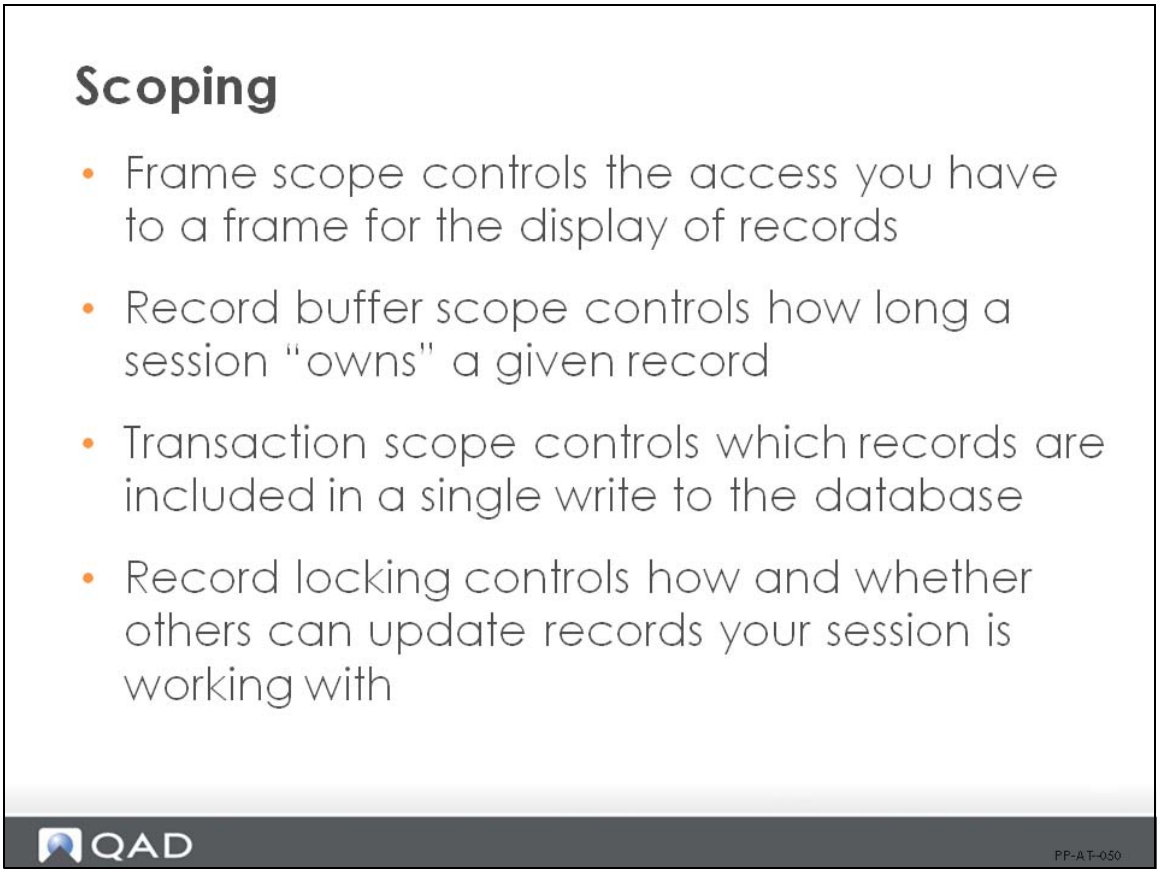
- Lesson 12: Scoping
- Lesson 13: ProDataSets
- Lesson 14: Triggers and Events (Optional)
- Lesson 15: Queries and Handles (Optional)

Lesson 12: Scoping

Upon completion of this lesson, you will be familiar with the following topics:

- Frame scope
- Record scope
- Transaction scope

Scoping



Scope is the period in which a frame is active or during which an individual record exists in the record buffer or the duration of a commit/rollback event. Scope describes the persistence of a transaction, record, or frame for purposes of access and memory usage.

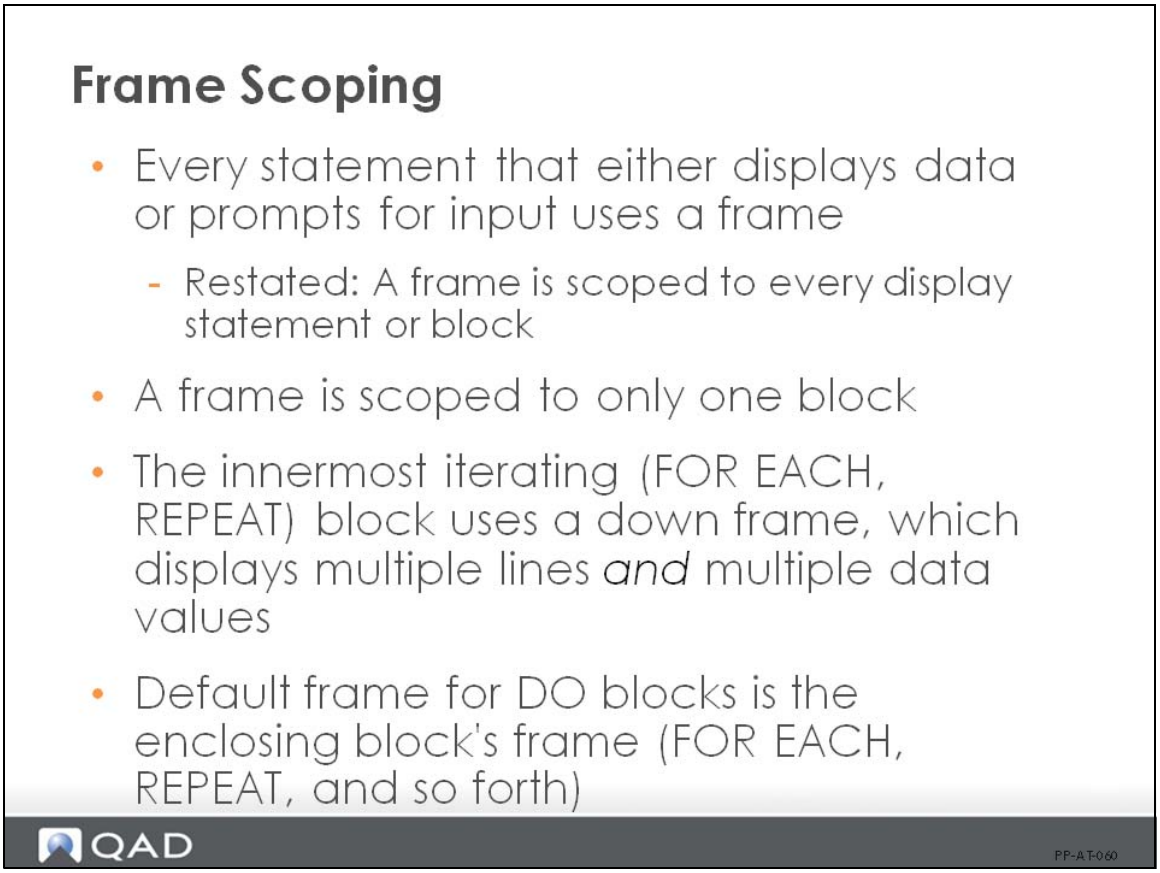
Scope is a concept unique to Progress and must be understood in order to program well in Progress. The scope of a record or frame depends on the nature of the programming block in which it was introduced.

The Progress Buffer Manager creates, maintains, and deallocates records in memory. It determines when to read information from the database into the record buffer.

Scope applies to:

- Frame and controls activation, attributes, and field-level display
- Records and controls buffer creation/close
- Transactions and controls when changes are committed to the database

Frame Scoping



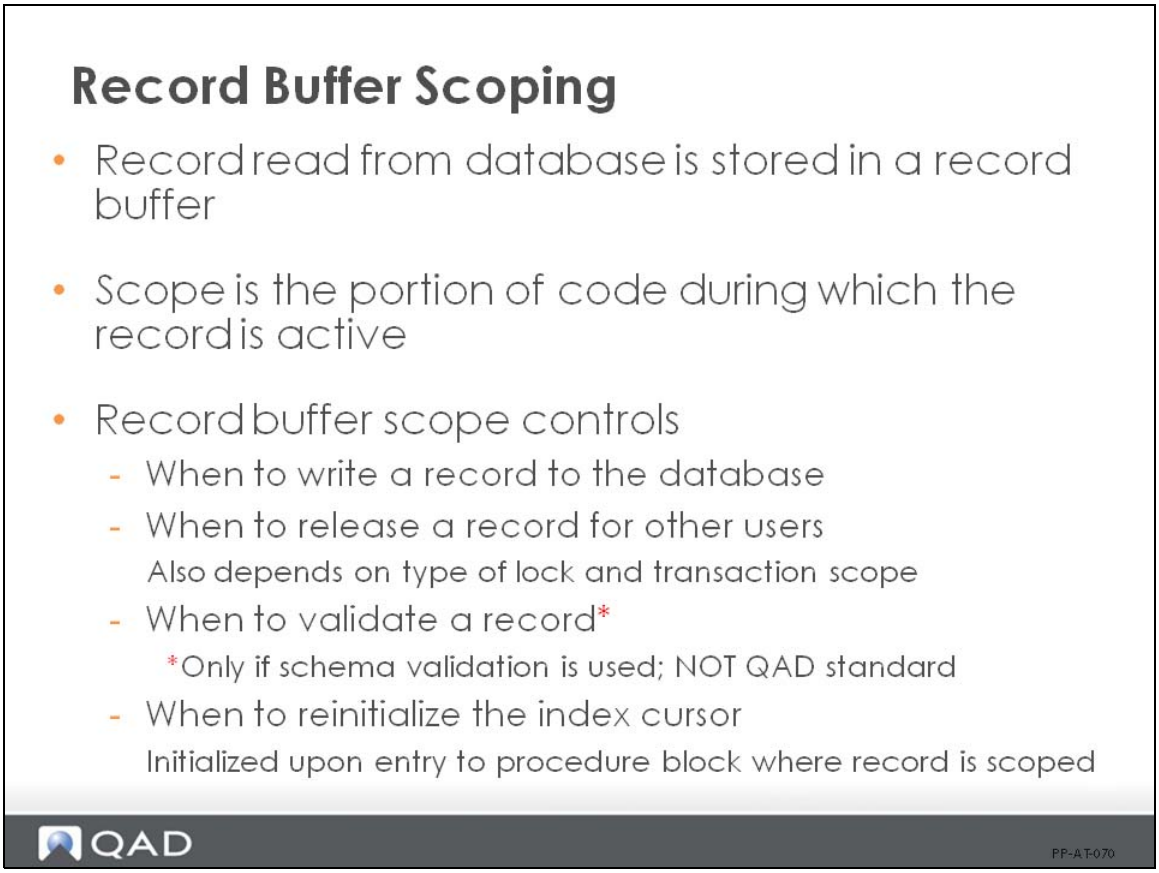
Frame scoping is monitored by the Frame Manager, which determines when to activate a frame, what the frame attributes are, when to perform frame handling tasks, and when to deactivate (hide) the frame.

To determine if the frame will be a Down Frame, move through this checklist:

1. Is the block an iterating block? YES.
2. Is the default frame scoped to the block. YES.
3. Is it the innermost iterating block. YES (no other block nested inside it).

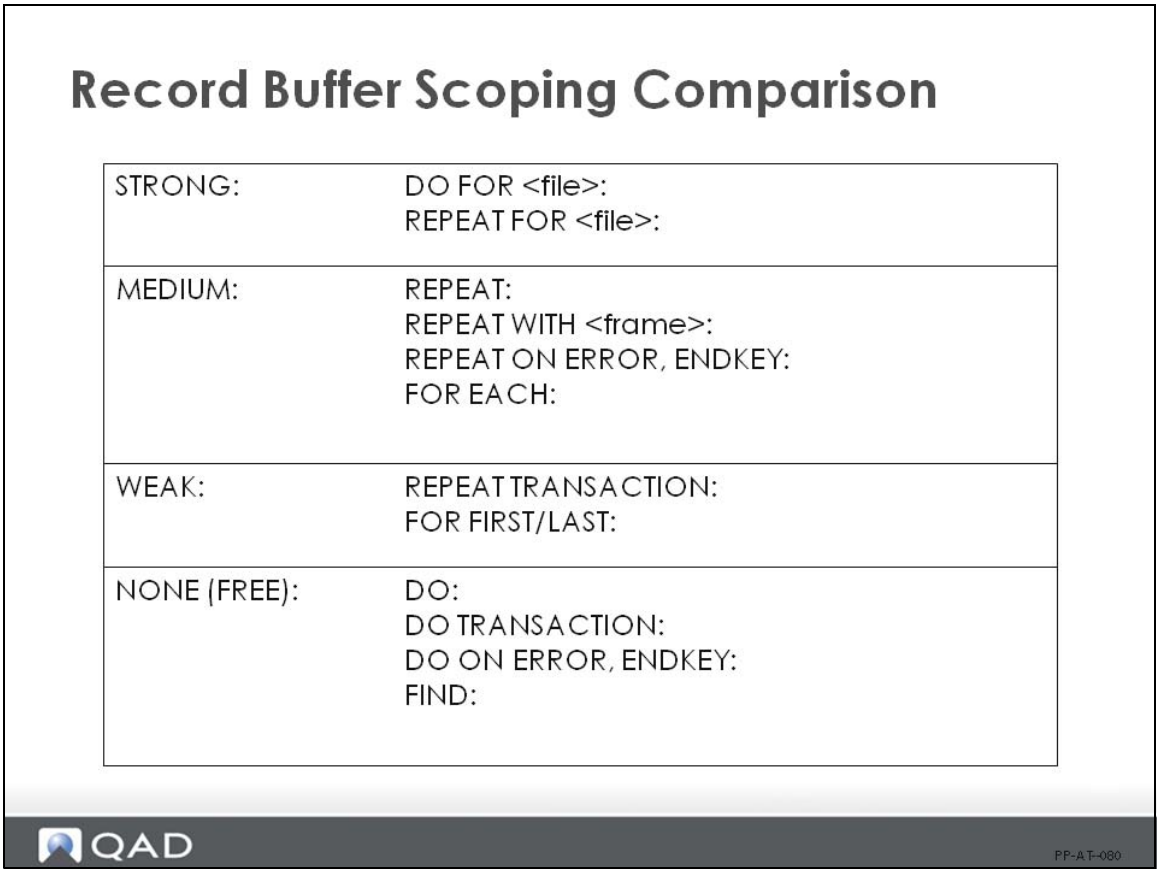
If all answers are YES then it will be a DOWN frame. All others will be a one-down frame.

Record Buffer Scoping



Note QAD does not use database validation. Instead, QAD validates record input in the code.

Record Buffer Scoping Comparison

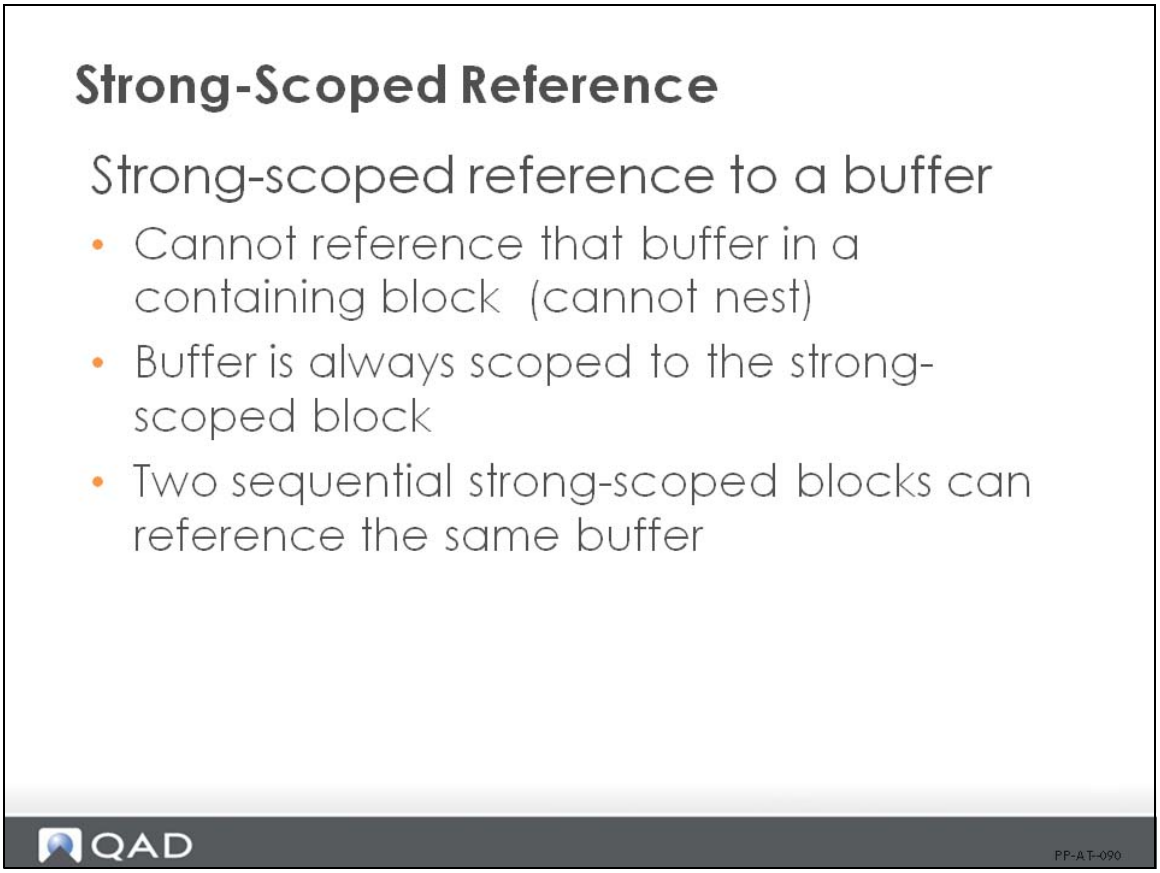


Strong Scope. No reference of any kind can be made to the record or frame outside the scope of the record. Progress does not compile the procedure if this is encountered.

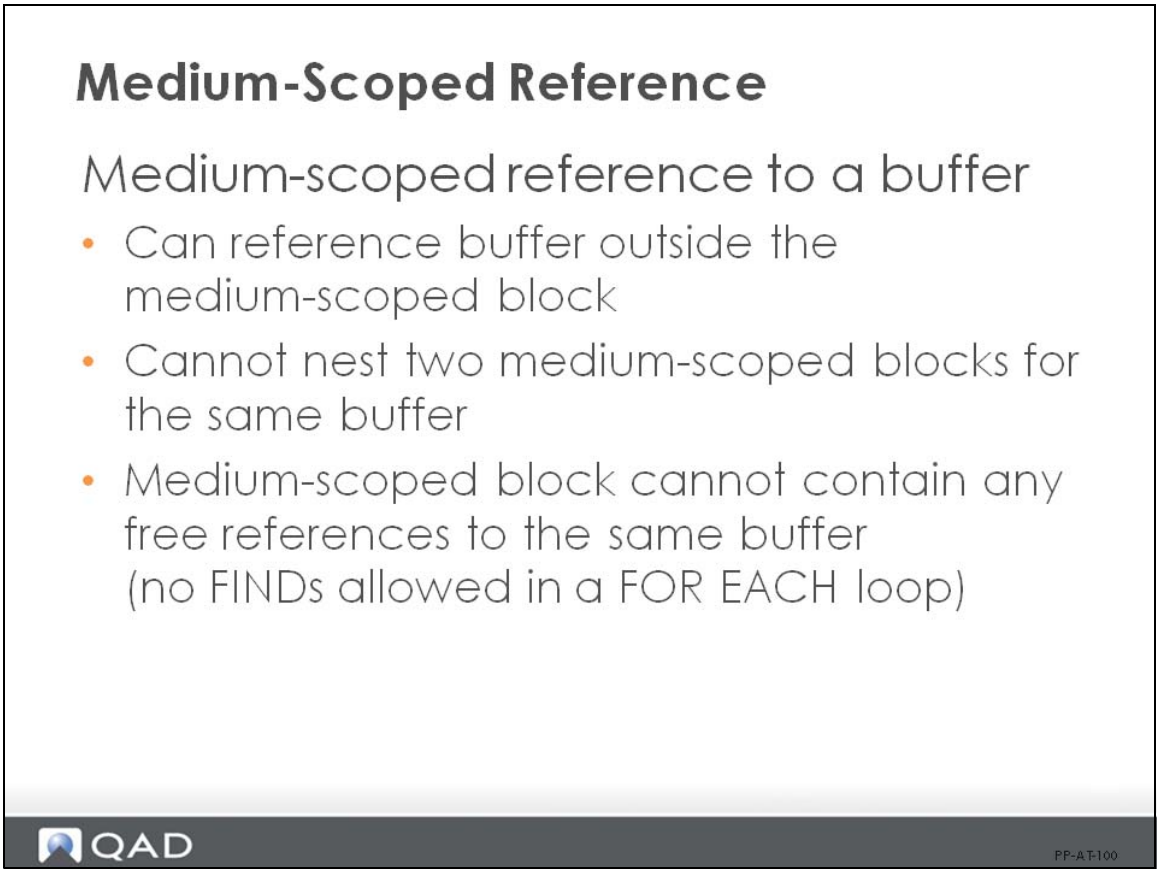
Medium Scope. Any reference to a record or its fields outside the record scope will generate an error: “no xxx record is available.”

Weak Scope. Limits transaction scope, but not record or frame. Changes are committed to the database at the end of the enclosing block or procedure. Fields are still available for reference outside of the block.

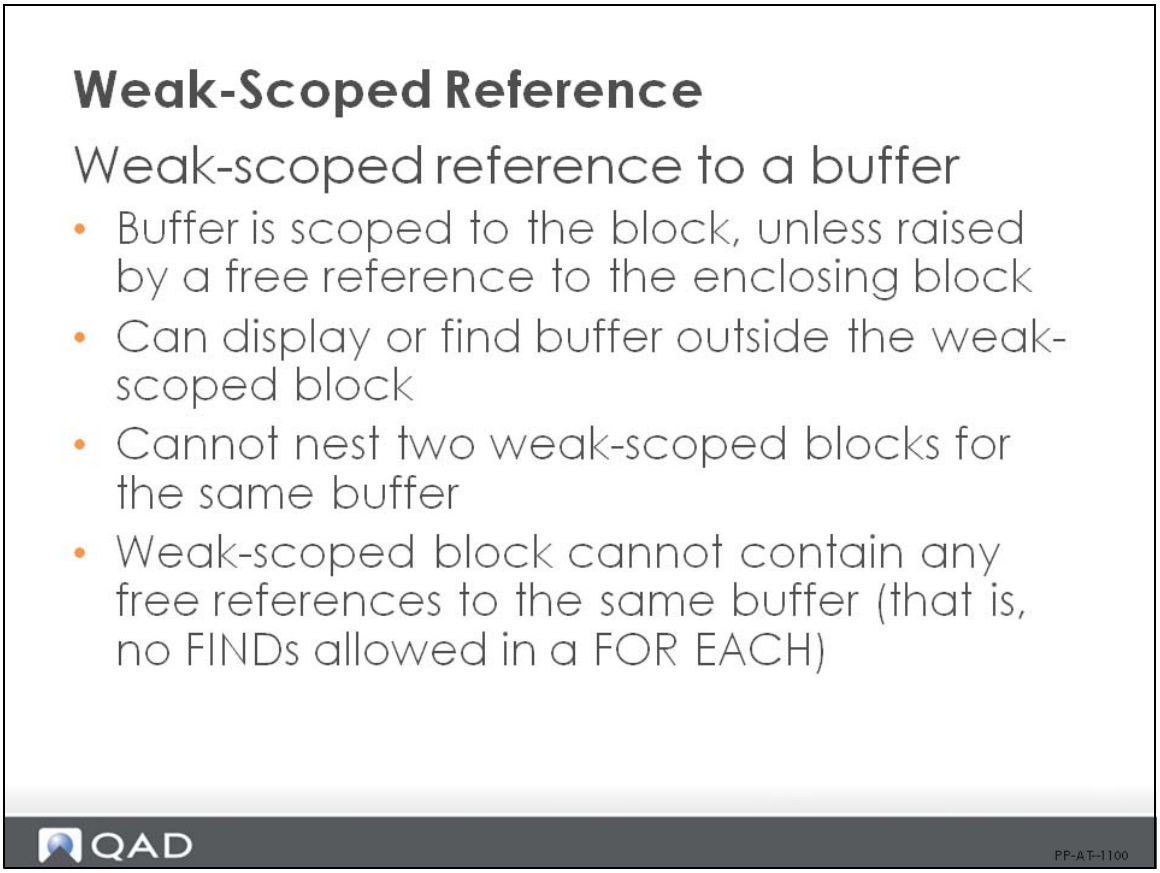
Strong-Scoped Reference



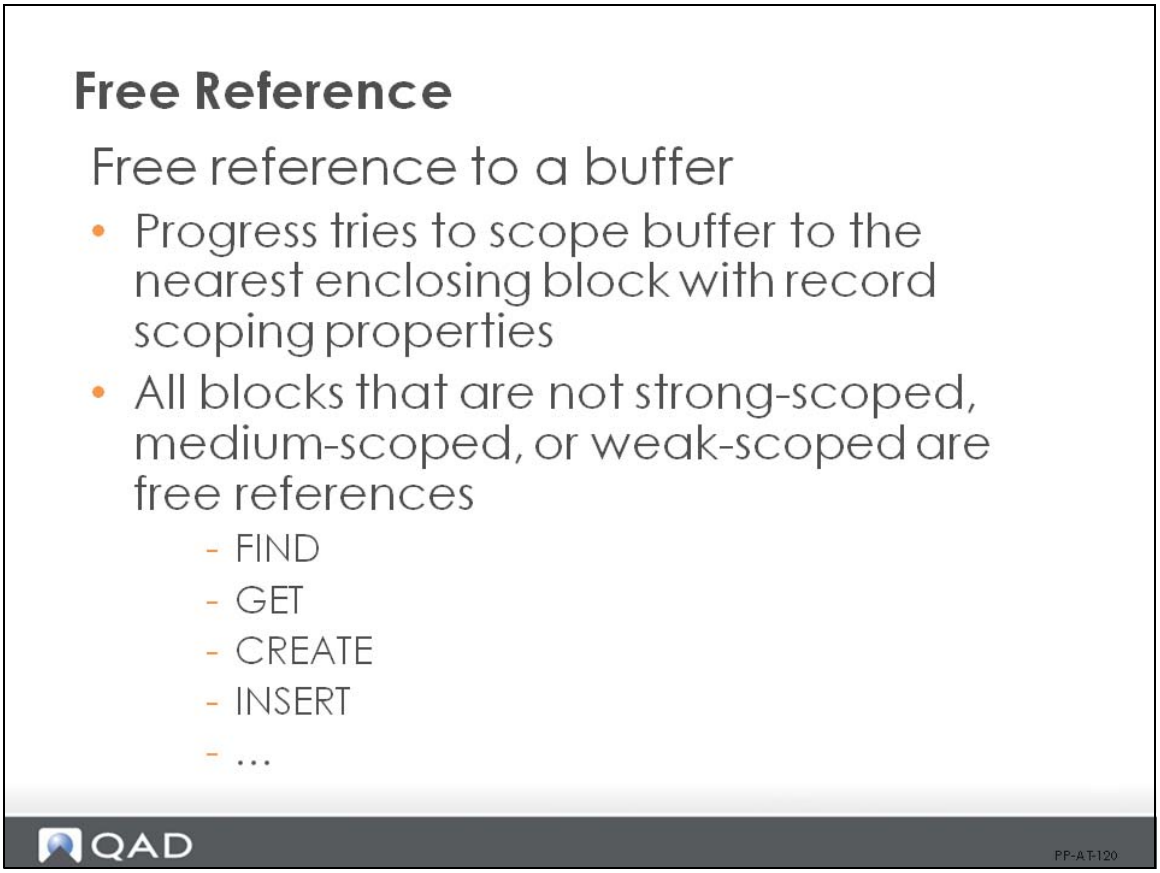
Medium-Scoped Reference



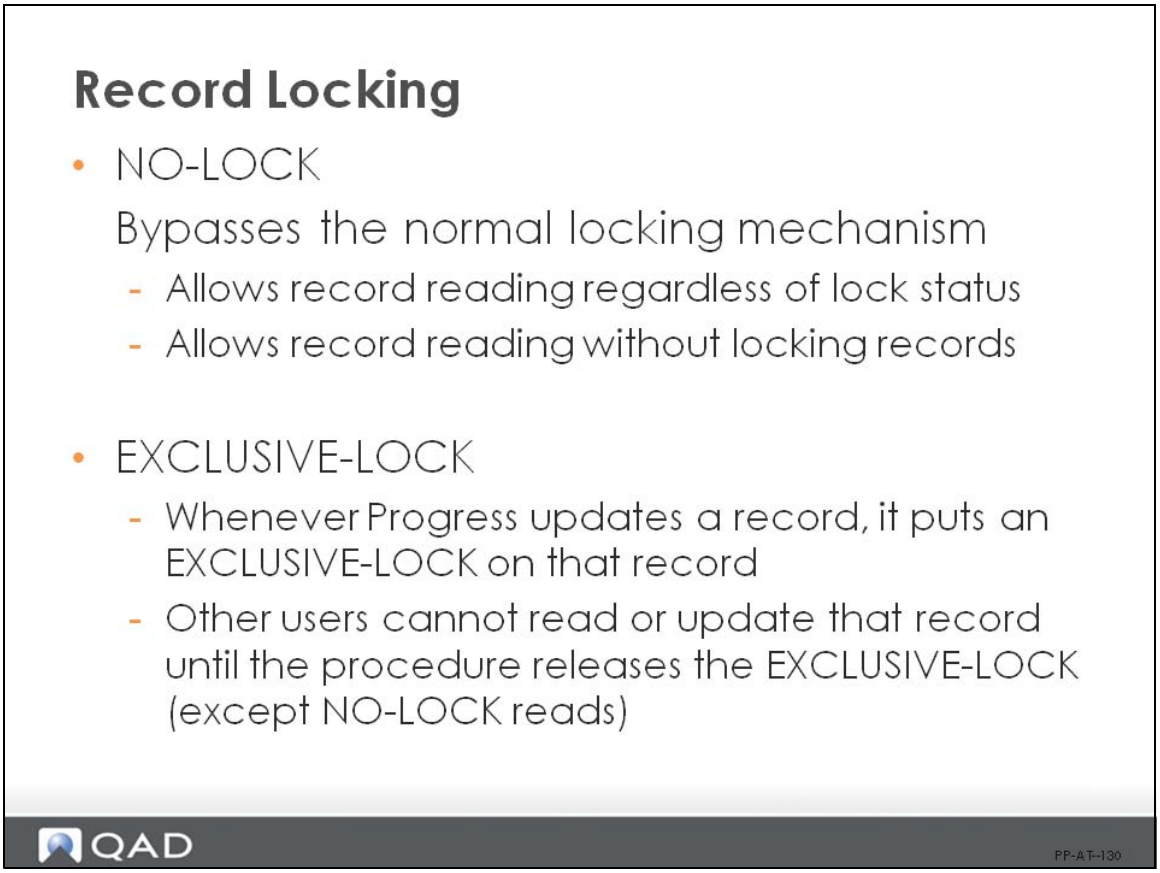
Weak-Scoped Reference



Free Reference



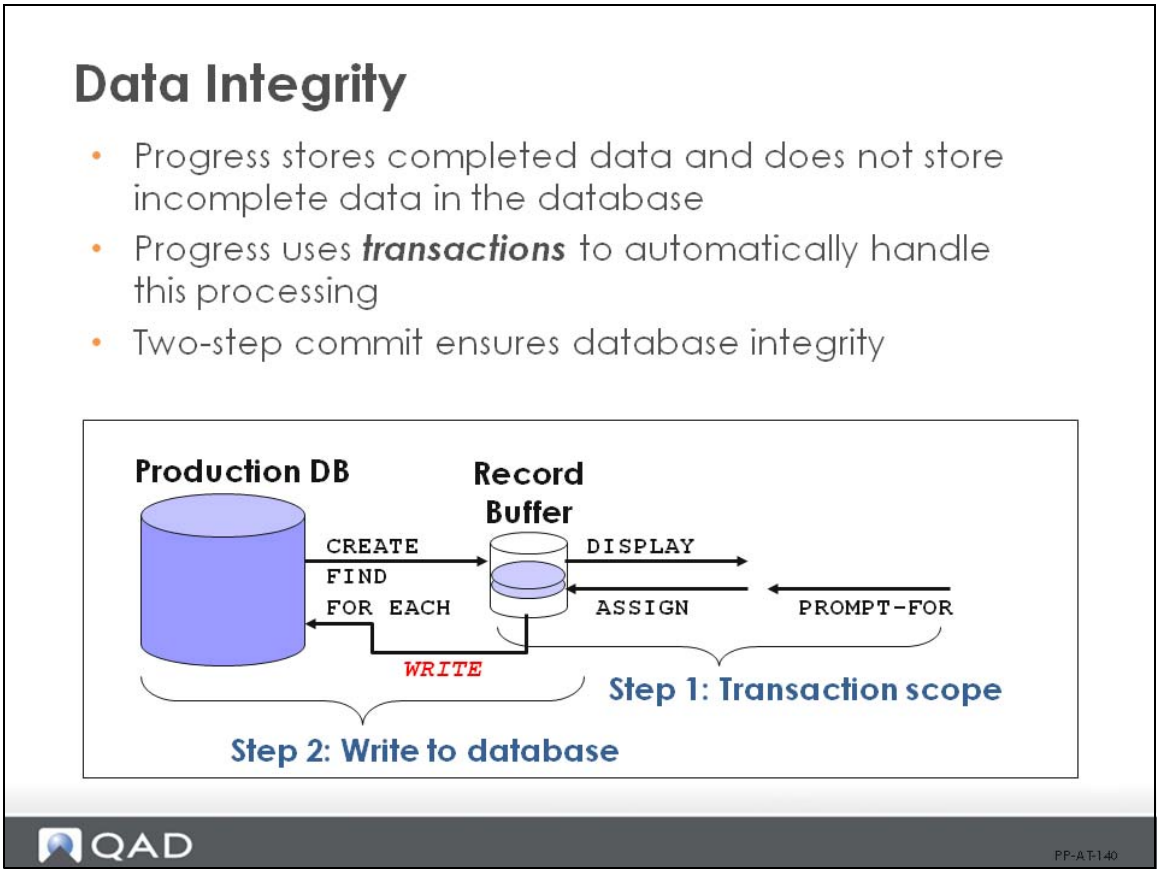
Record Locking



If you do not remember to specify a specific lock method, Progress applies SHARE-LOCK by default. This is an anticipatory form of locking that can result in two programs contending for the same record.

Important Never use the SHARE-LOCK feature. Always specify a LOCK condition on all database record retrieval statements other than CAN-FIND.

Data Integrity

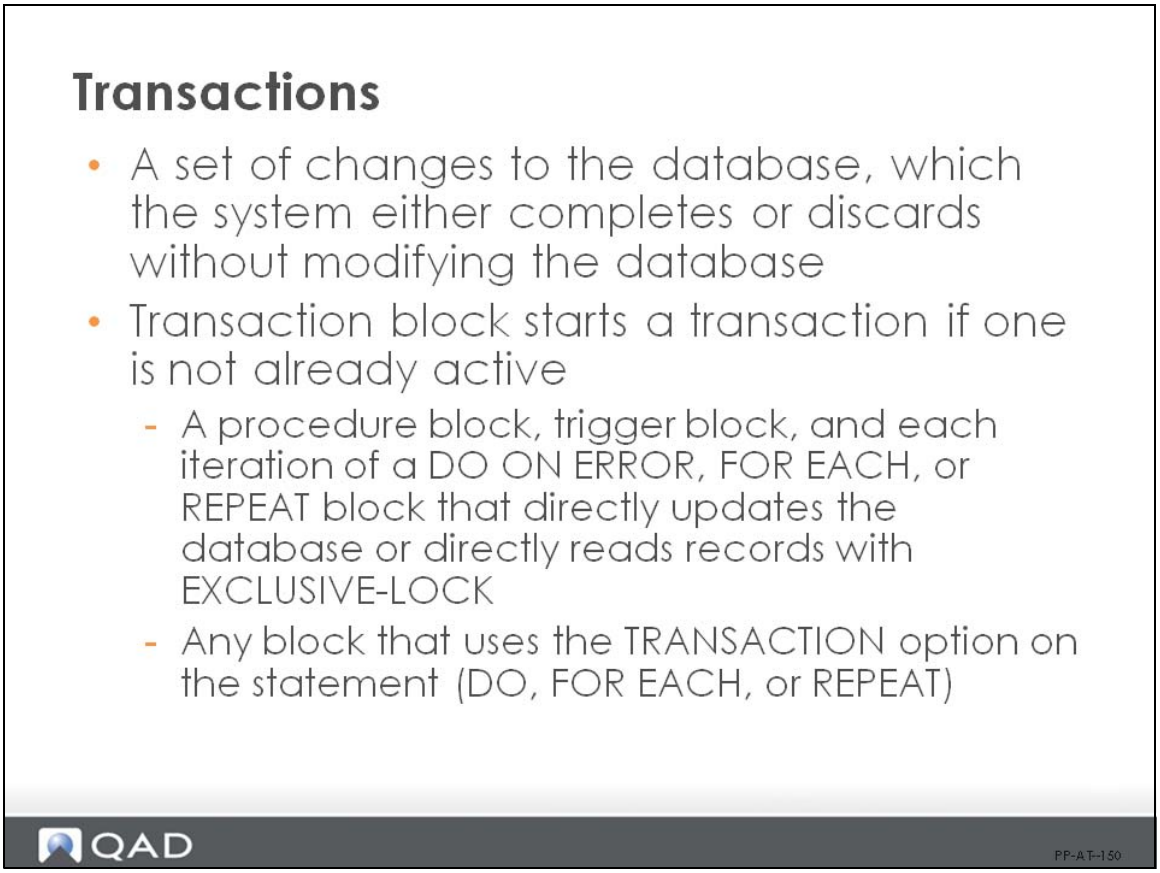


Write is not a Progress 4GL command. It is something that Progress does for you in the background.

There is no explicit commit command in Progress.

The RELEASE command does not force a write to happen. The command only disables the record in the record buffer so you cannot make further changes to it after the record has been written.

Transactions



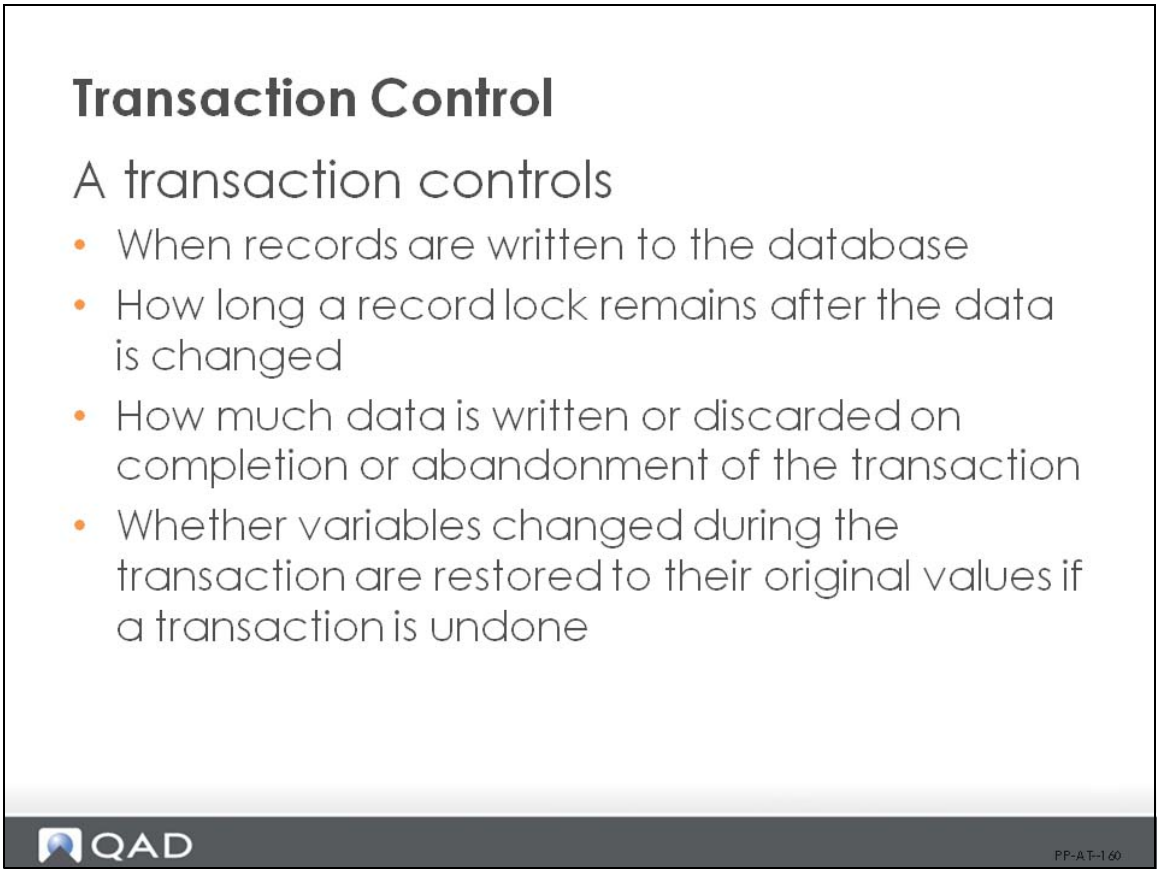
A transaction defines a time period during which a change or series of changes to variables or the database can be committed or undone.

Transactions are not as tangible as frame or record scope. They are handled by the Transaction Manager in conjunction with the Lock Manager.

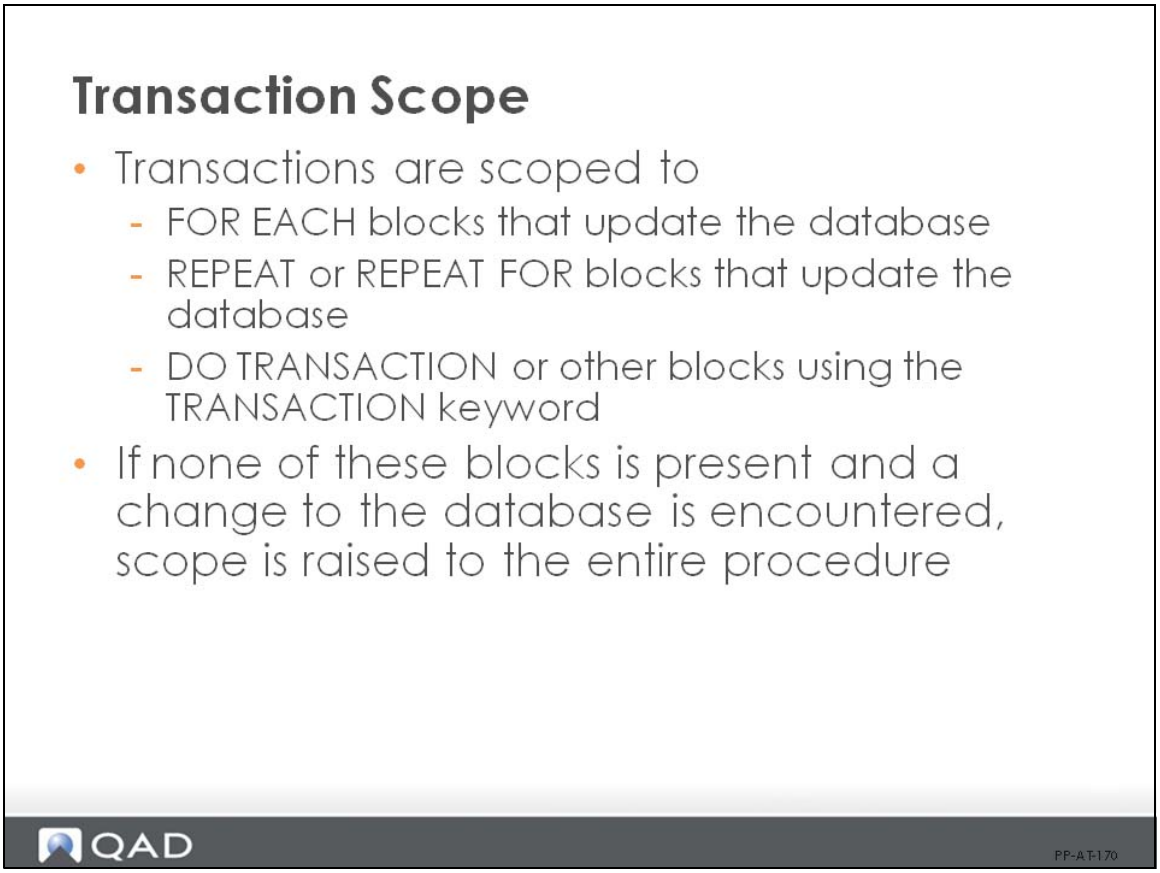
A transaction remains active for every block that is entered and every program that is called down the stack until the transaction is completed.

Keep transaction scope small and do not enable any user interaction in the scope because records are held for the duration of time the user takes to respond. Remember… the user could go to lunch!

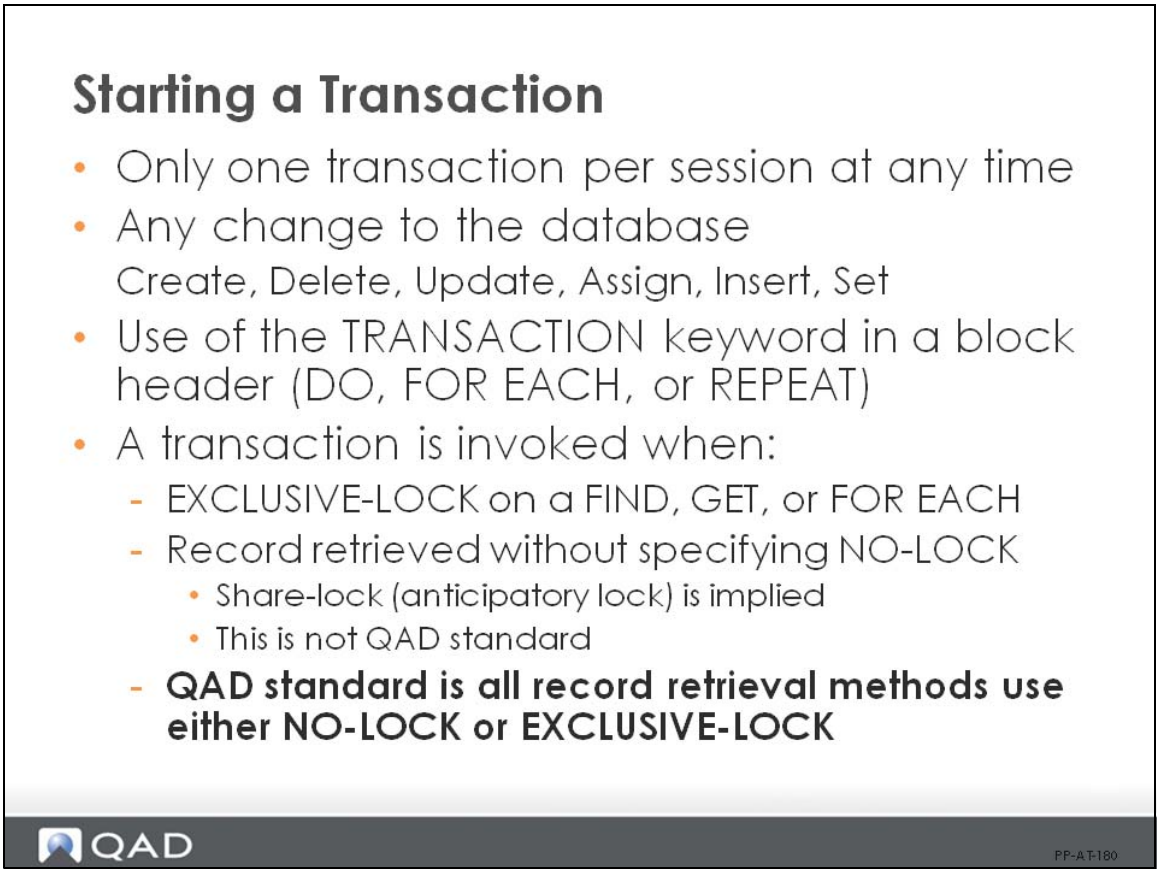
Transaction Control



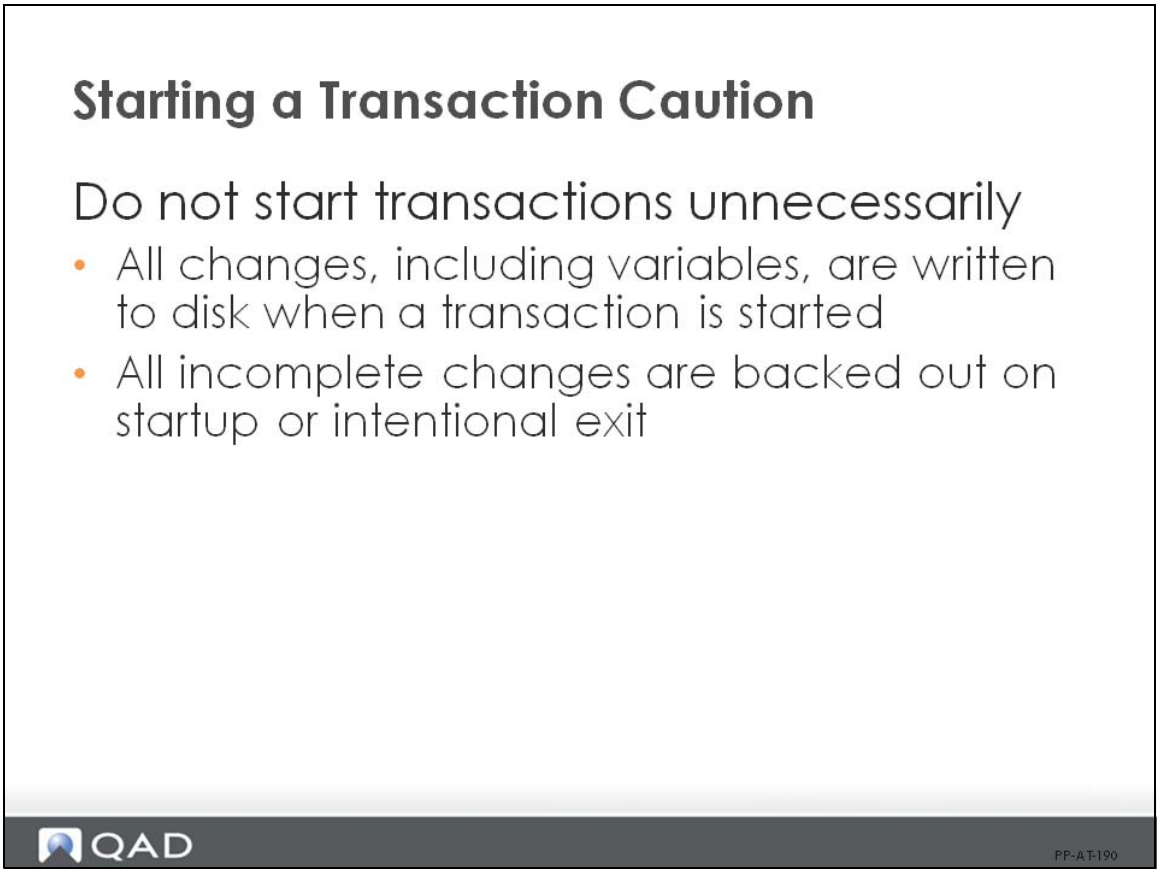
Transaction Scope



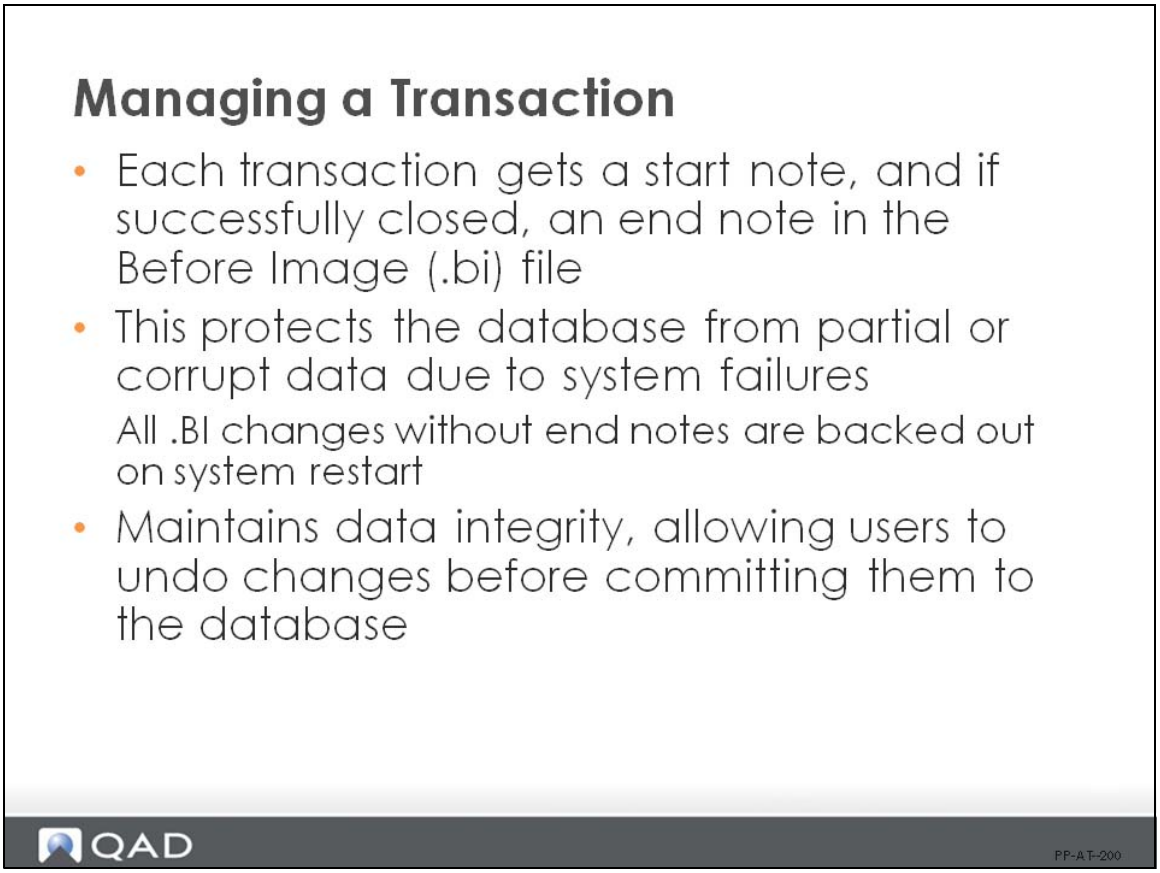
Starting a Transaction



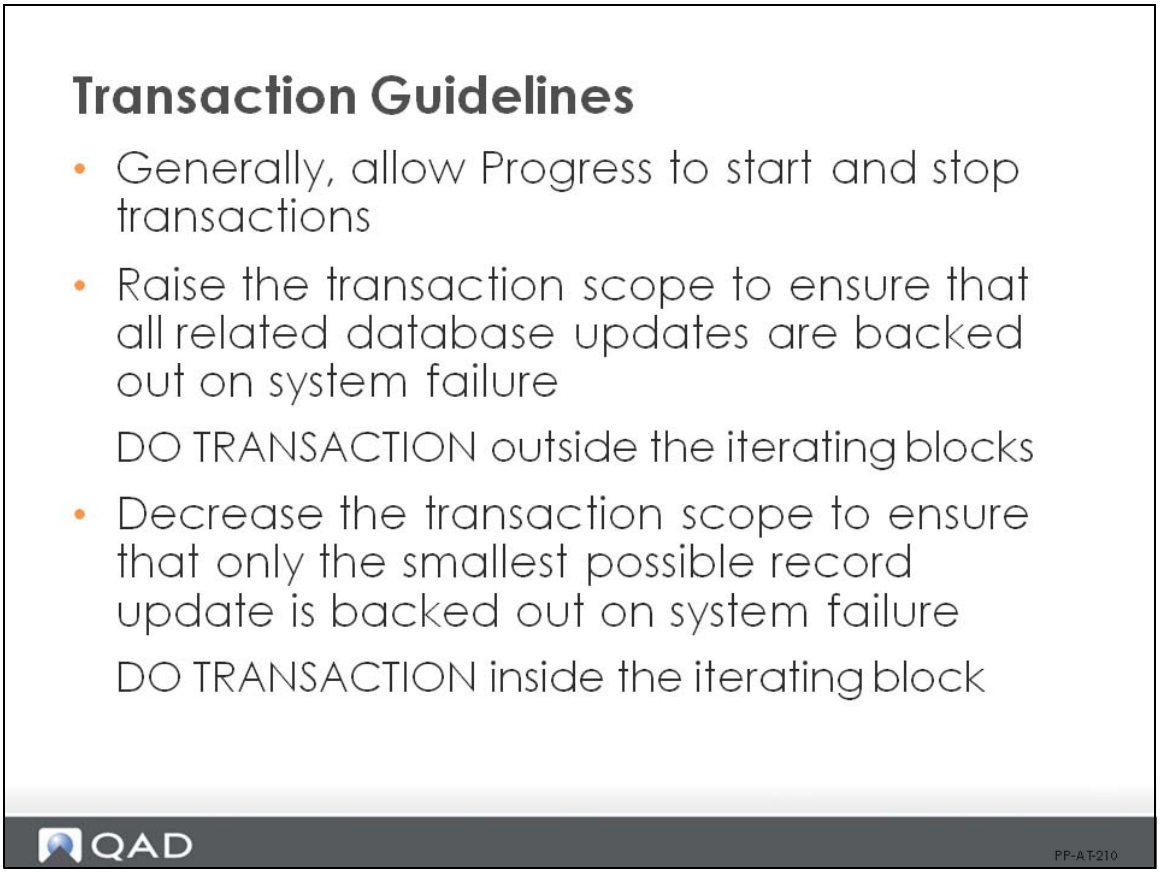
Starting a Transaction



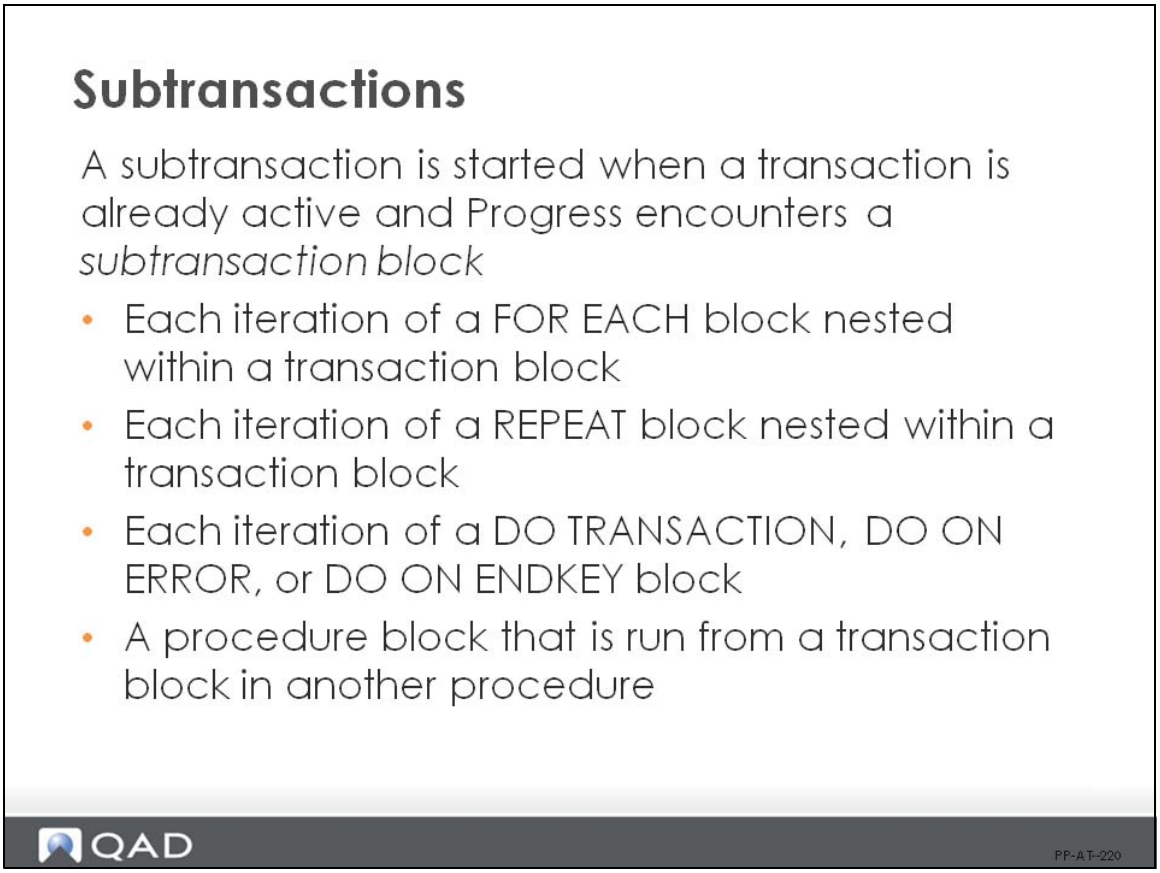
Managing a Transaction



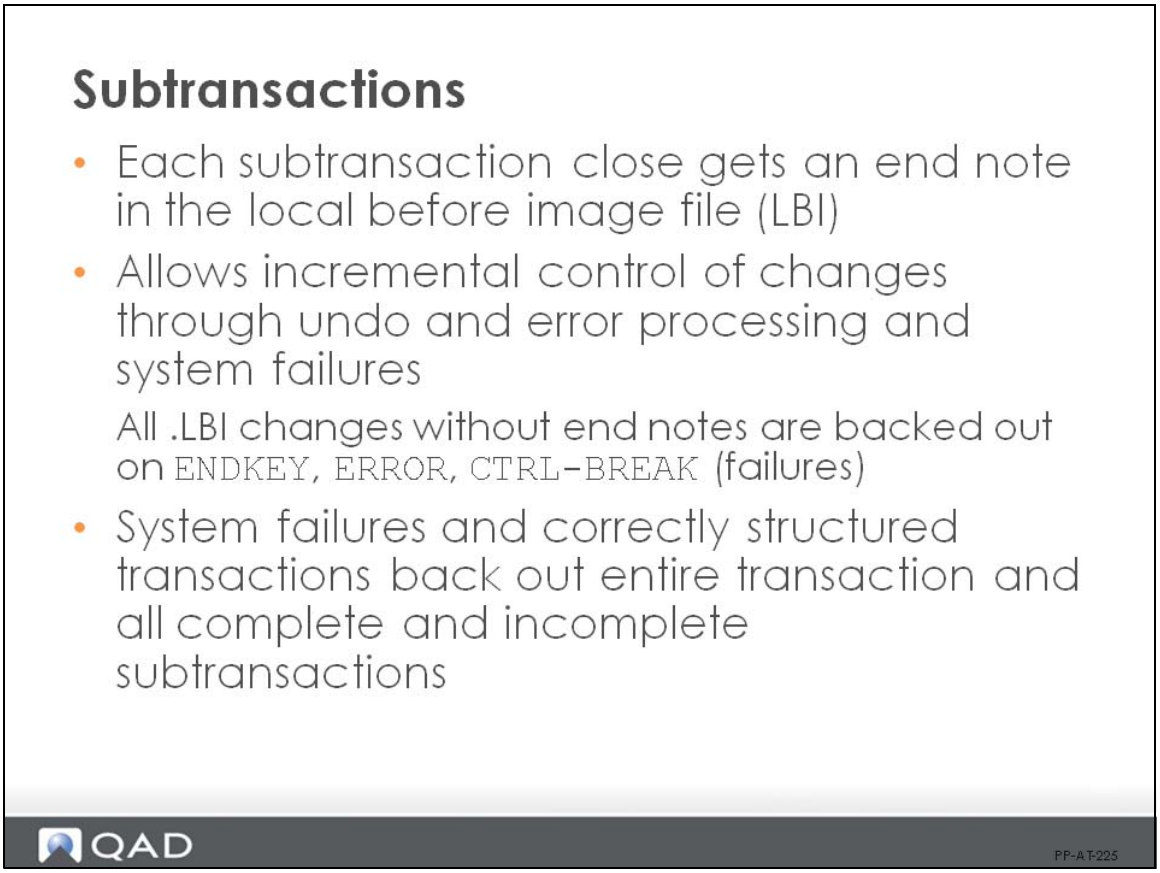
Transaction Guidelines



Subtransactions



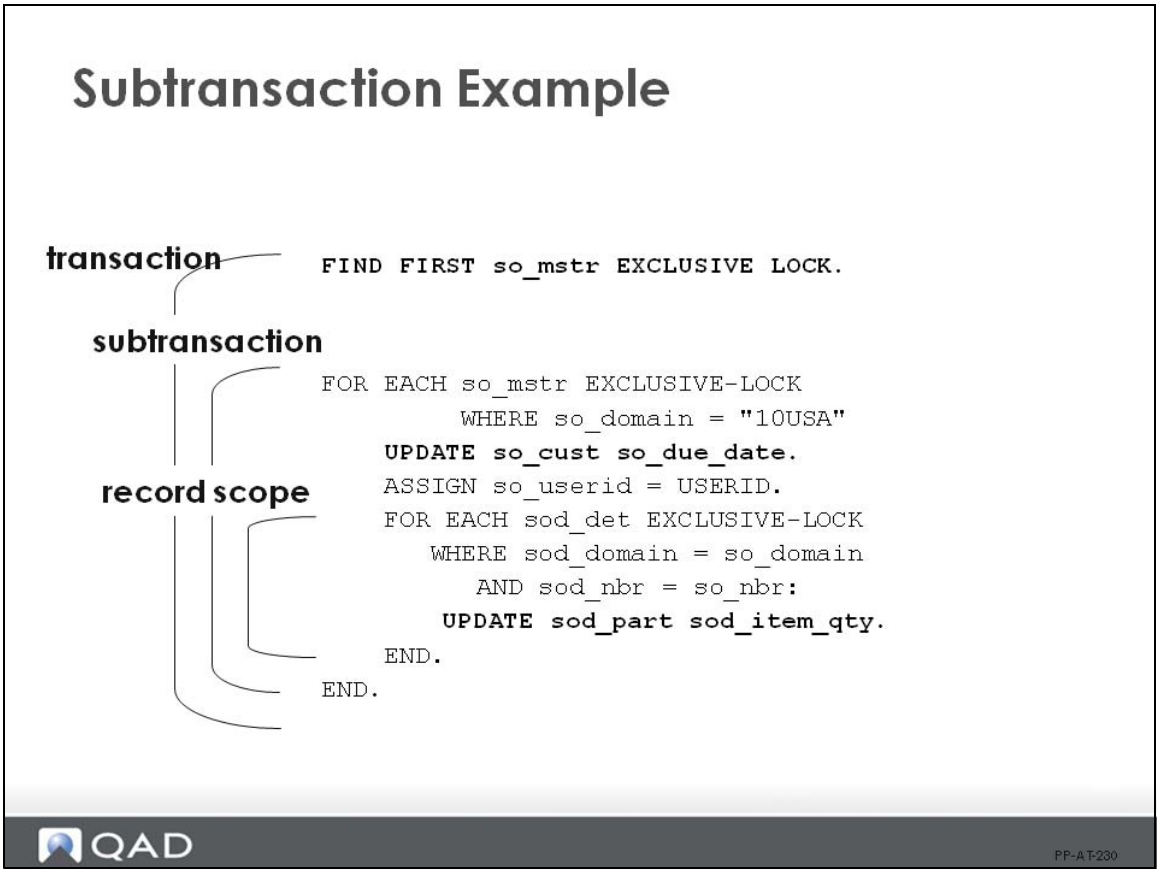
Subtransactions



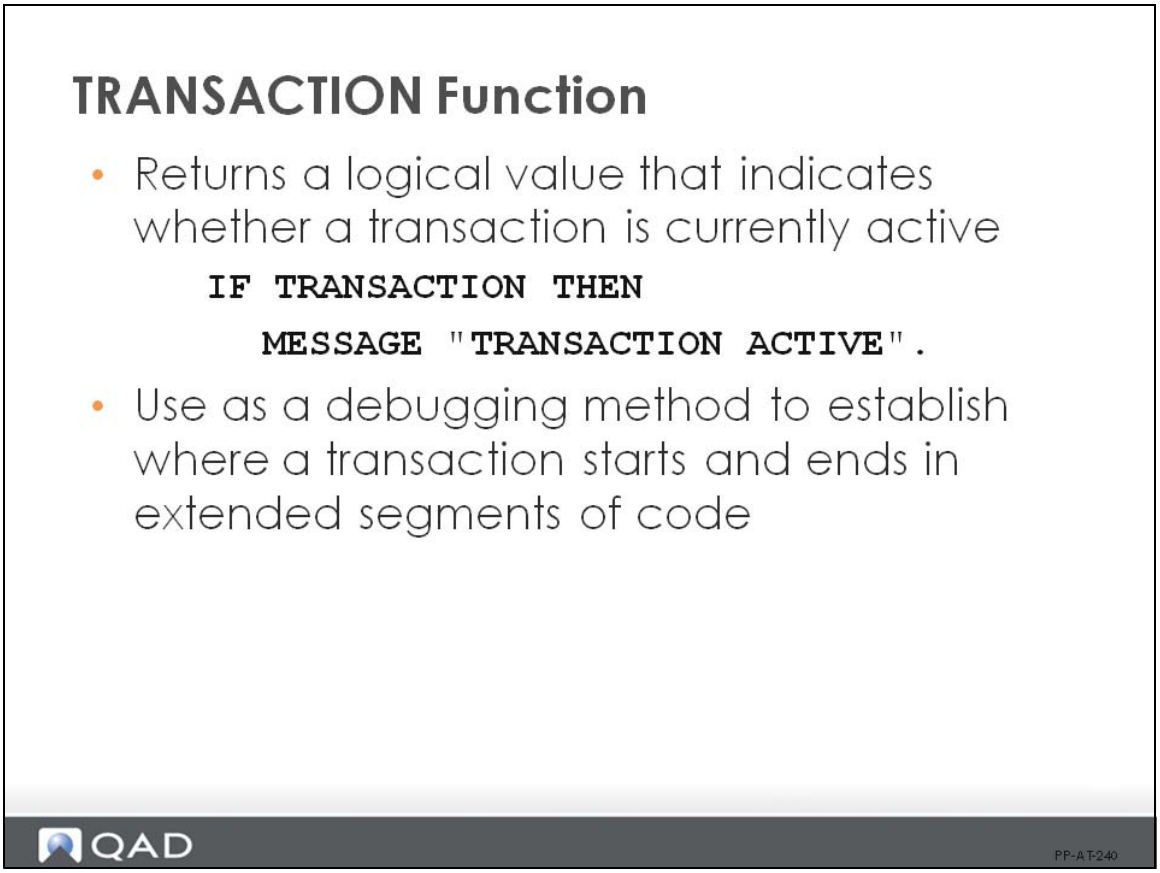
Allow a smaller scope of control within a larger transaction.

On local terminations (Ctrl-C), the sub-transaction is undone. To ensure Data Integrity, the main transaction should then have error processing to continue.

Subtransaction Example

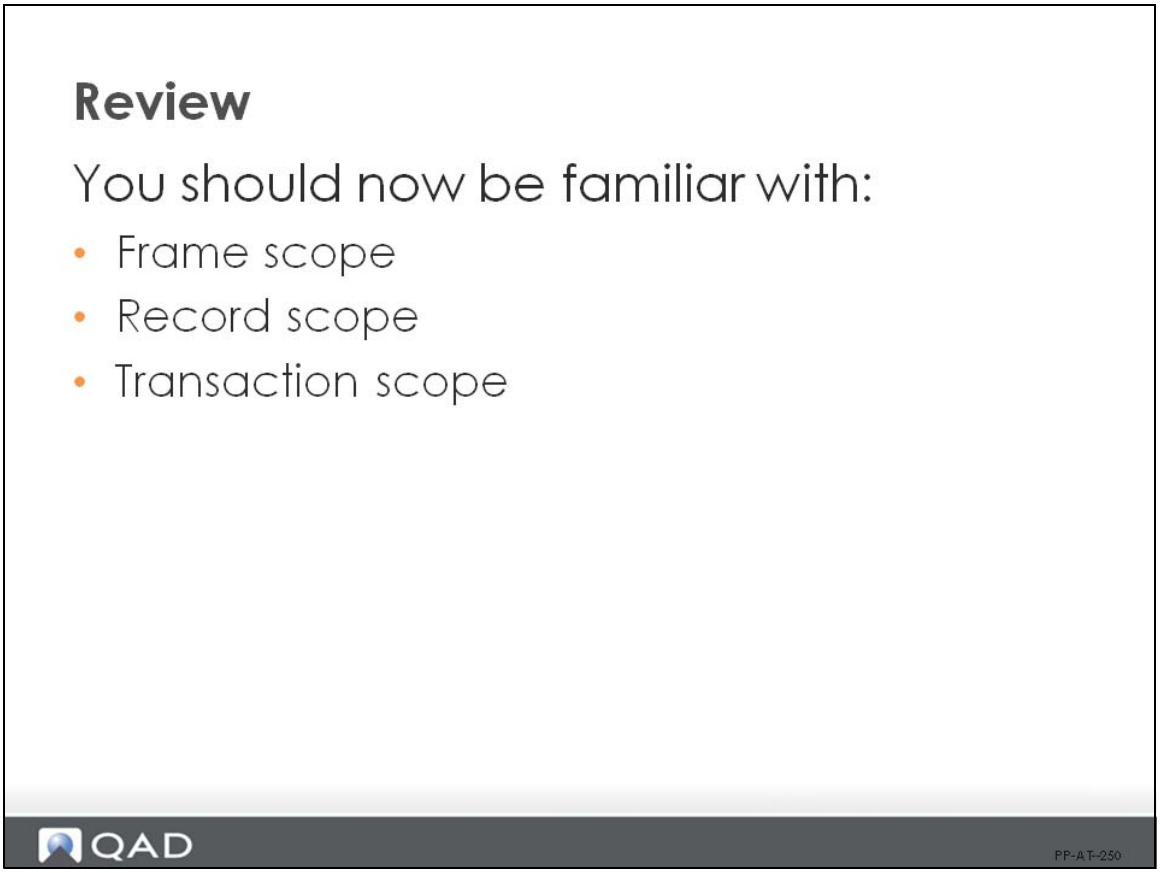


TRANSACTION Function



Note A safer way of determining transaction scope is to compile the program using the XREF option. Then look at the end of the output file to see the transaction scope. This method prevents debugging code from being accidentally left behind in the program.

Review

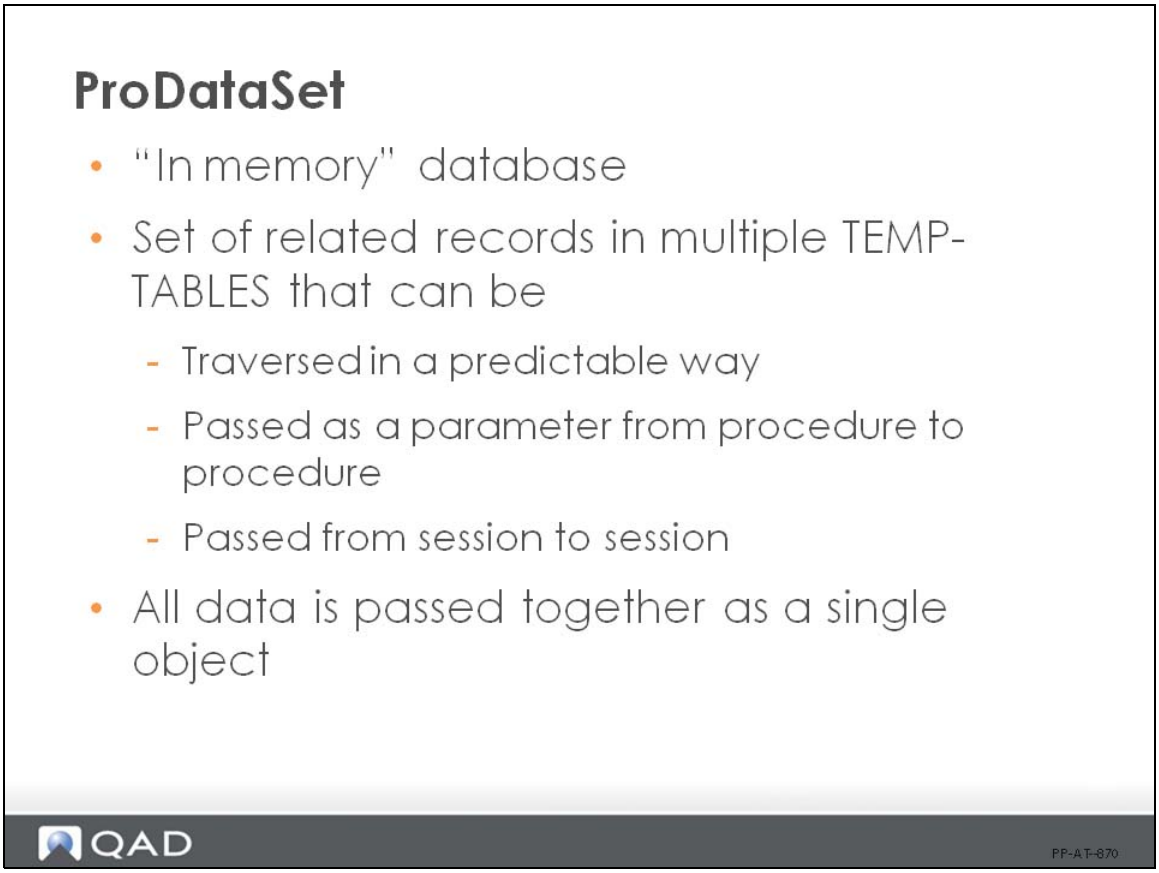

Review
You should now be familiar with:
• Frame scope
• Record scope
• Transaction scope
QAD
PP-AT-250

Lesson 13: ProDataSets

Upon completion of this lesson, you will be familiar with the following topics:

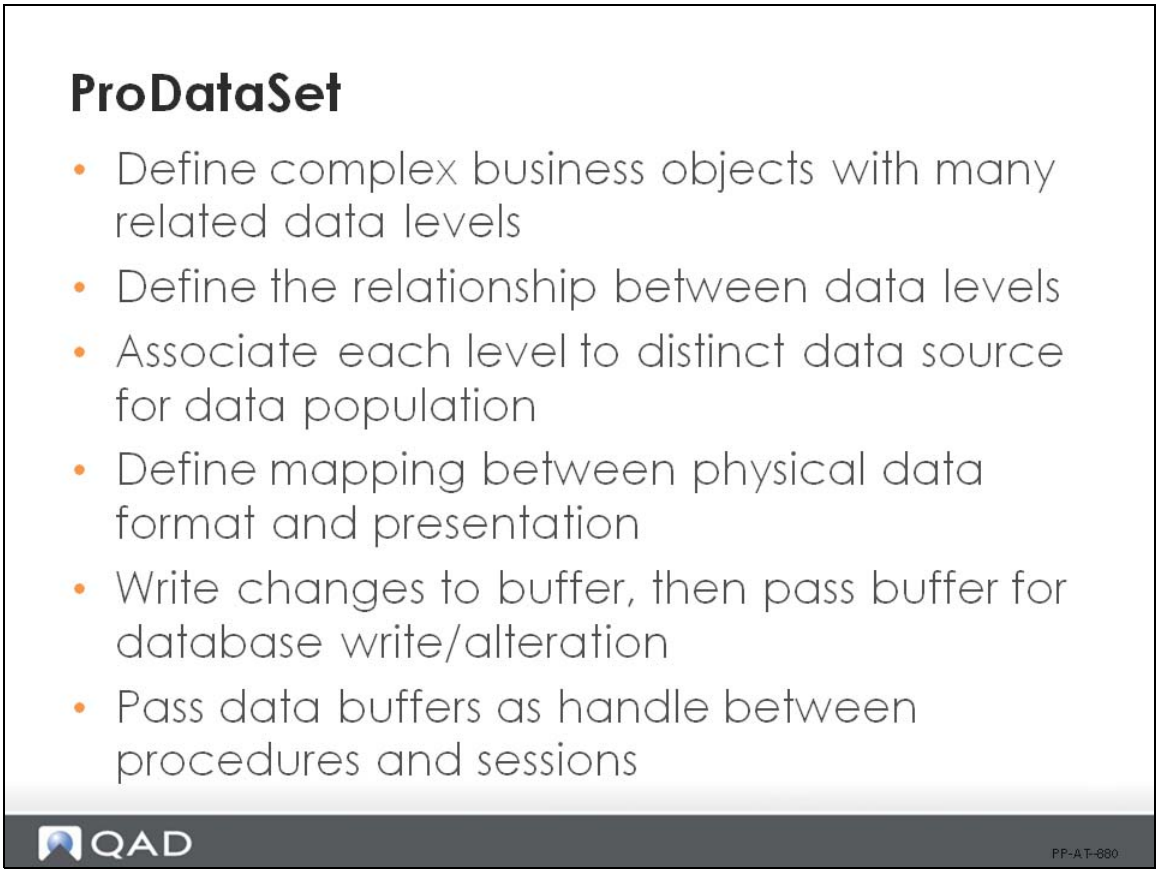
- ProDataSets
- How QAD uses them in Reporting Framework

ProDataSet

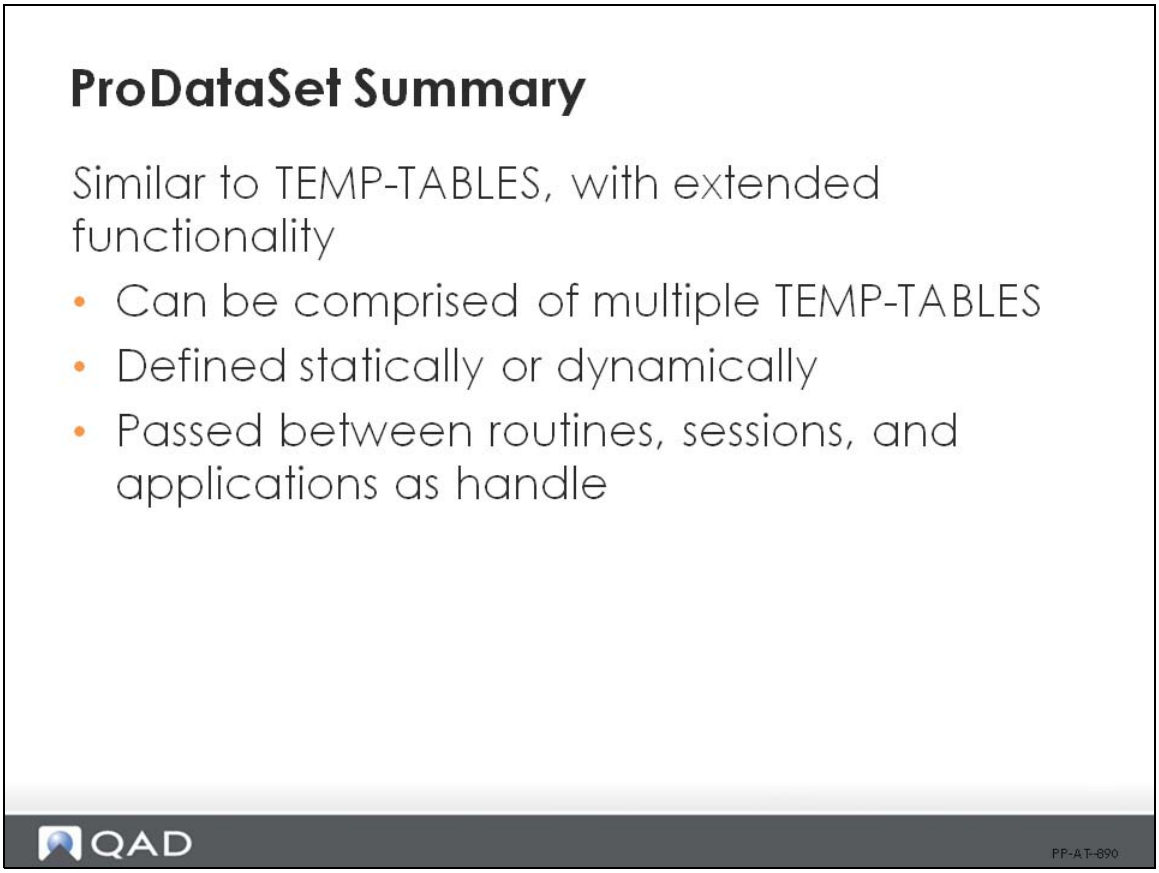


The ProDataSet can be thought of as an “in-memory database” because it is filled with a set of related records in potentially multiple temp-tables that can be traversed in a predictable way. They can also be passed as a single parameter from procedure to procedure or session to session with all of the data being passed together.

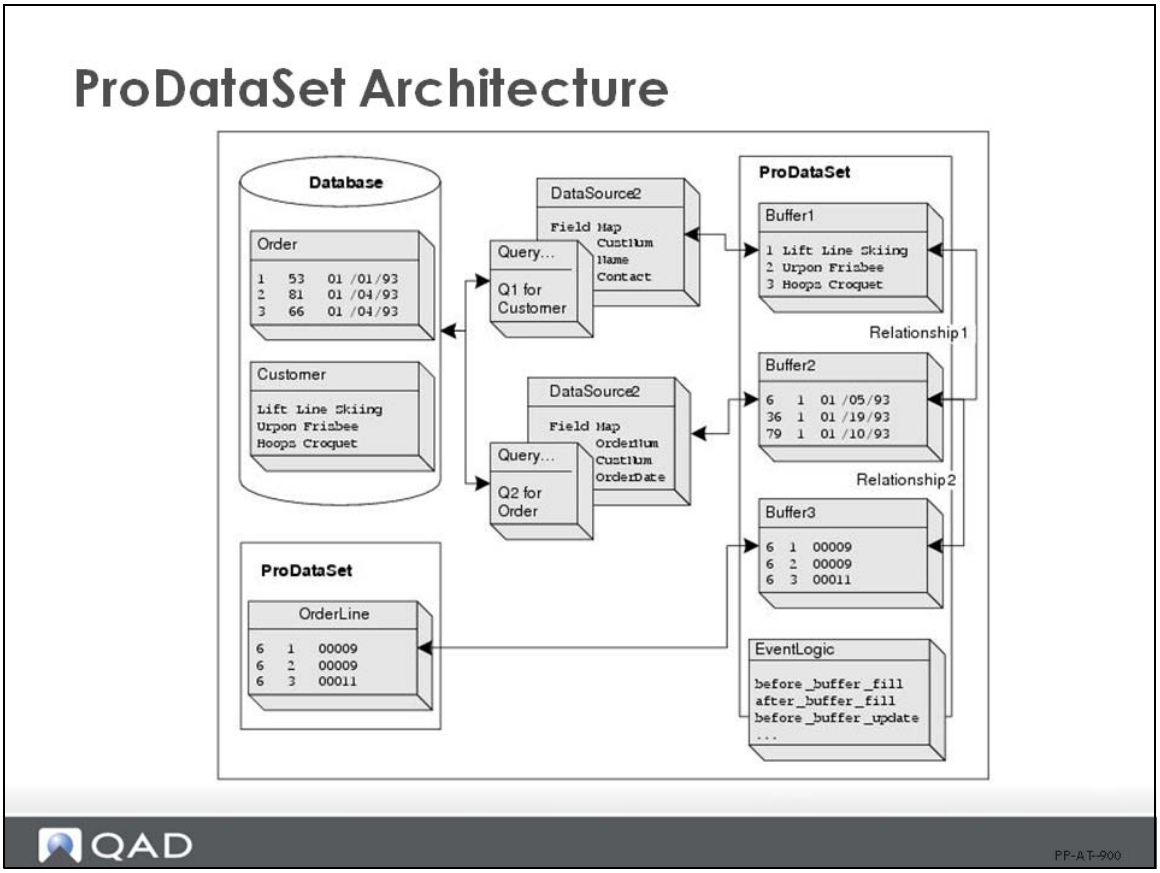
ProDataSet


ProDataSet
Define complex business objects with many related data levels
Define the relationship between data levels
Associate each level to distinct data source for data population
Define mapping between physical data format and presentation
Write changes to buffer, then pass buffer for database write/alteration
Pass data buffers as handle between procedures and sessions
QAD
PP-AT-680

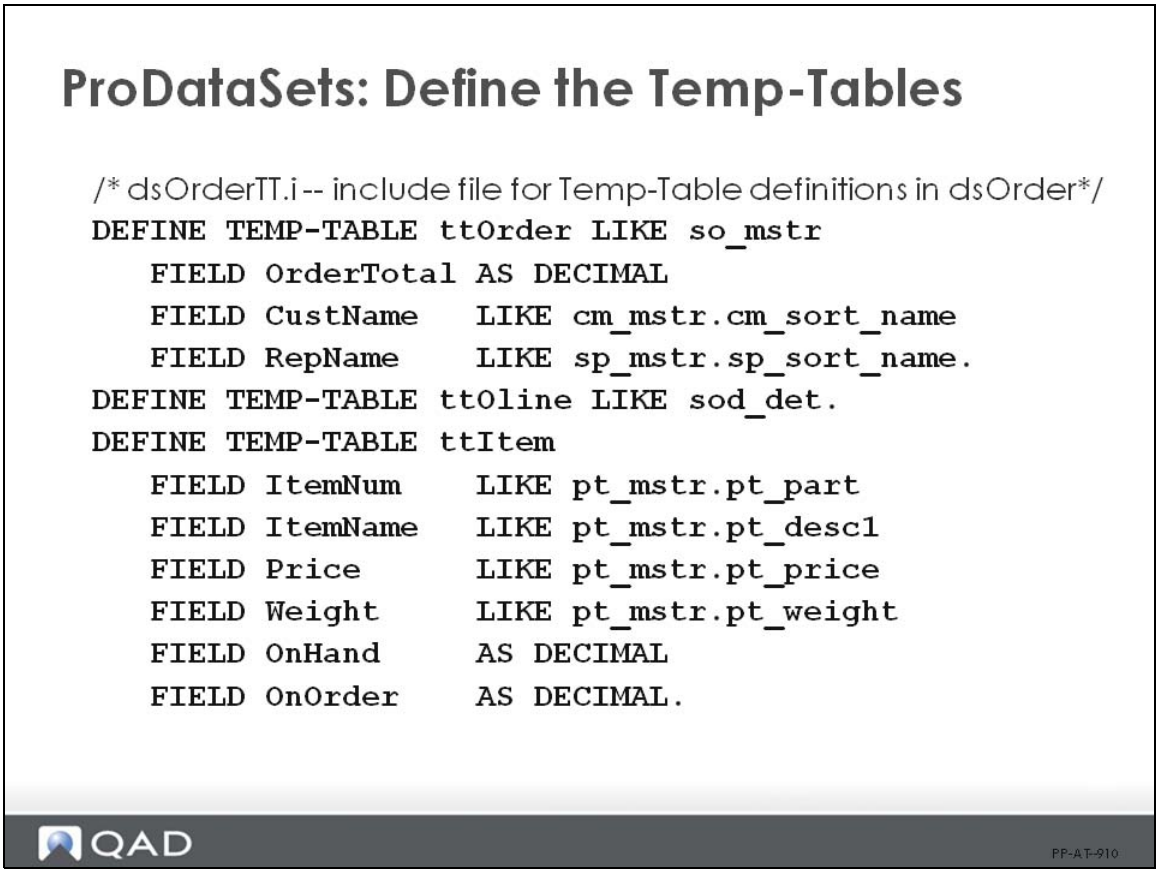
ProDataSet Summary



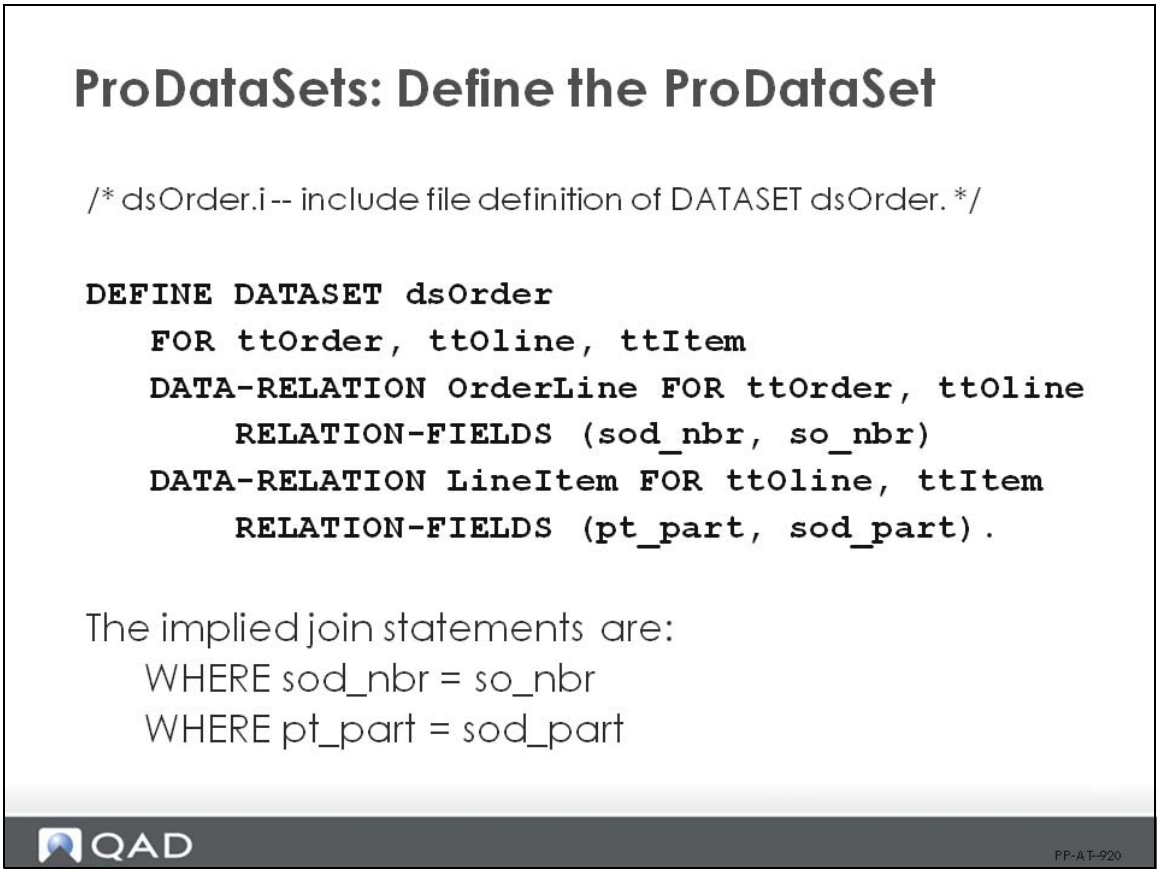
ProDataSet Architecture



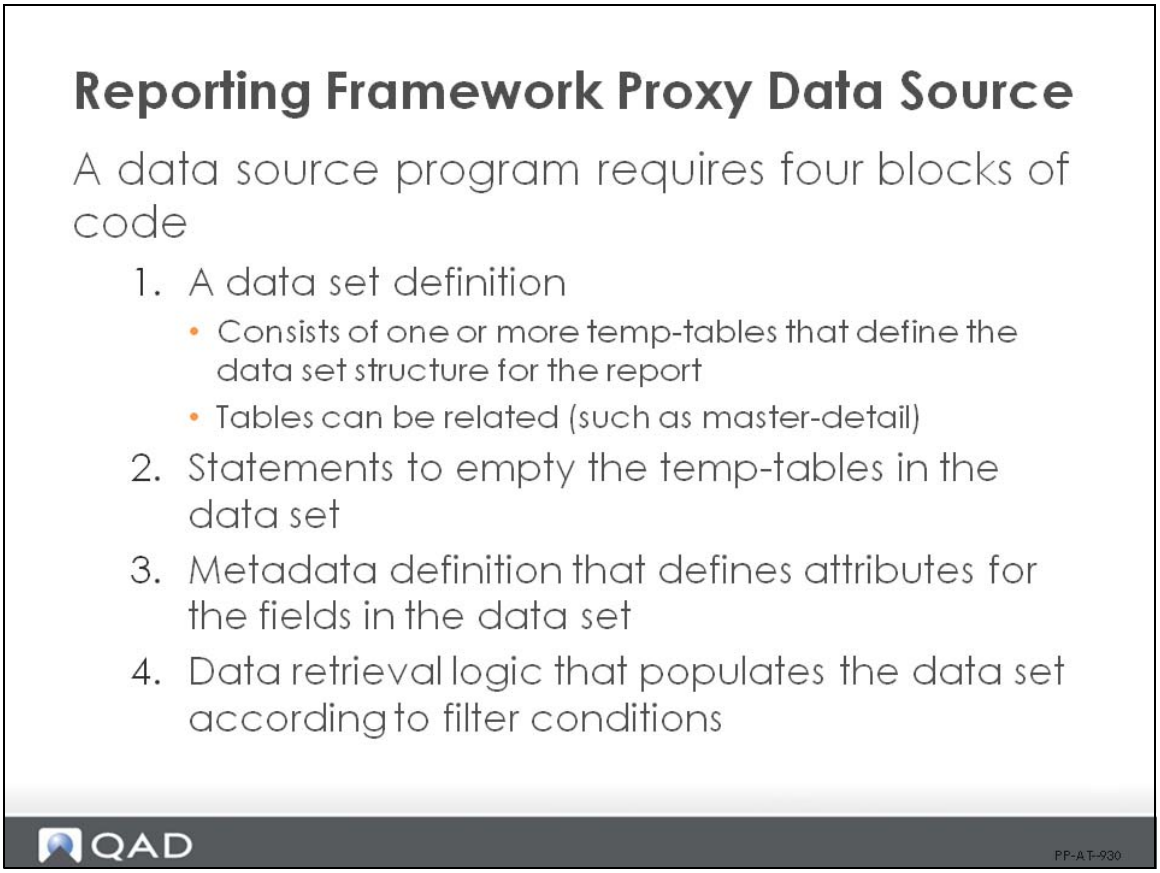
ProDataSets: Define the Temp-Tables



ProDataSets: Define the ProDataSet

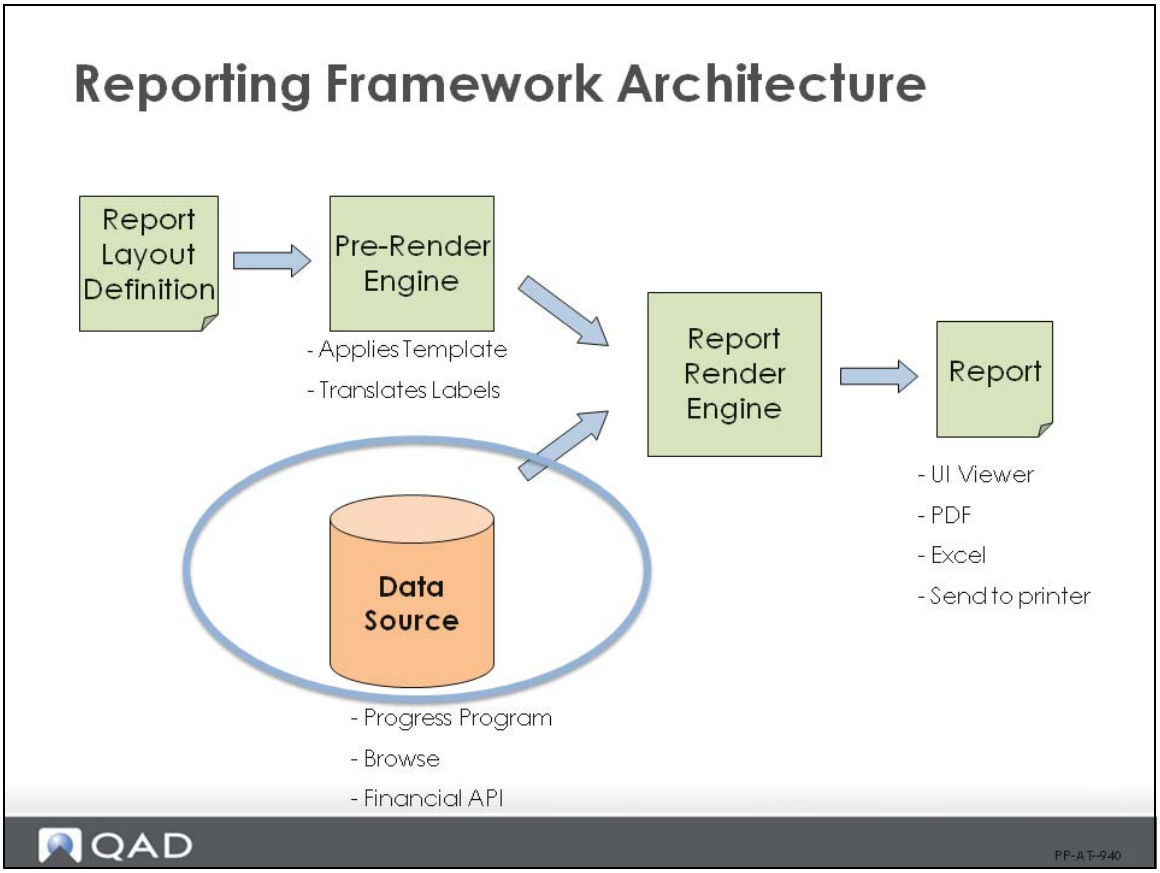


Reporting Framework Proxy Data Source



Writing proxy data source programs is covered in detail in the *Reporting Framework Training Guide*.

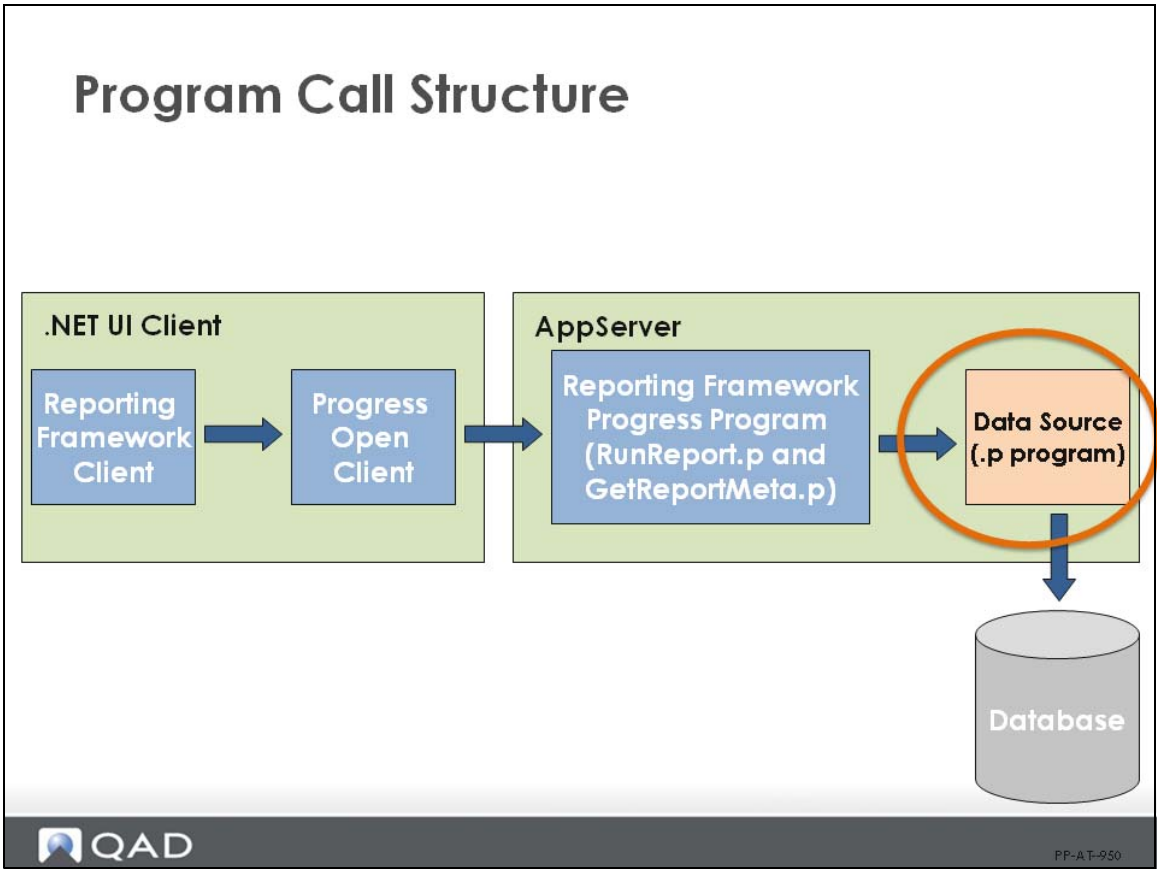
Reporting Framework Architecture



A report can use three types of data sources: Browse, FinancialAPI, and Progress program.

Note FinancialAPI only applies to QAD Enterprise Edition, not QAD Standard Edition.

Program Call Structure



This diagram illustrates the context in which a Progress data source program is run.

Review


Review
You should now be familiar with:
• ProDataSets
• How QAD uses them in Reporting Framework
QAD
PP-AT-960

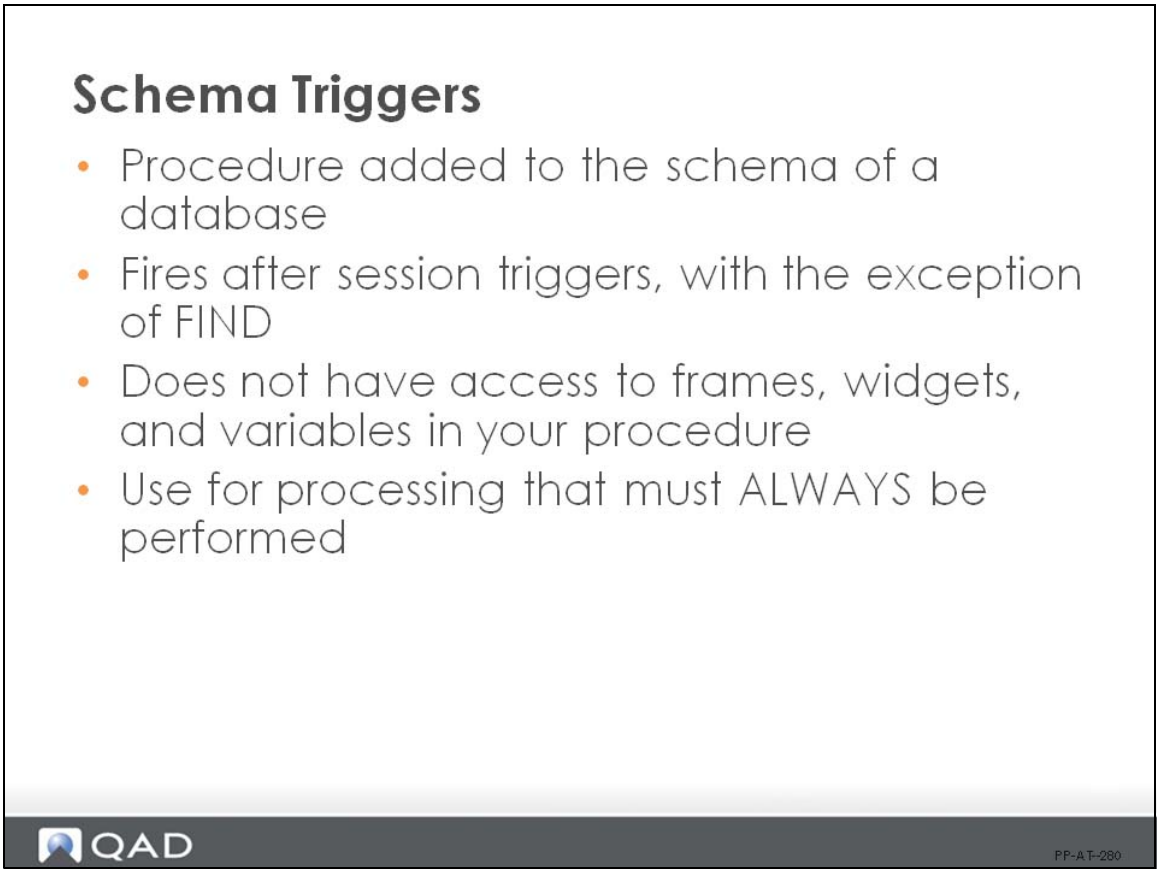
Lesson 14: Triggers and Events (Optional)

Upon completion of this lesson, you will be familiar with the following topics:

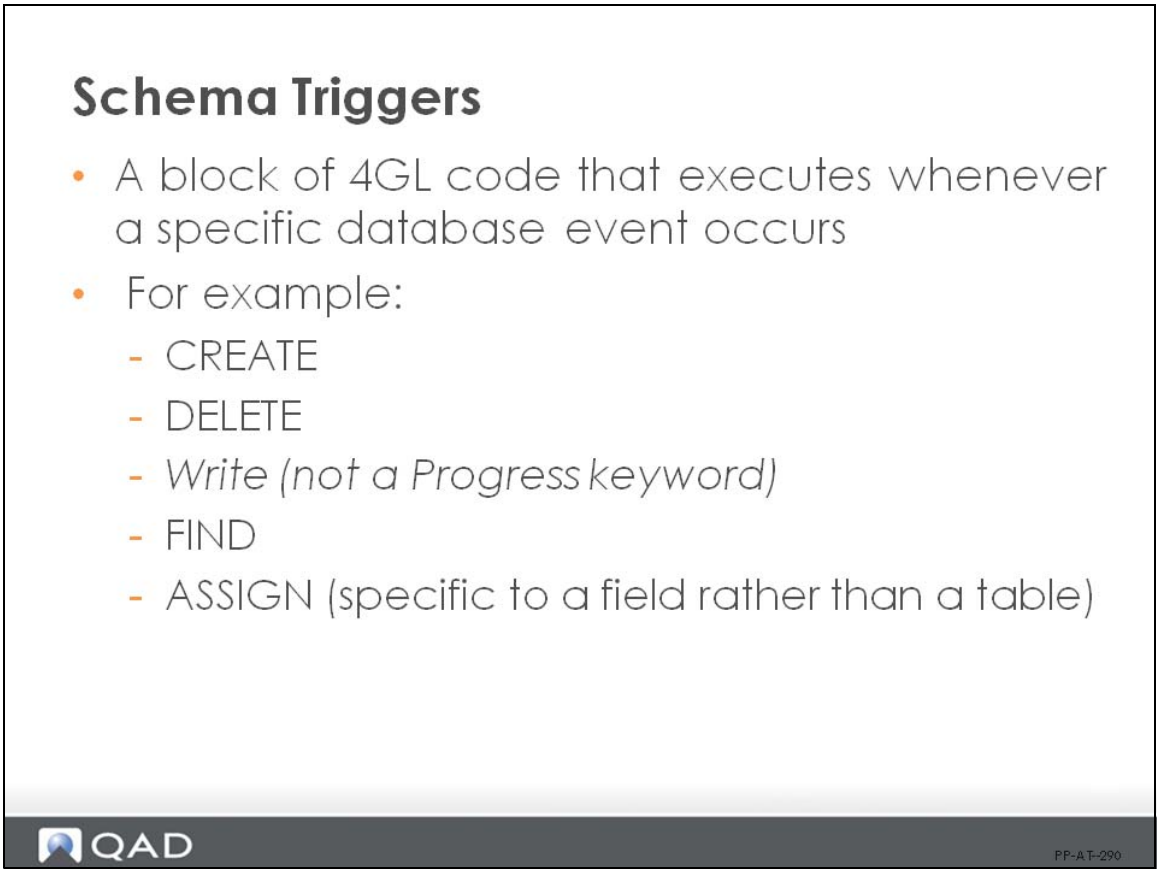
- Schema triggers
- Session triggers and events
- Modal vs. modeless programming
- Event-driven structure
  - WAIT-FOR
  - ENABLE
  - DISABLE

Note In general, QAD does not use triggers as a standard coding practice. However, there are useful applications for this technology, including QAD's Integrated Customization Toolkit (ICT).

Schema Triggers



Schema Triggers



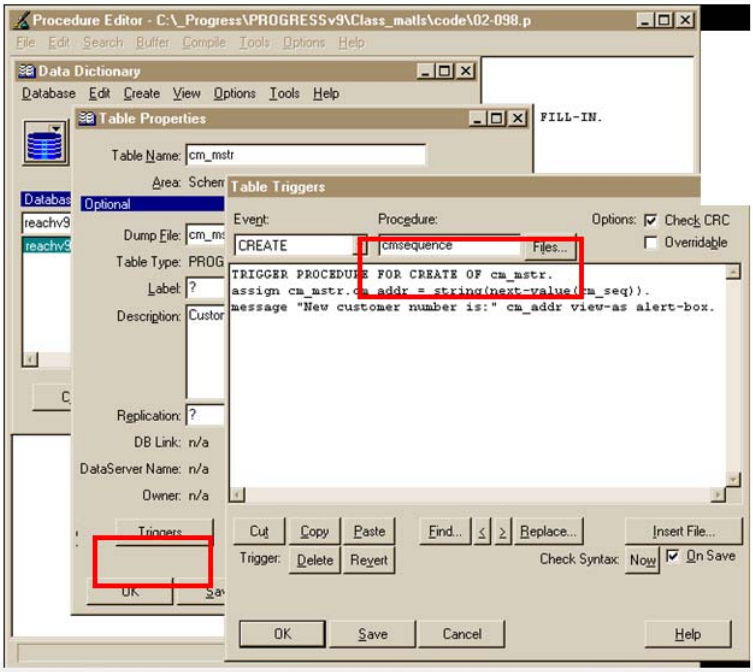
A database event is an action performed against a database.

Schema Triggers

Schema Trigger Example

Triggers are written and stored in the Data Dictionary for the table or field they apply to

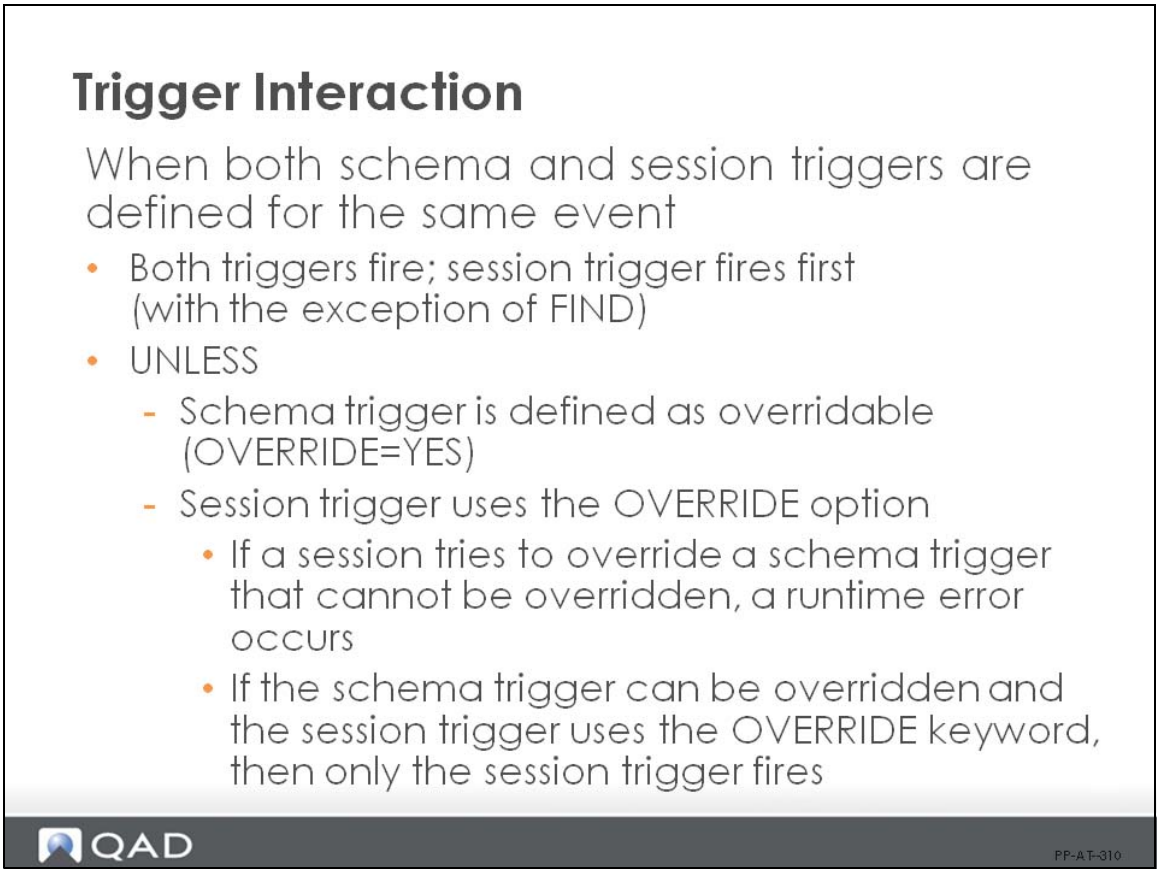
Triggers must have a file name



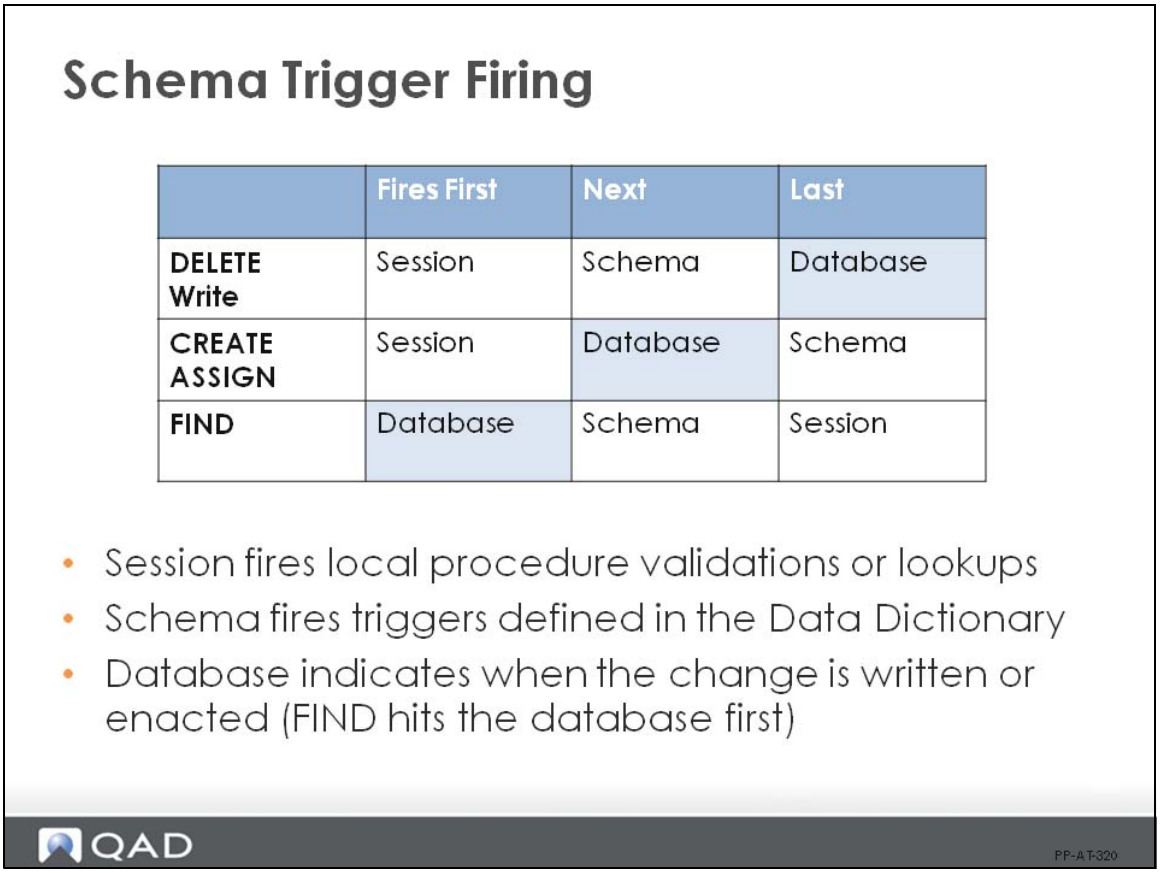
QAD

PP-AT-300

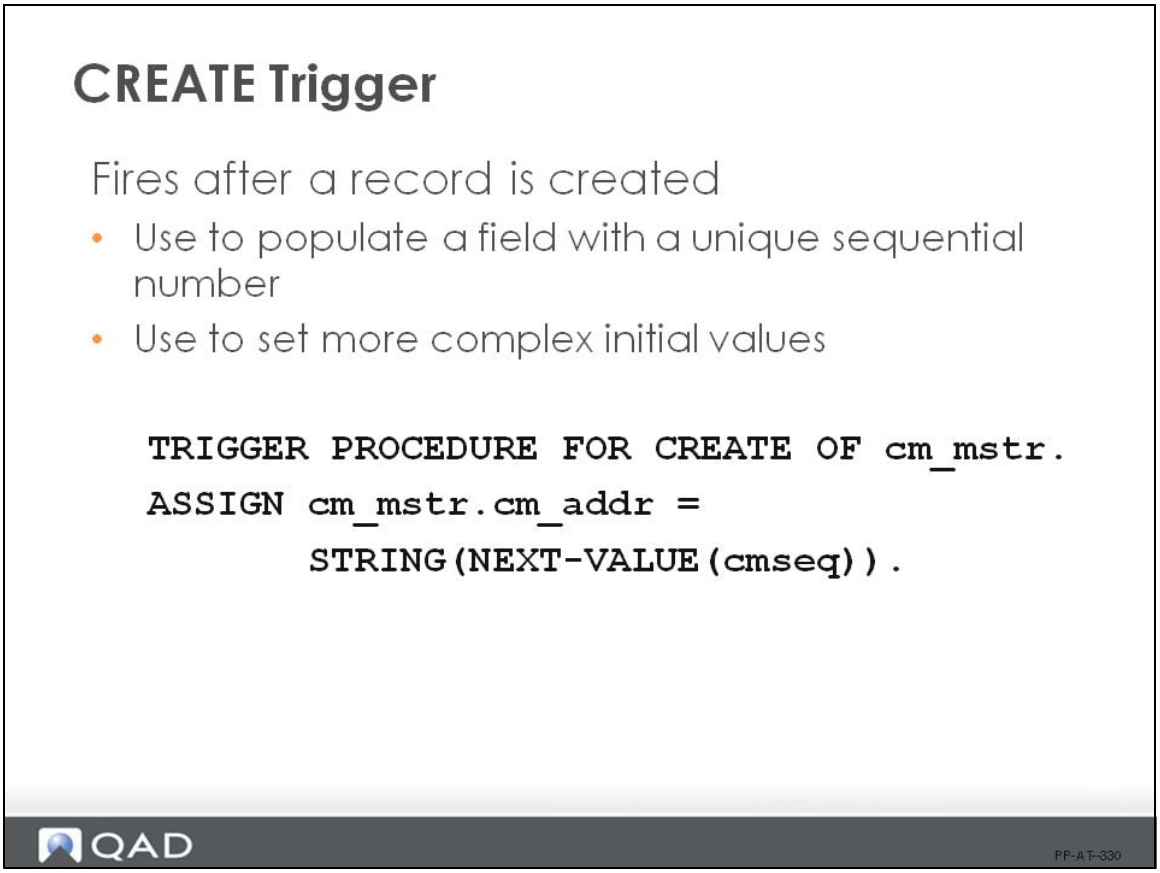
Trigger Interaction



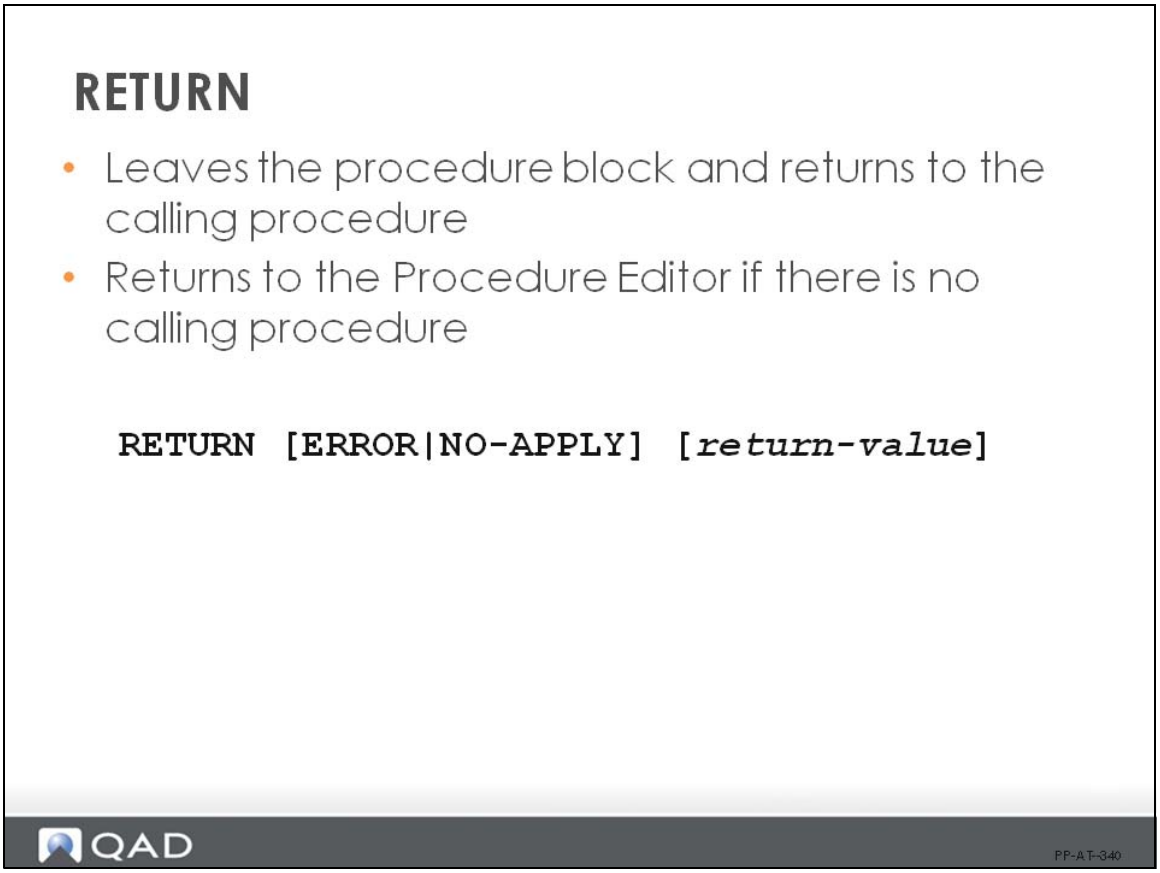
Schema Trigger Firing



CREATE Trigger



RETURN


RETURN
• Leaves the procedure block and returns to the calling procedure
• Returns to the Procedure Editor if there is no calling procedure
RETURN [ERROR|NO-APPLY] [return-value]
QAD
PP-AT-340

RETURN-VALUE Function

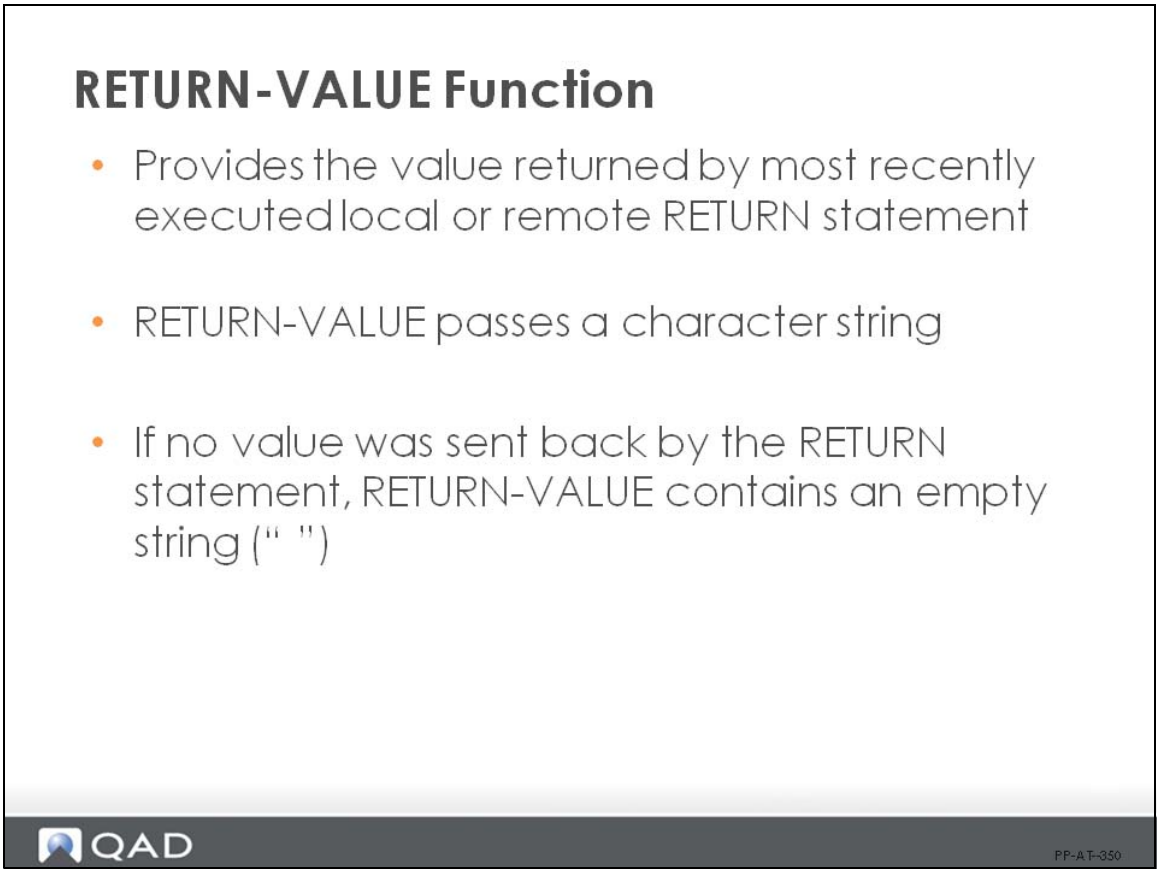
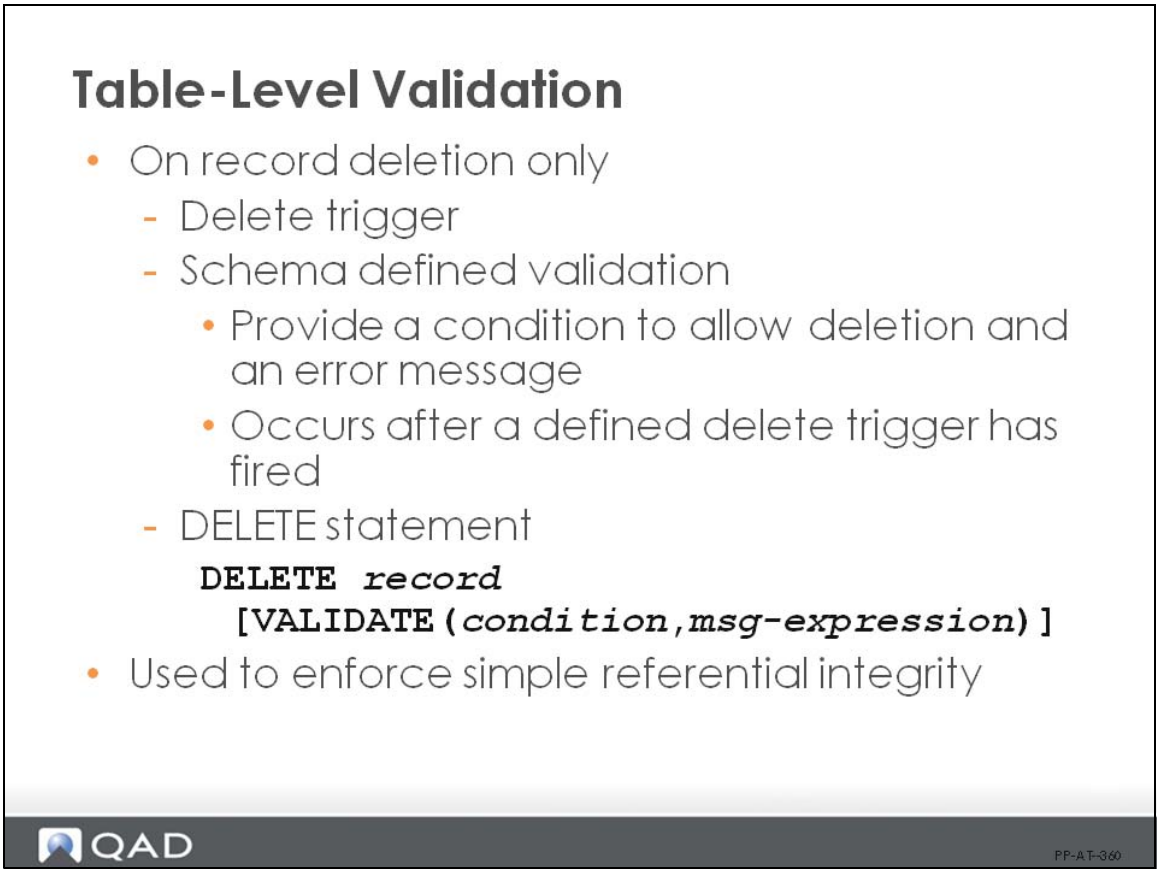
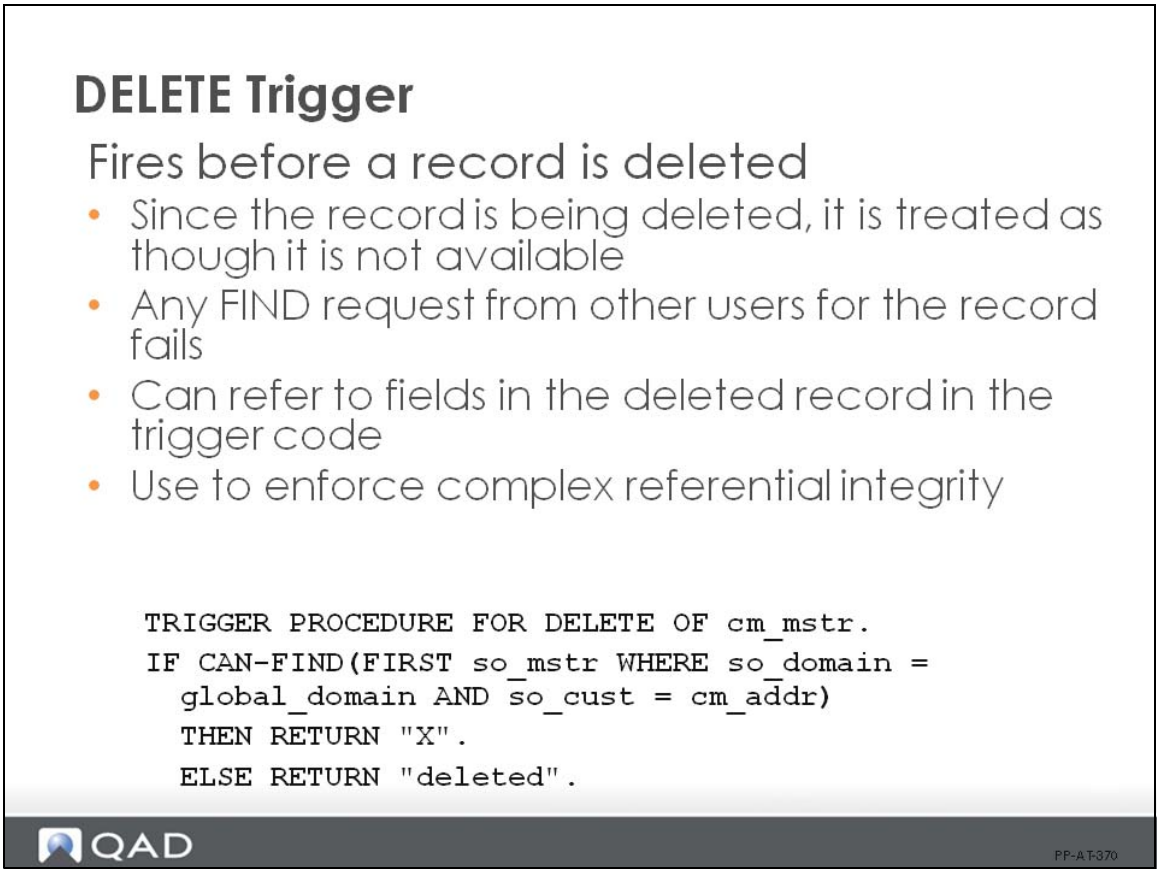


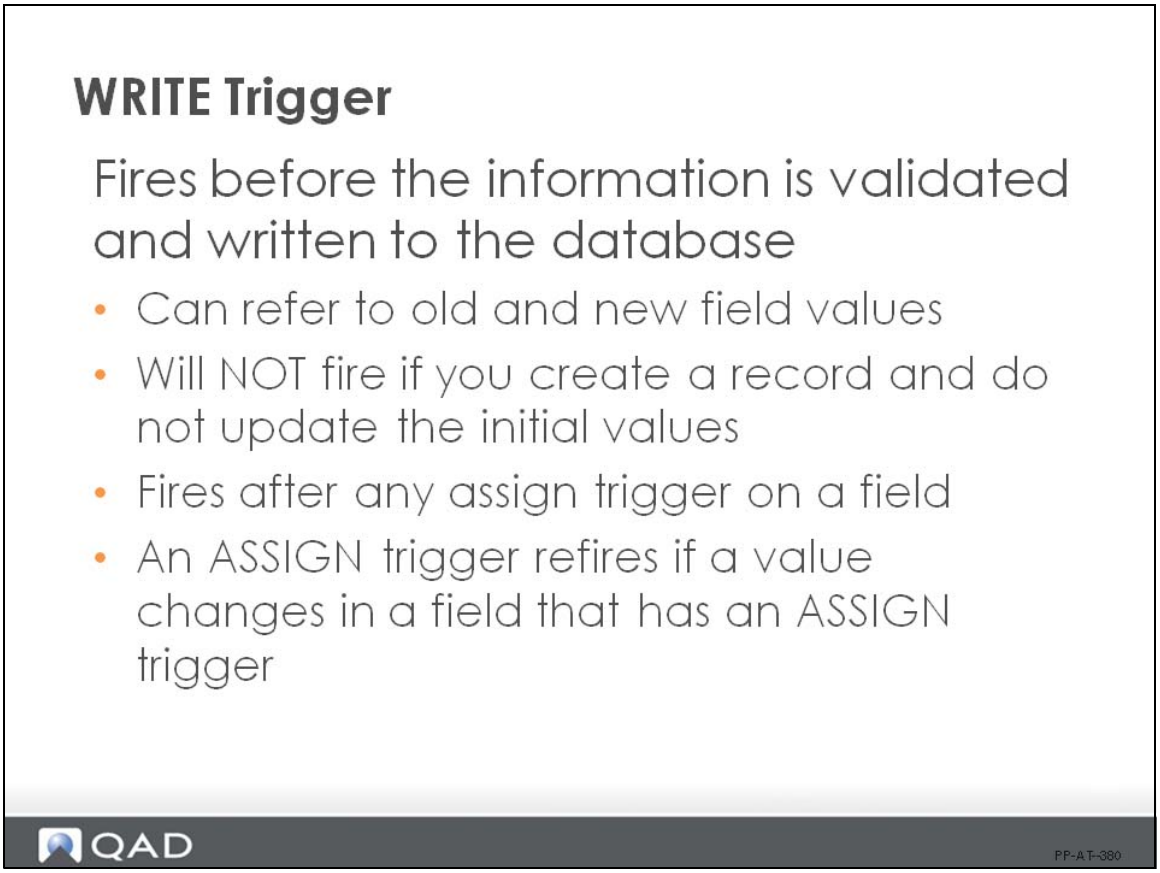
Table-Level Validation



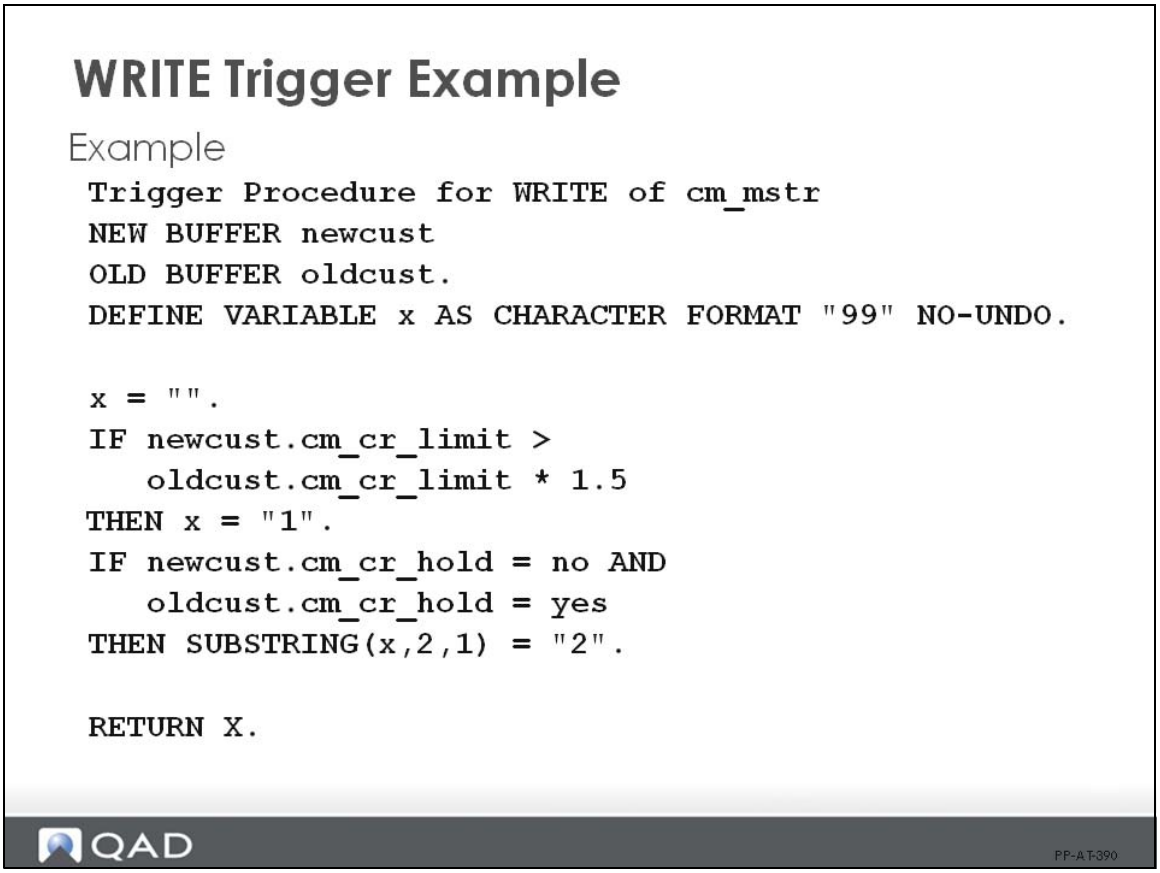
DELETE Trigger



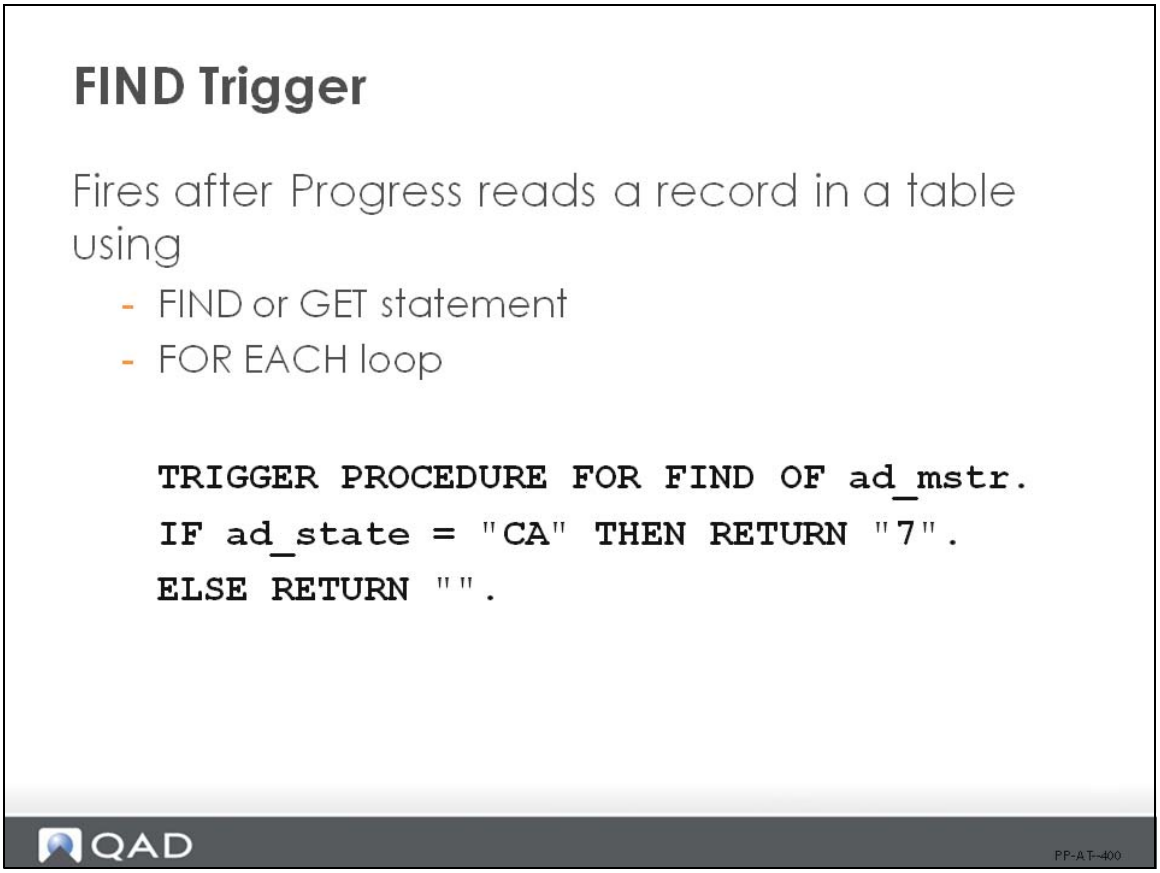
WRITE Trigger



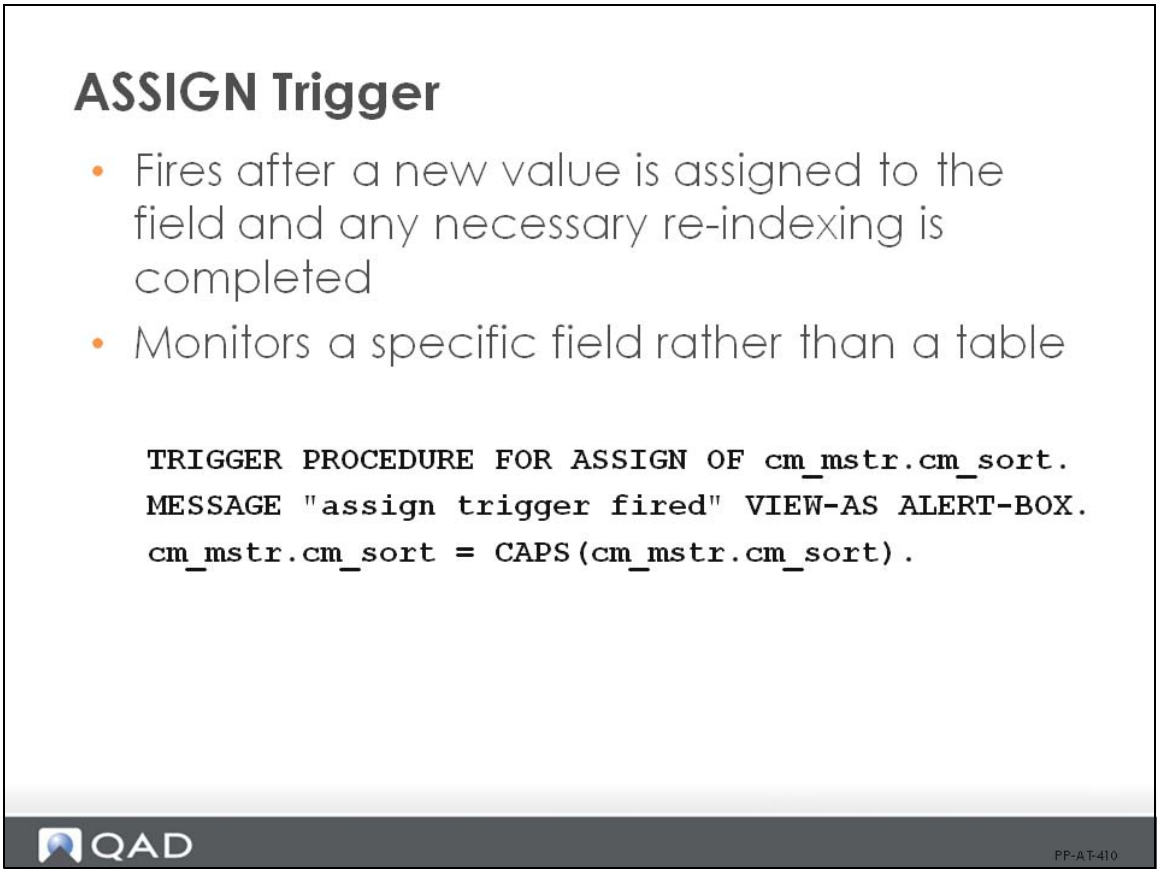
WRITE Trigger Example



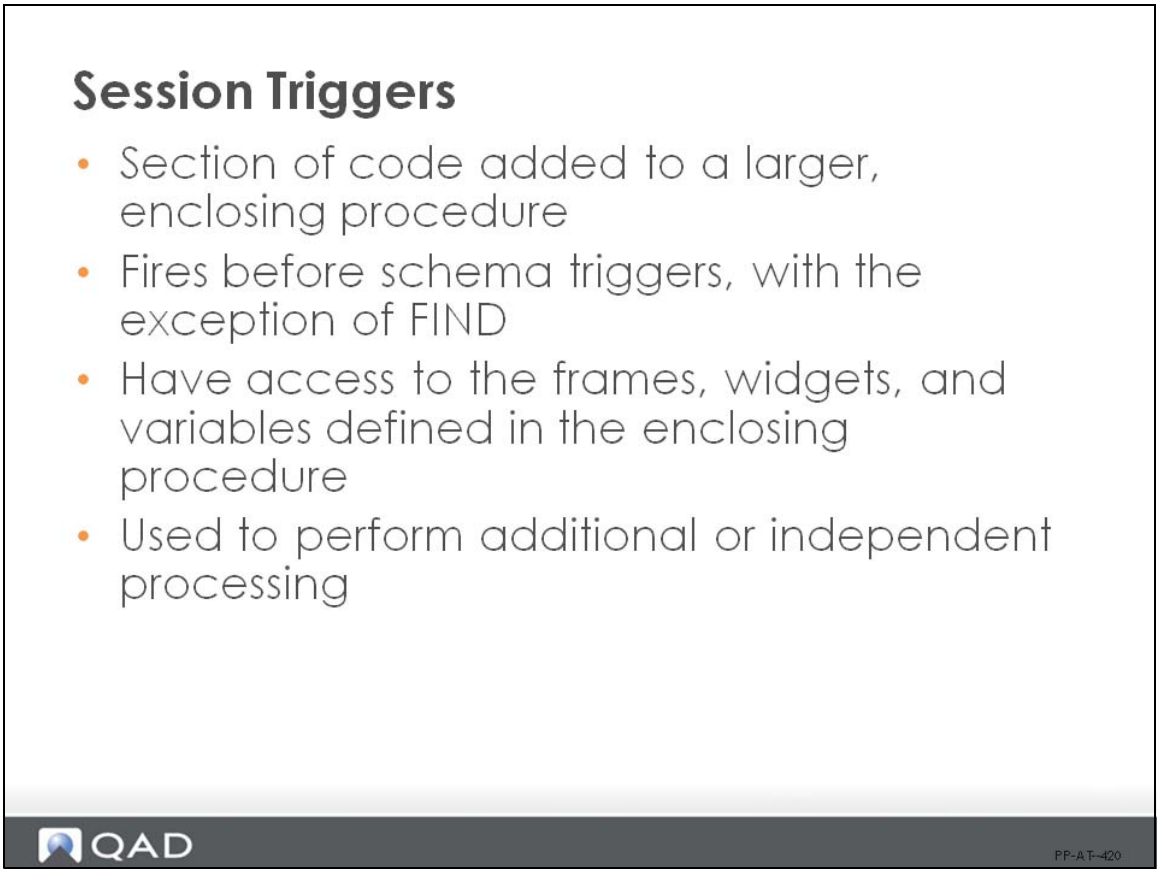
FIND Trigger



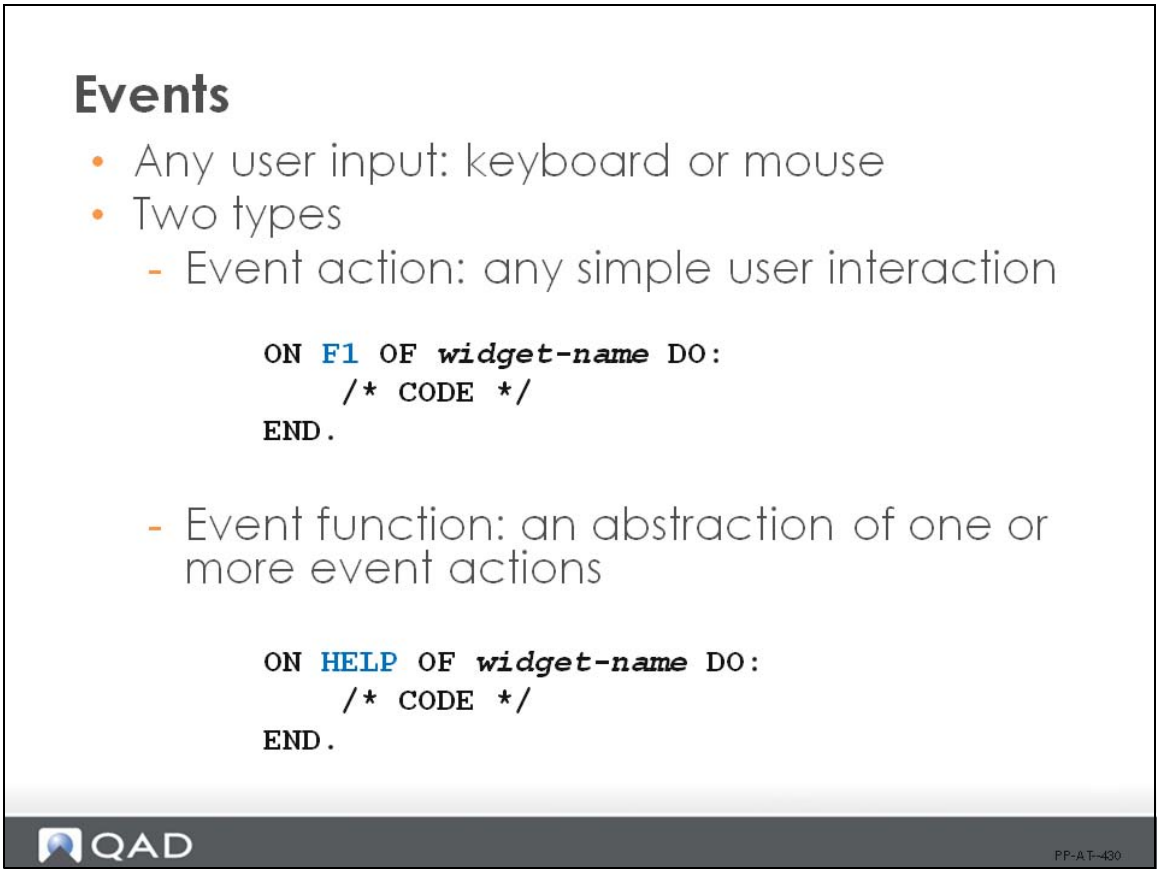
ASSIGN Trigger



Session Triggers

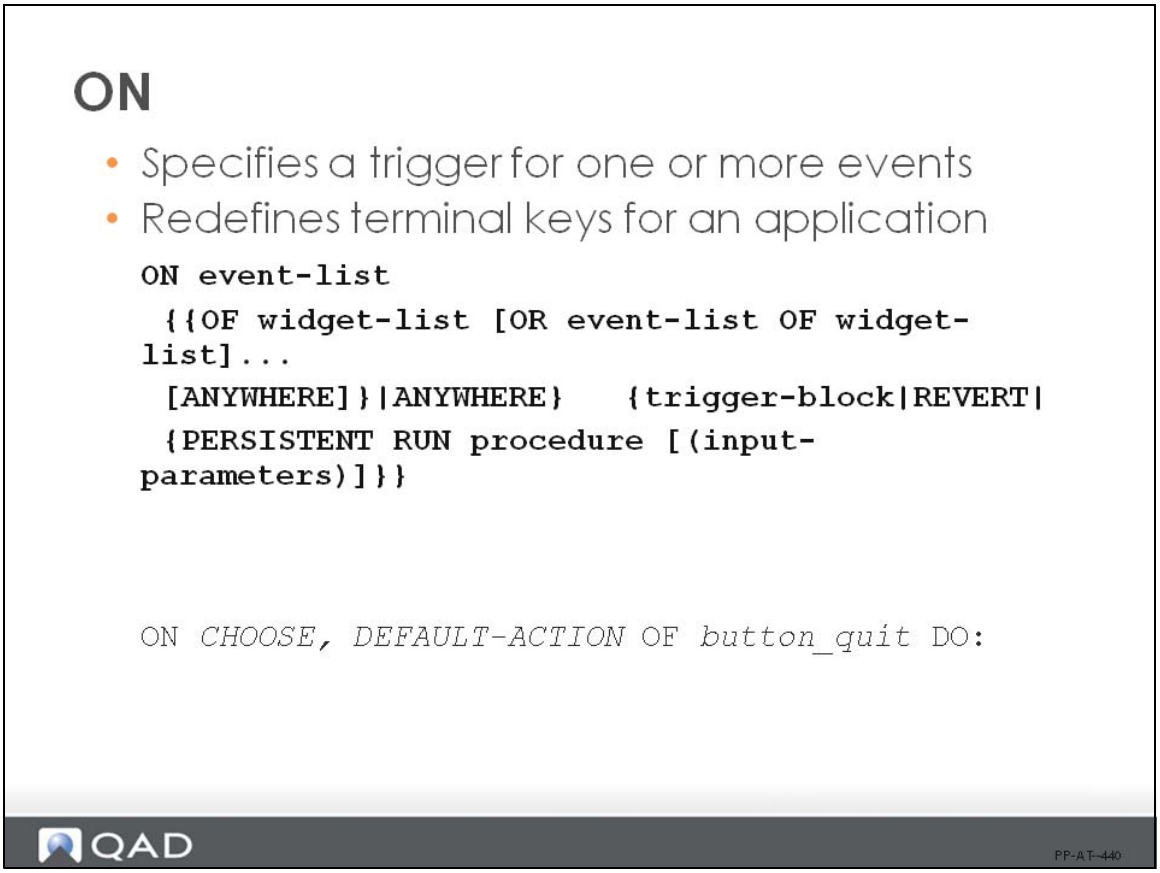


Events

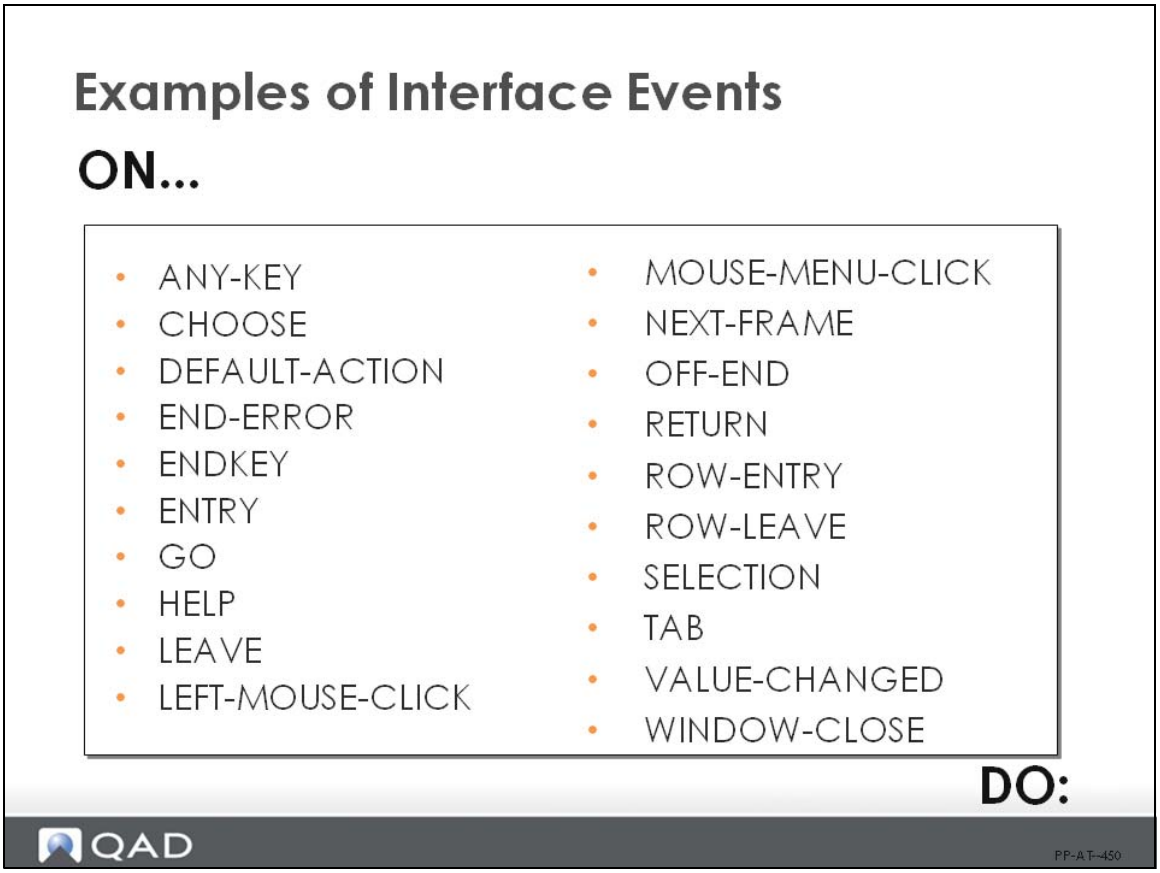


You can set up triggers for raw keystrokes or for key functions. This approach leads to much cleaner code and is flexible and well suited for event-driven code.

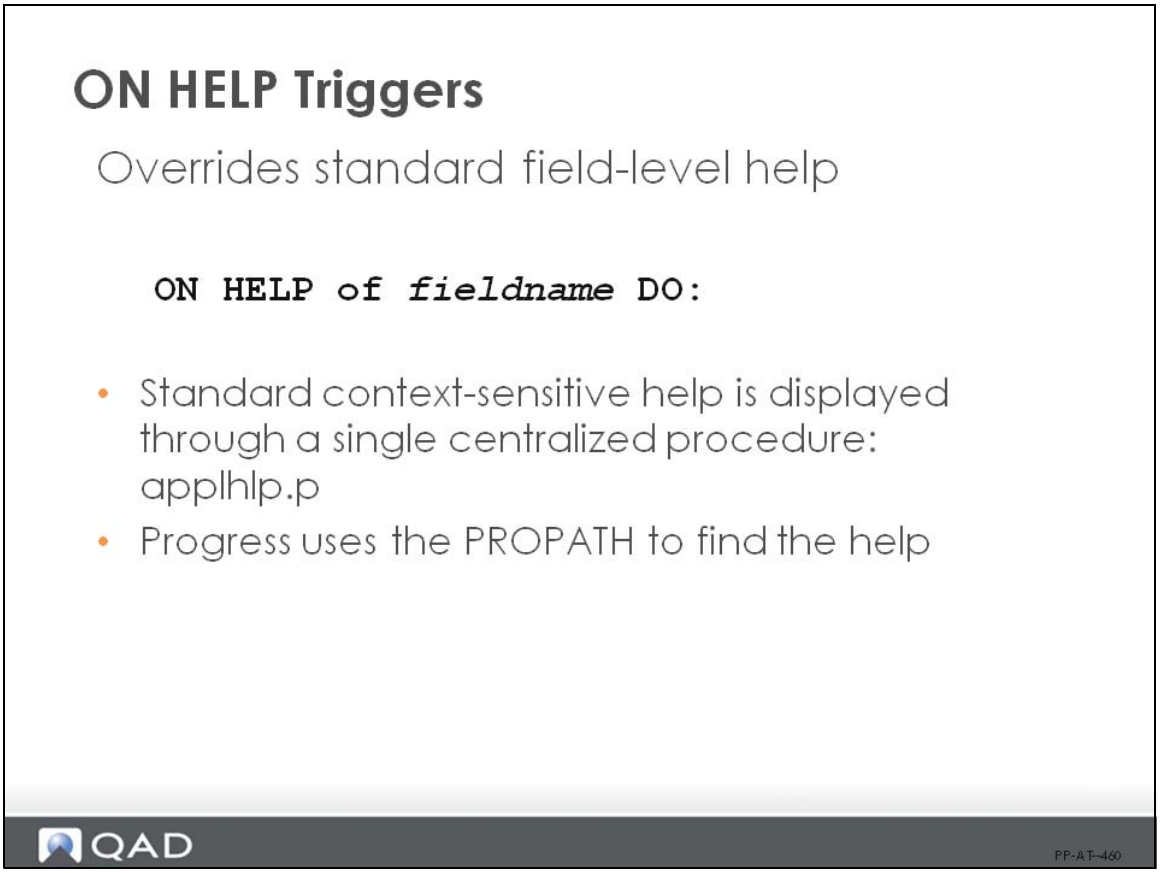
ON


ON
Specifies a trigger for one or more events
Redefines terminal keys for an application
ON event-list
{{OF widget-list [OR event-list OF widget-list]...
[ANYWHERE]}|ANYWHERE} {trigger-block|REVERT|
{PERSISTENT RUN procedure [(input-parameters)]}}
ON CHOOSE, DEFAULT-ACTION OF button_quit DO:
QAD
PP-AT-440

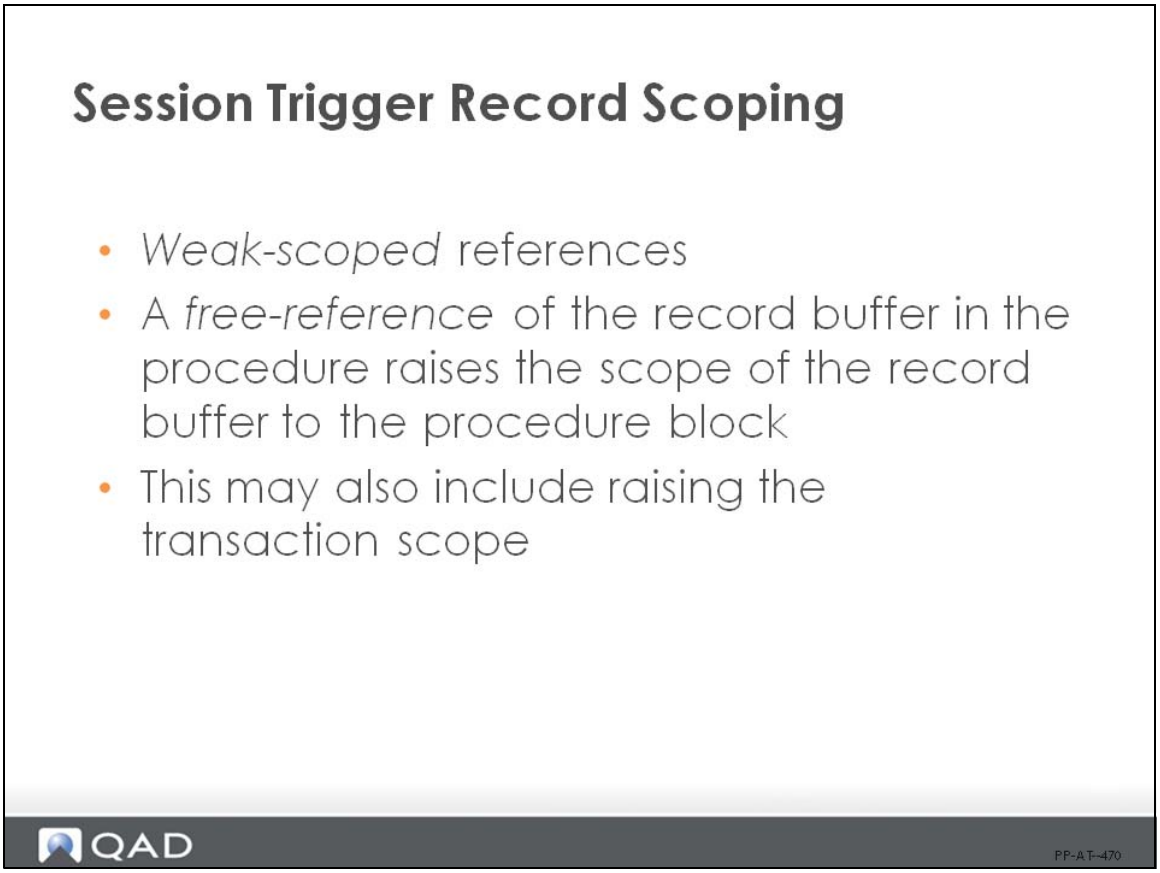
Examples of Interface Events



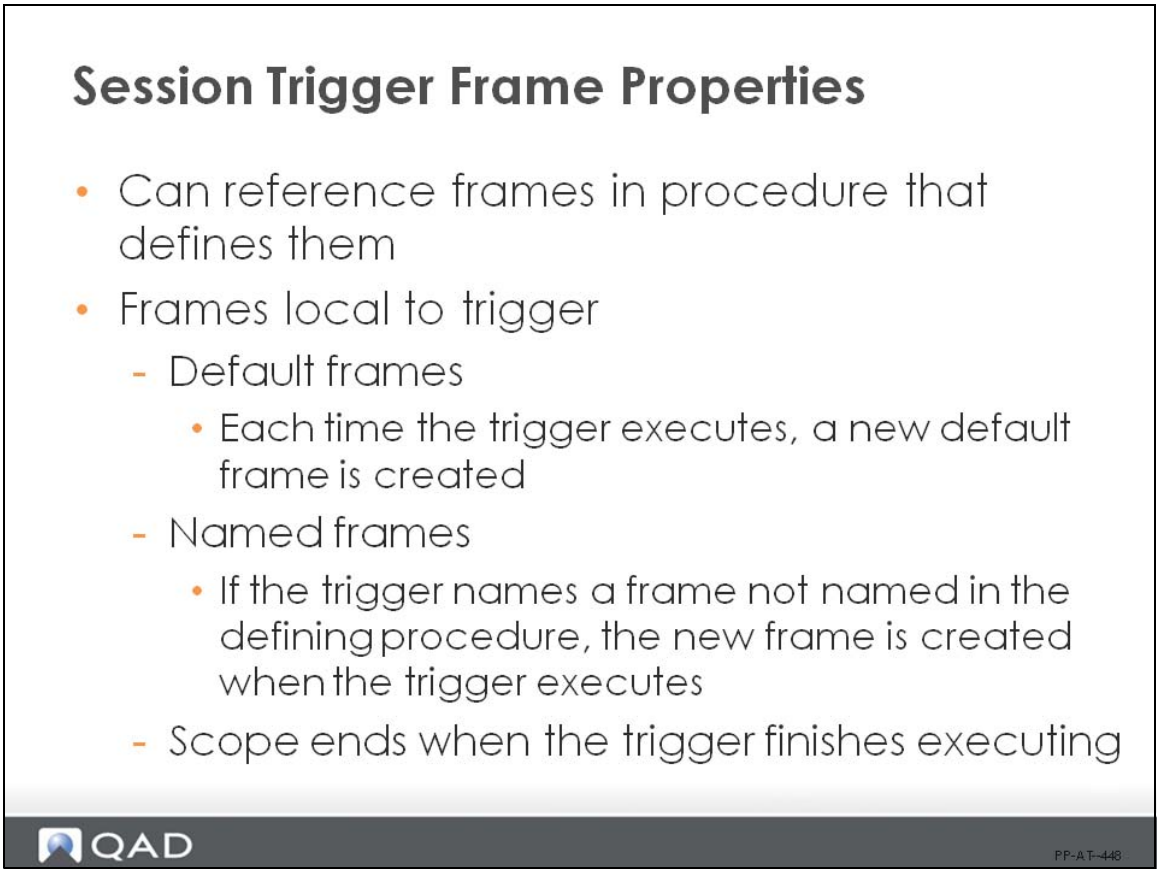
ON HELP Triggers



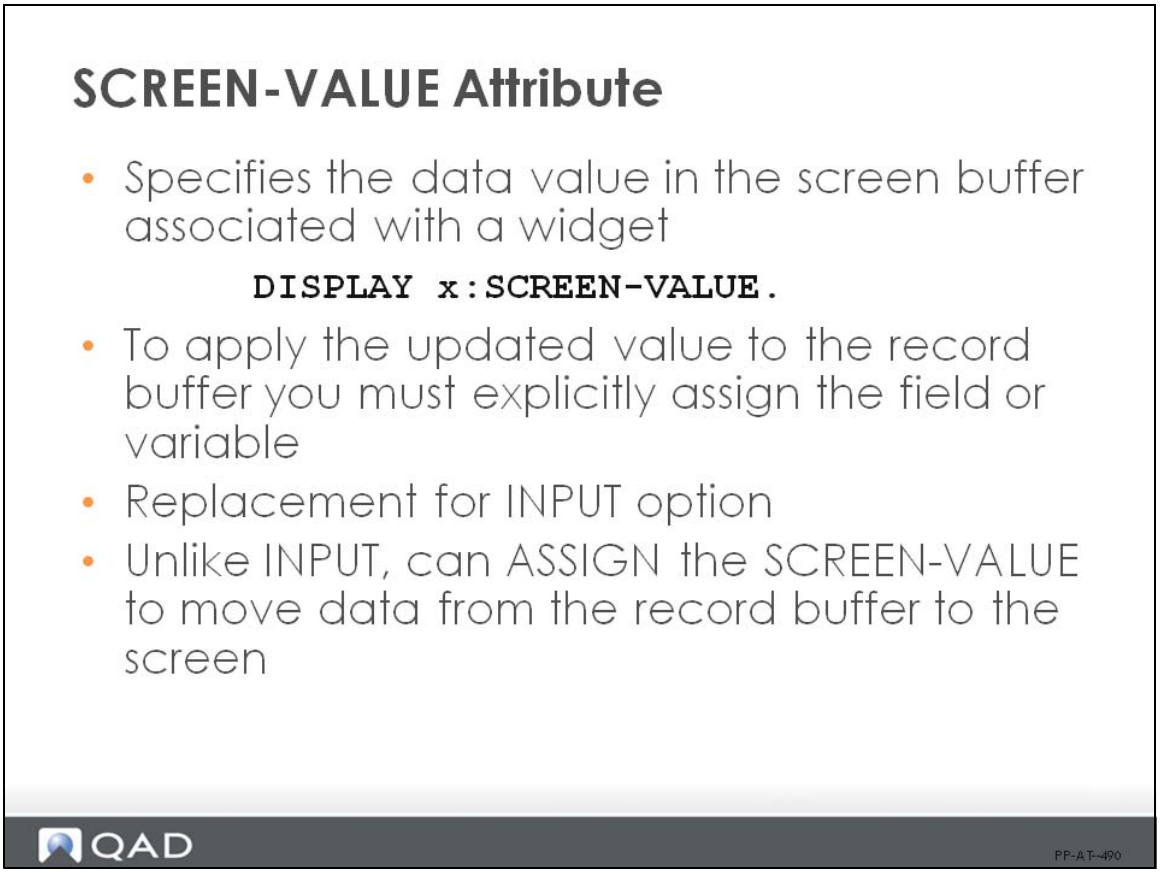
Session Trigger Record Scoping



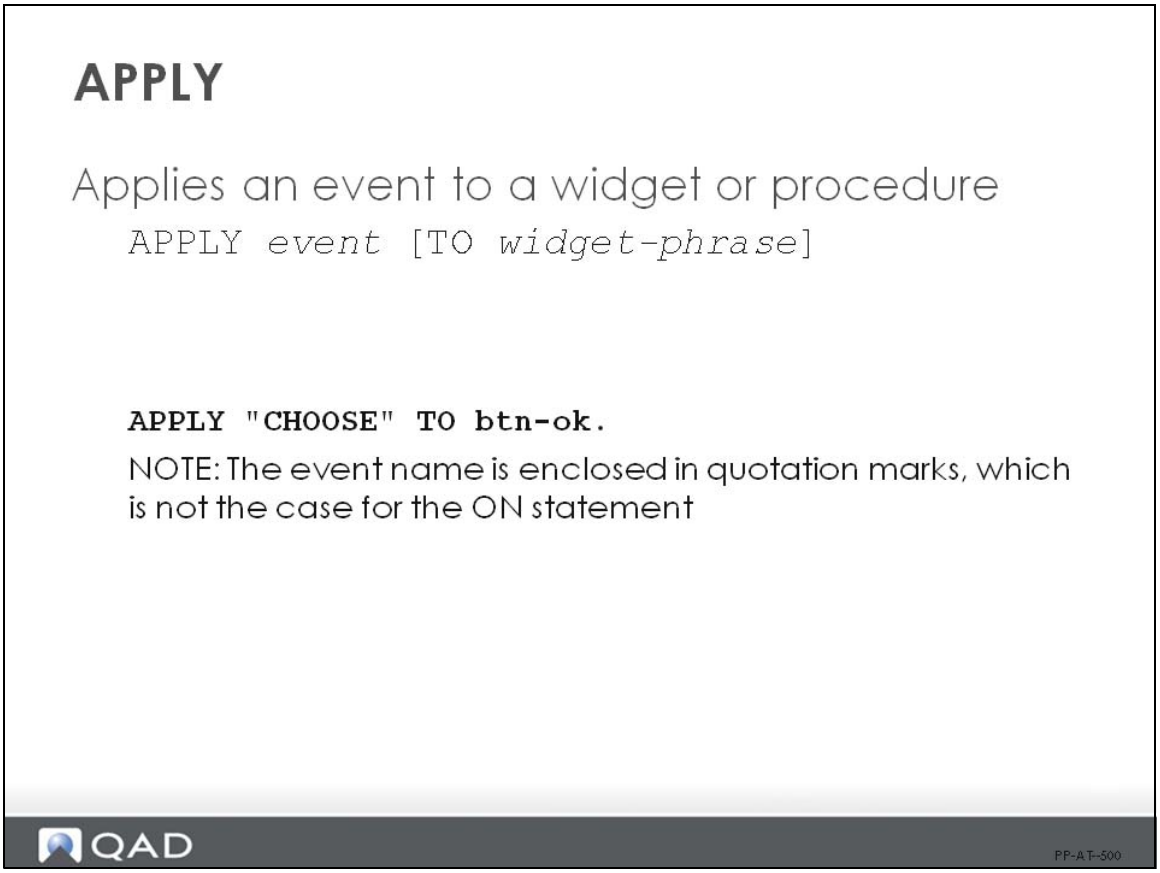
Session Trigger Frame Properties



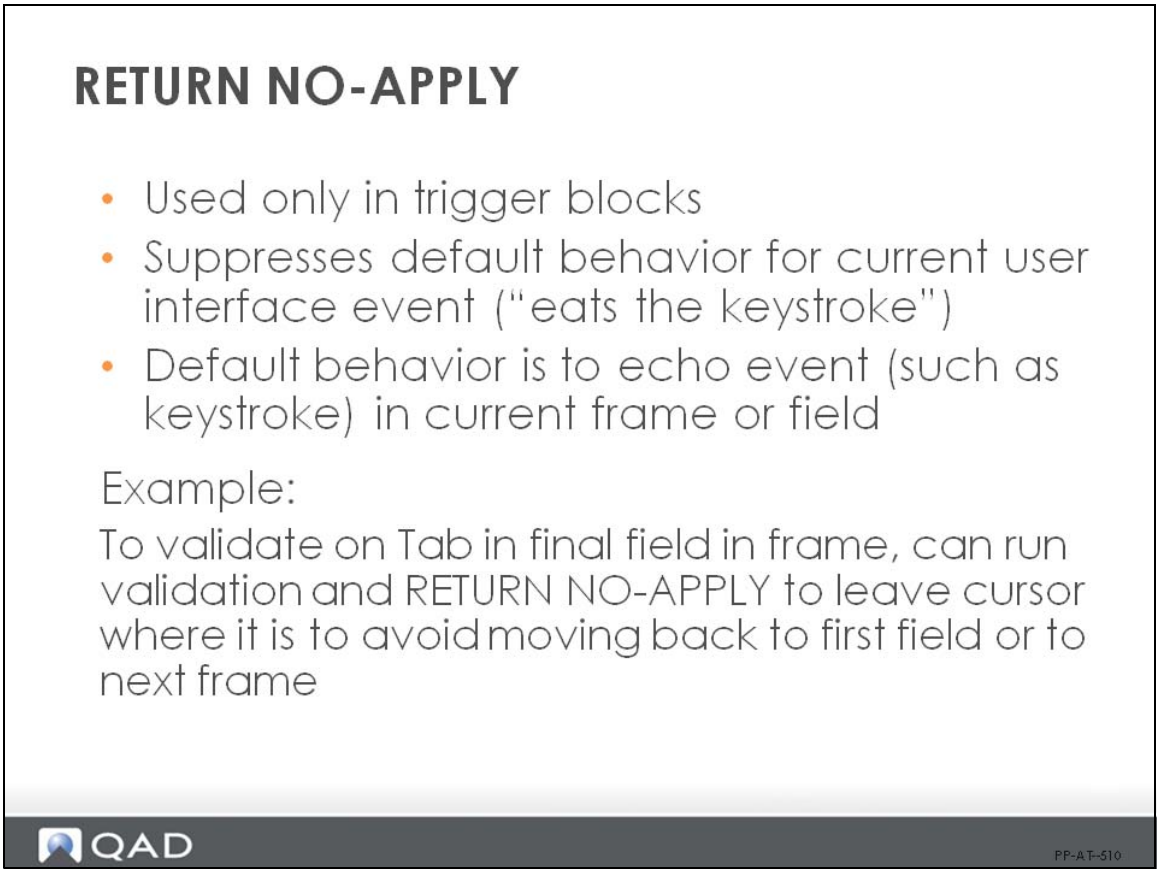
SCREEN-VALUE Attribute



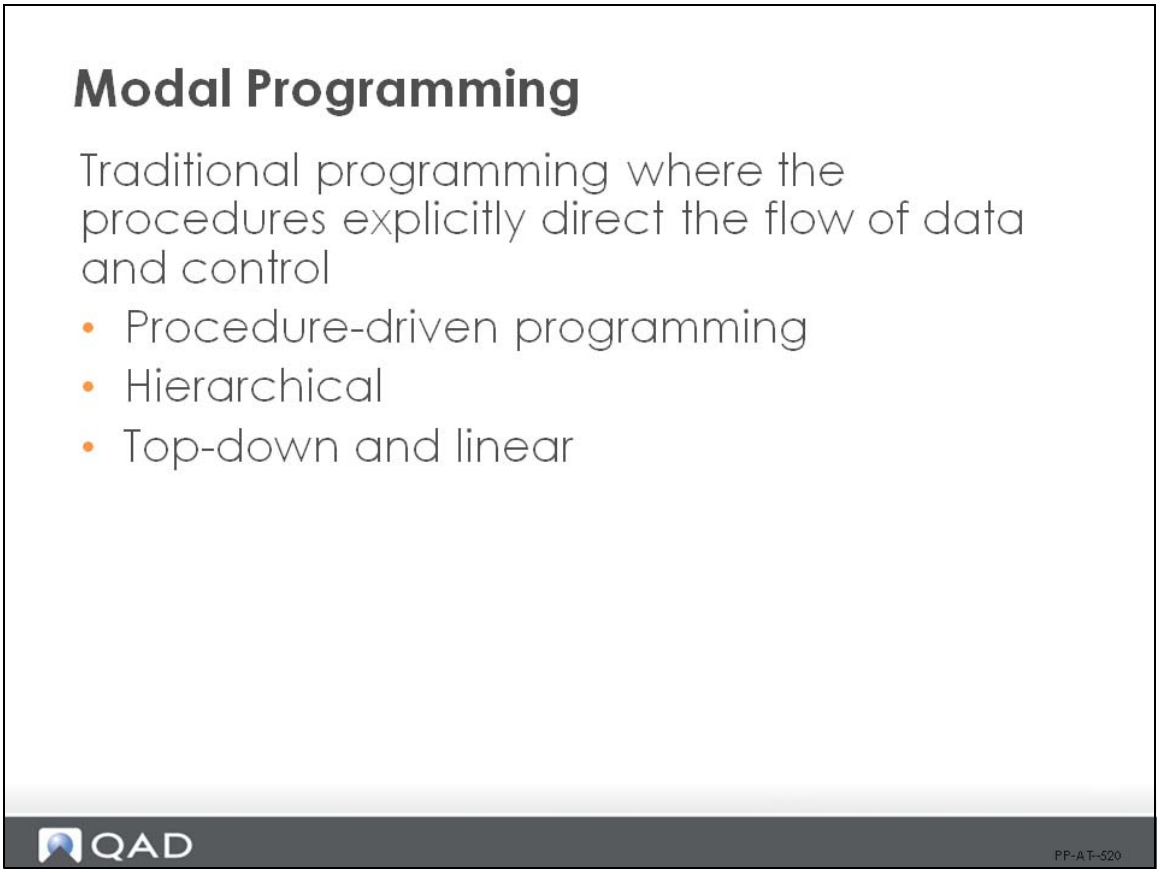
APPLY


APPLY
Applies an event to a widget or procedure
APPLY event [TO widget-phrase]
APPLY "CHOOSE" TO btn-ok.
NOTE: The event name is enclosed in quotation marks, which is not the case for the ON statement
QAD
PP-AT-500

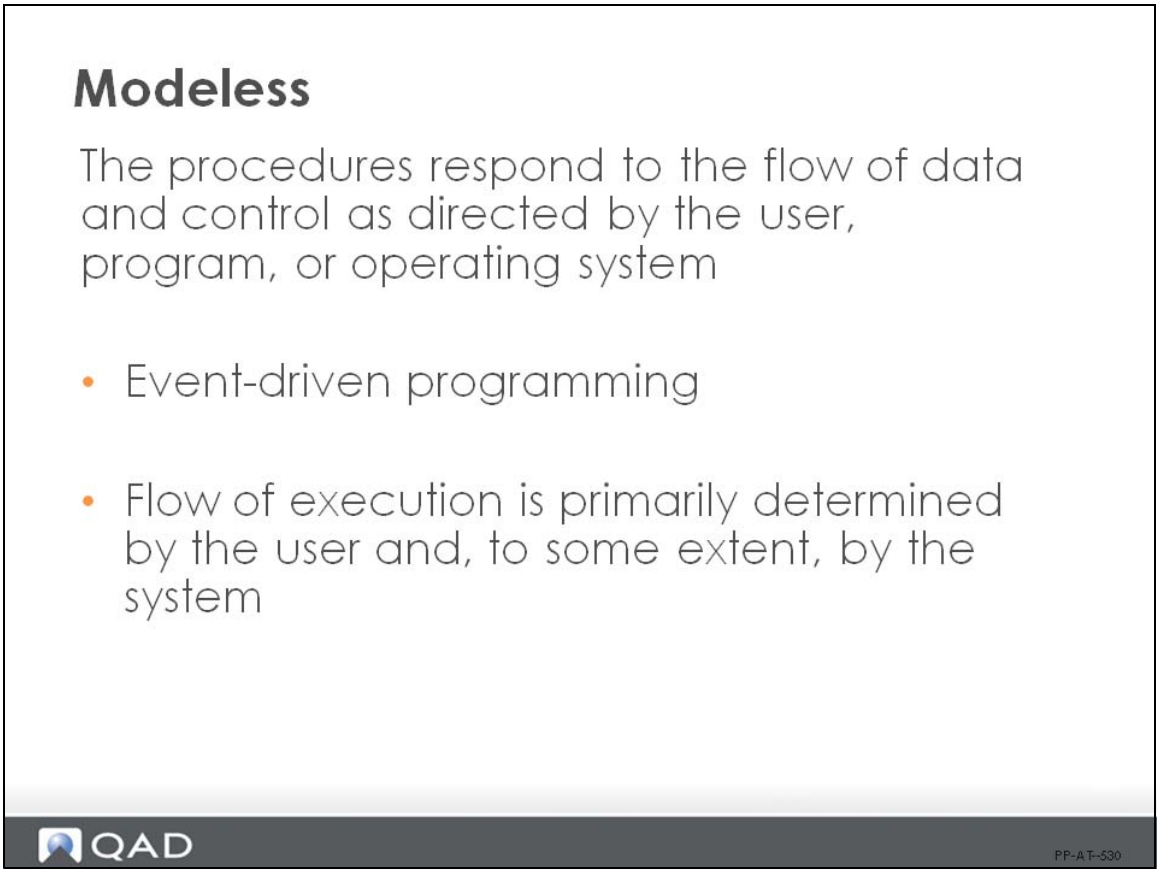
RETURN NO-APPLY



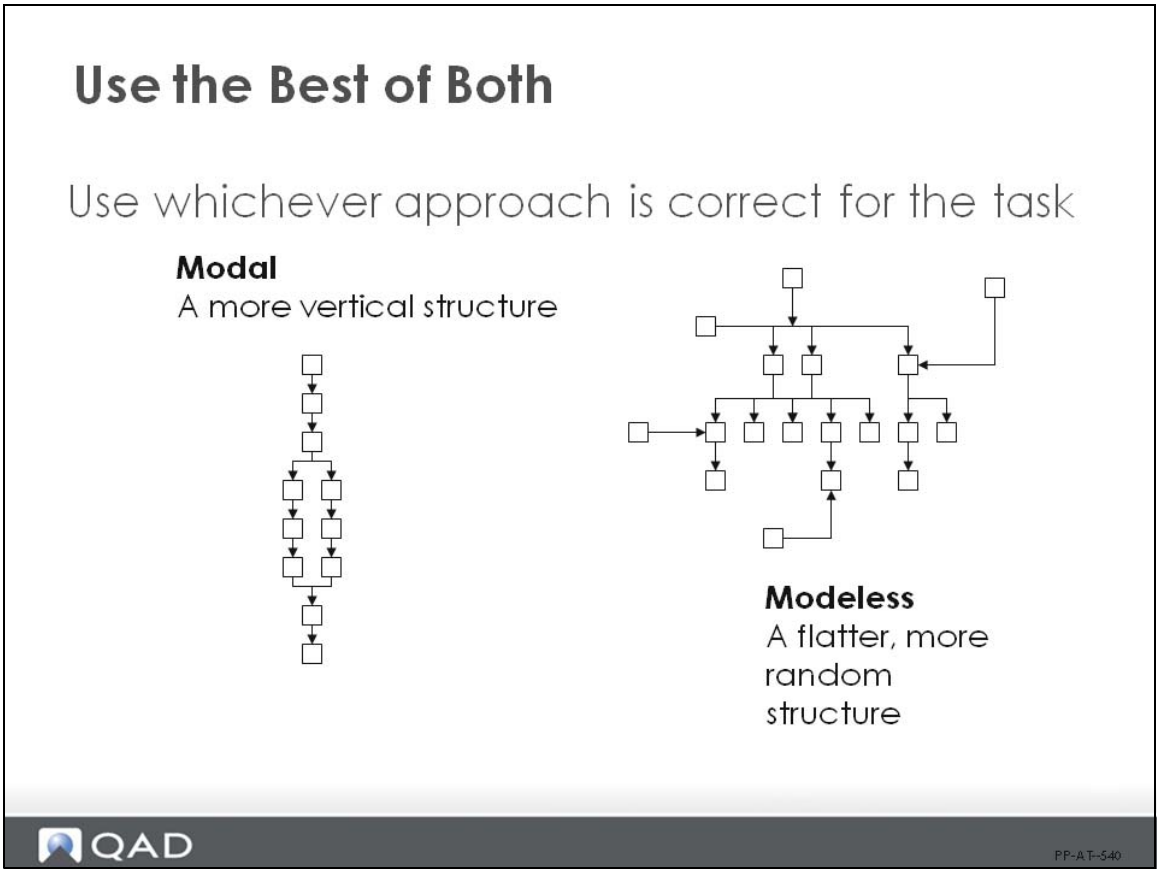
Modal Programming



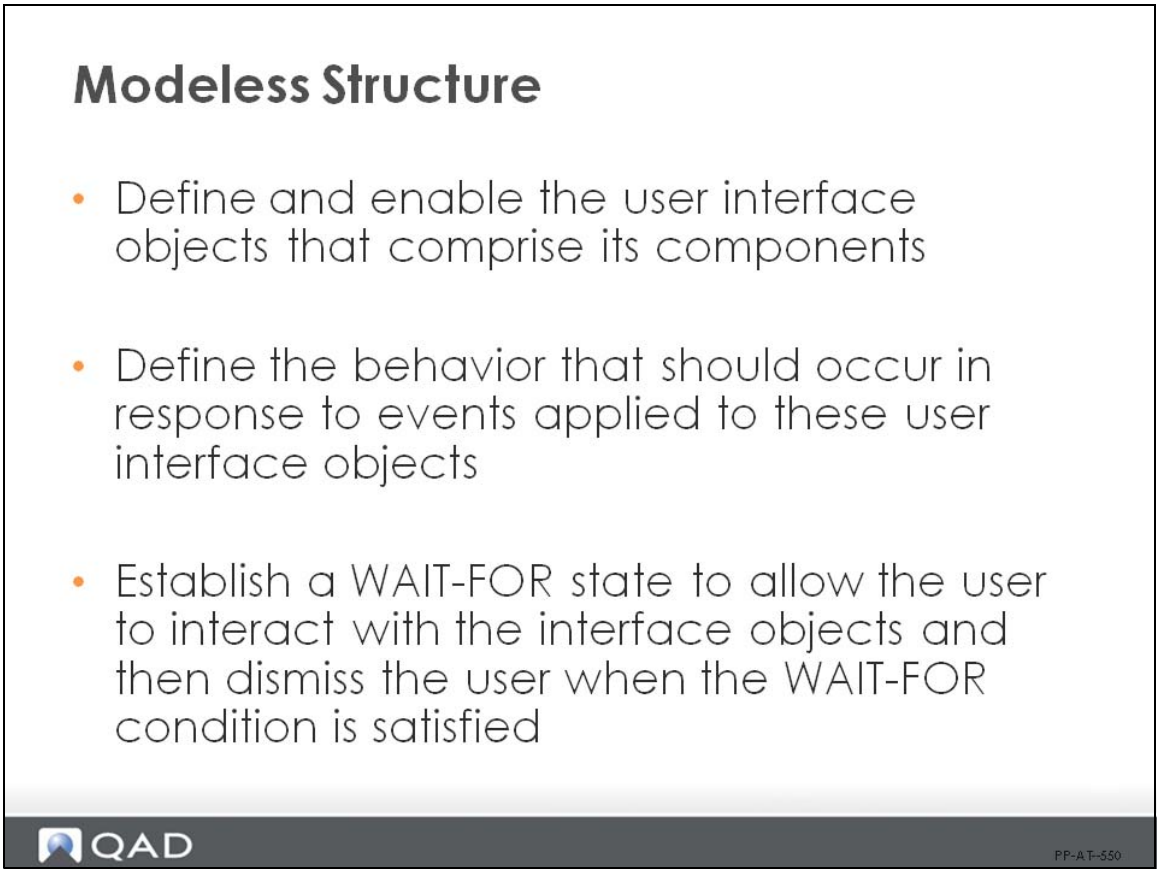
Modeless


Modeless
The procedures respond to the flow of data and control as directed by the user, program, or operating system
Event-driven programming
Flow of execution is primarily determined by the user and, to some extent, by the system
QAD
PP-AT-530

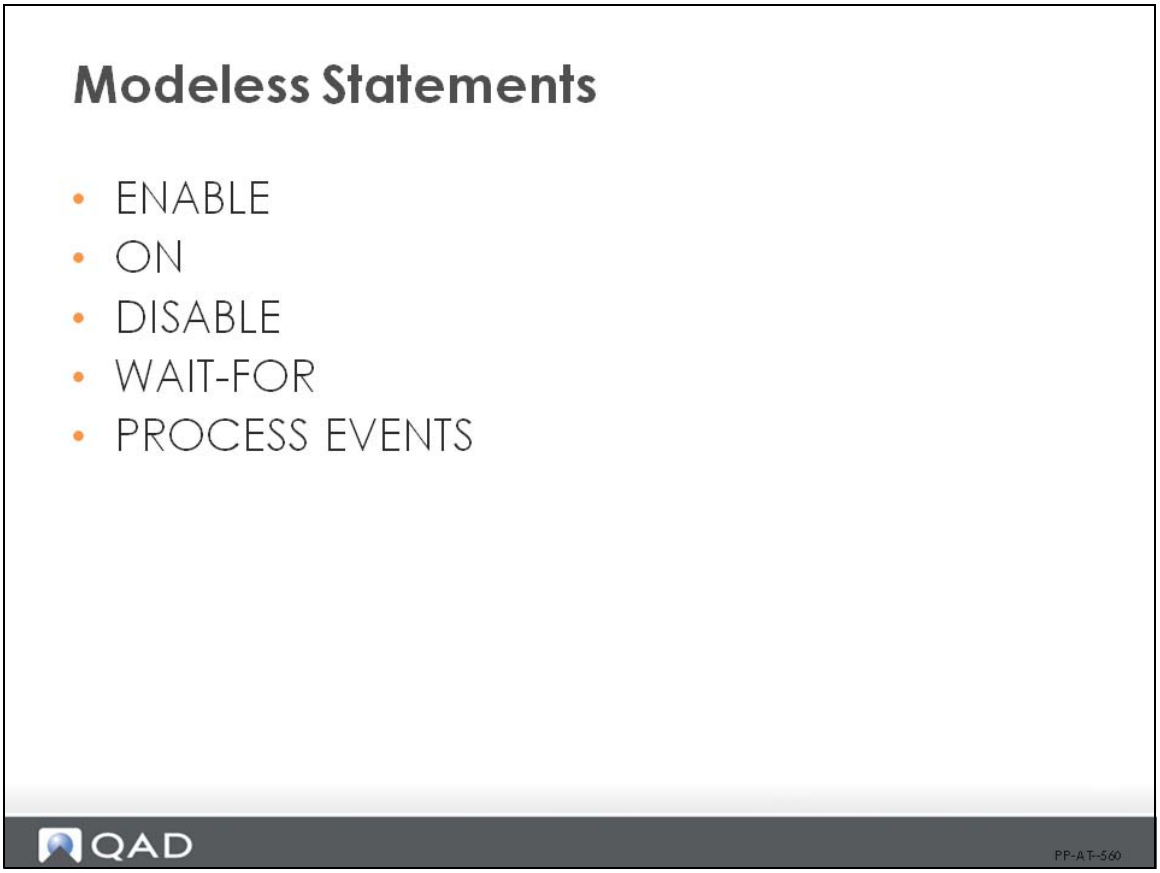
Use the Best of Both



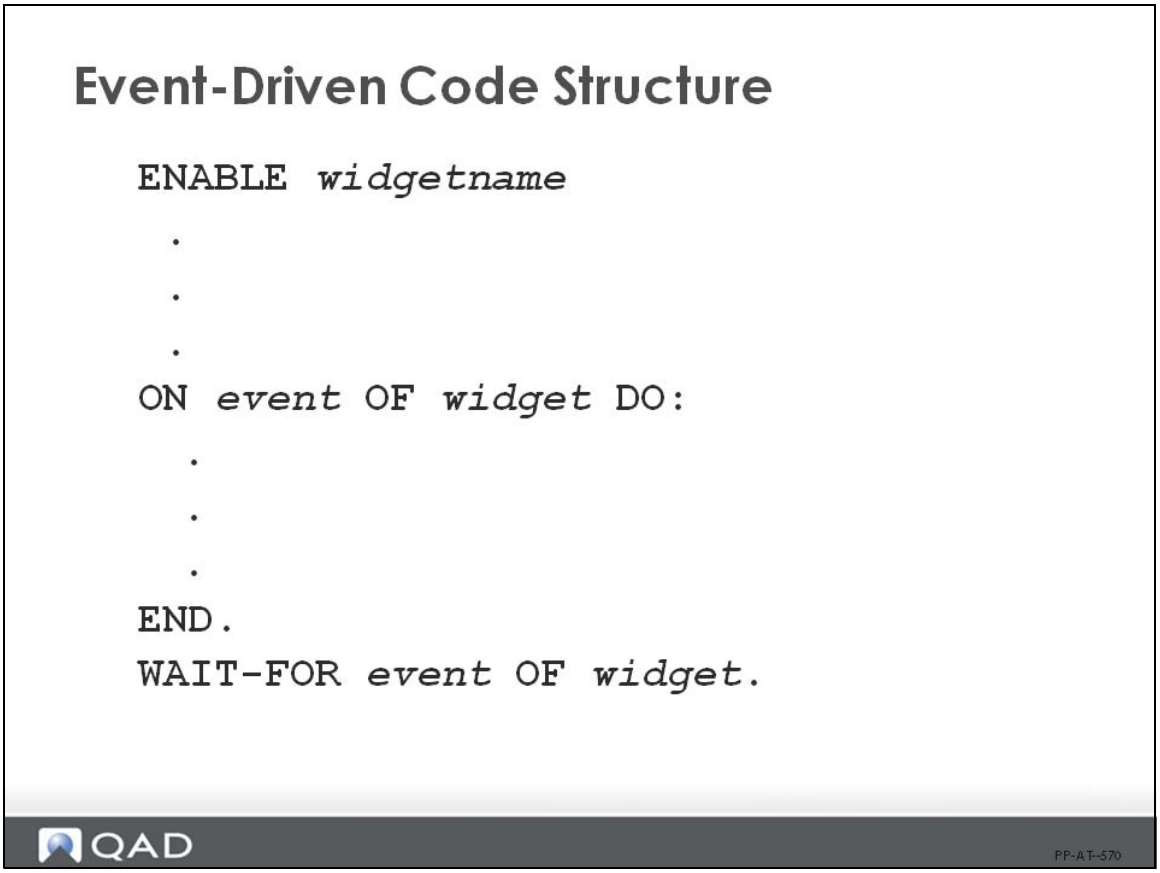
Modeless Structure



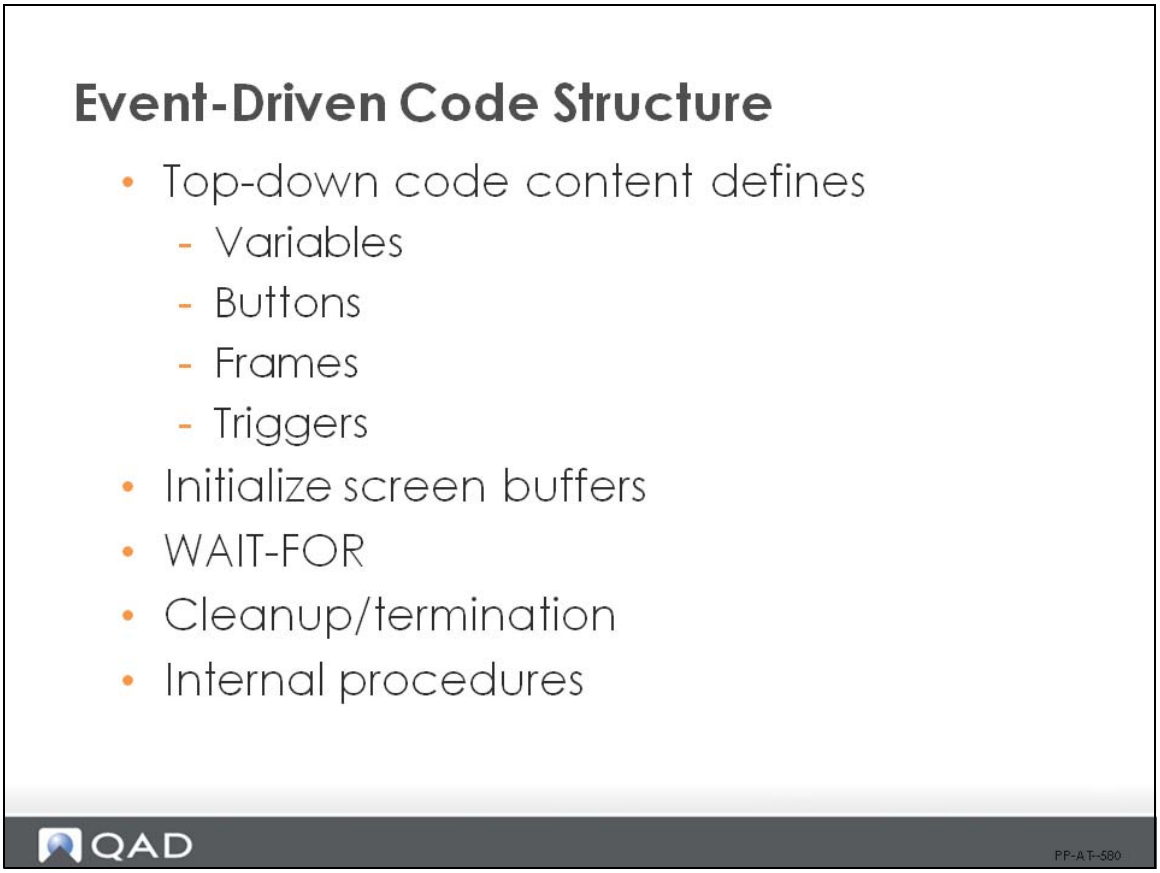
Modeless Statements



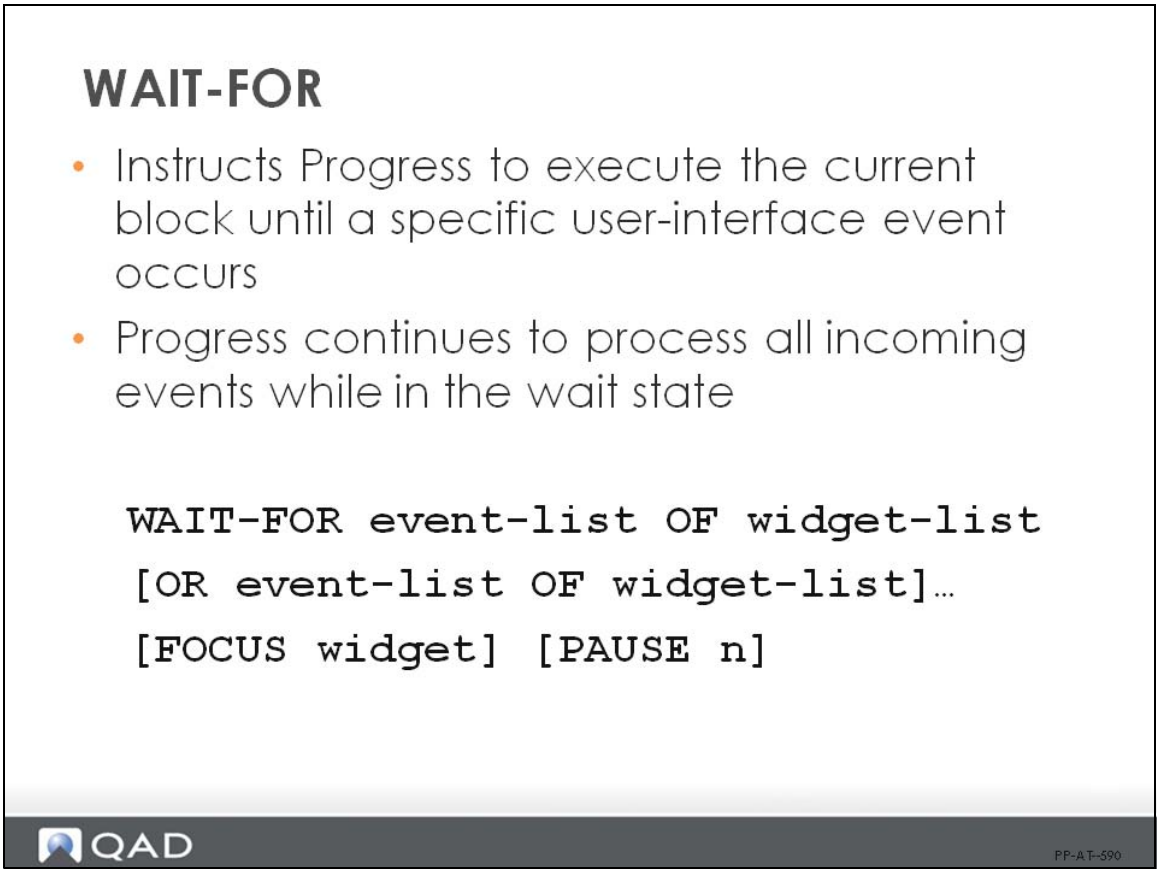
Event-Driven Code Structure



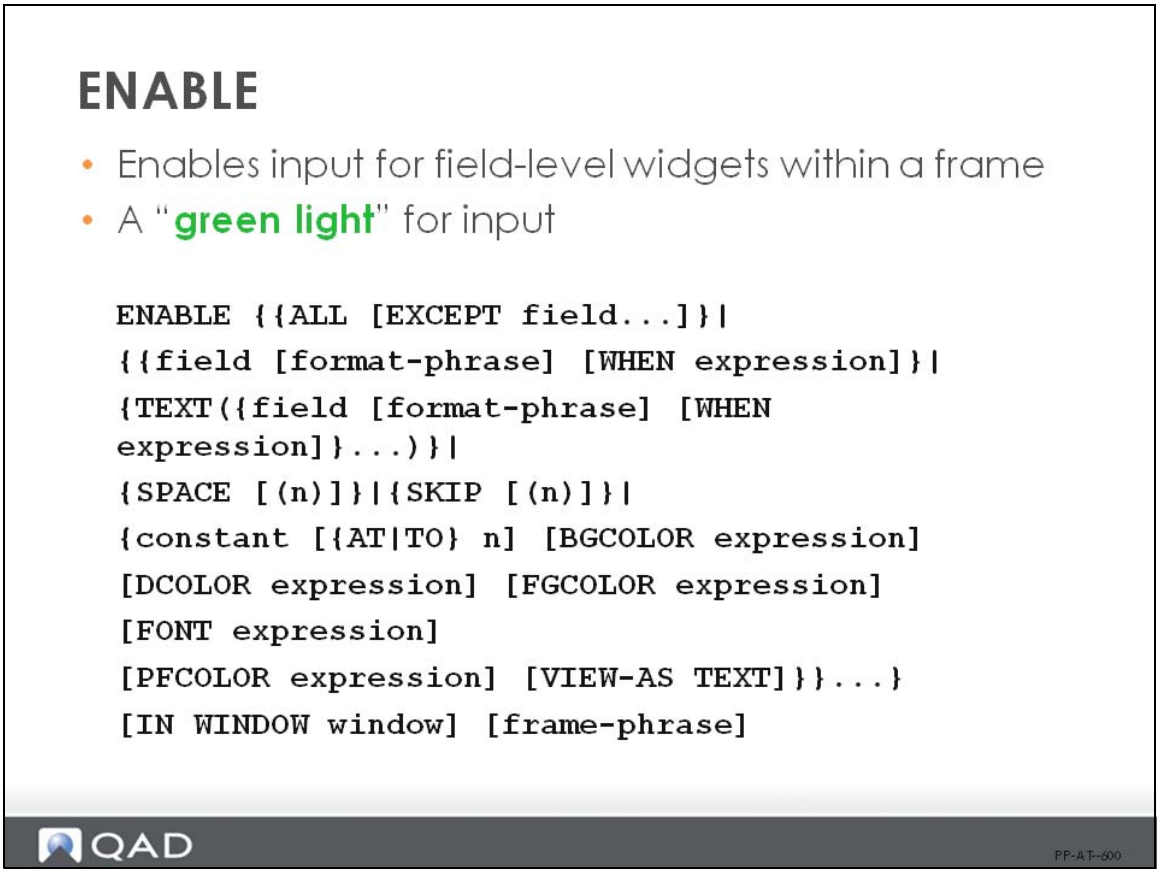
Event-Driven Code Structure



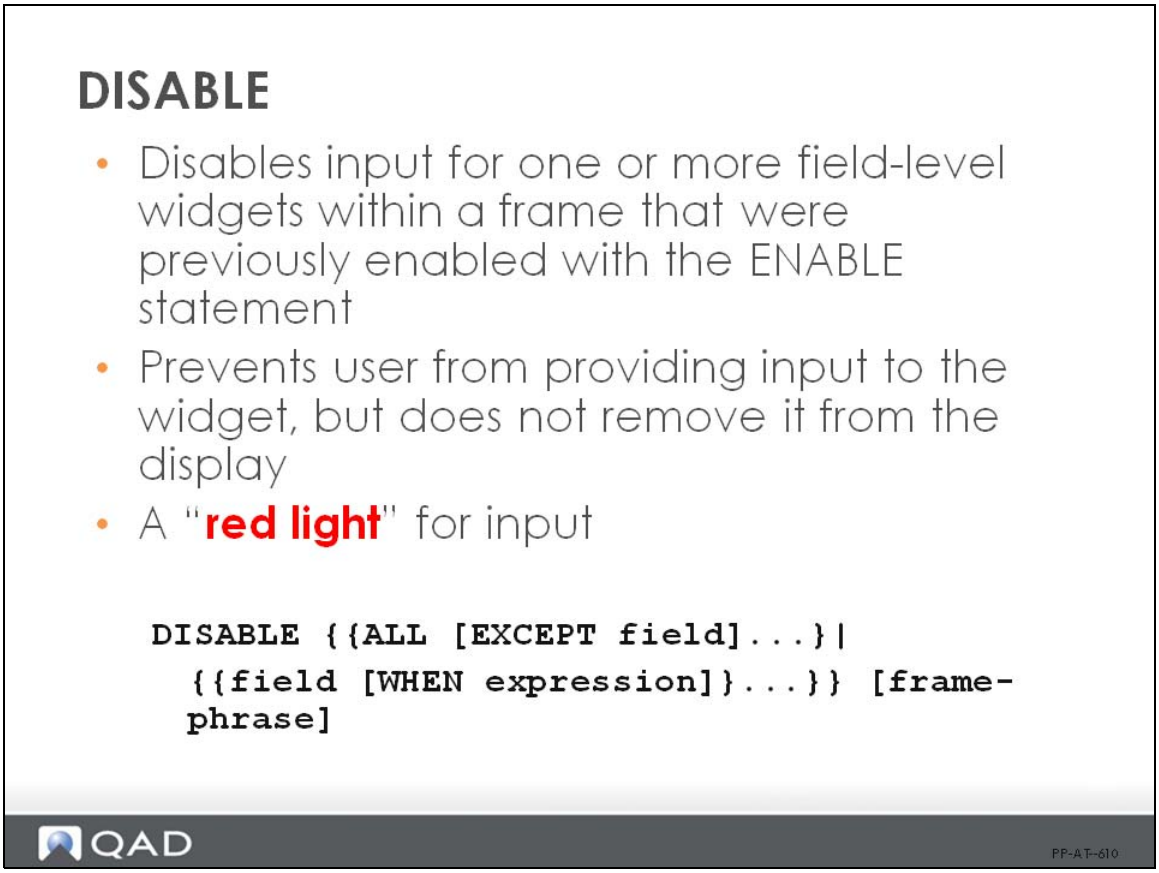
WAIT-FOR


WAIT-FOR
Instructs Progress to execute the current block until a specific user-interface event occurs
Progress continues to process all incoming events while in the wait state
WAIT-FOR event-list OF widget-list
[OR event-list OF widget-list]…
[FOCUS widget] [PAUSE n]
QAD
PP-AT-590

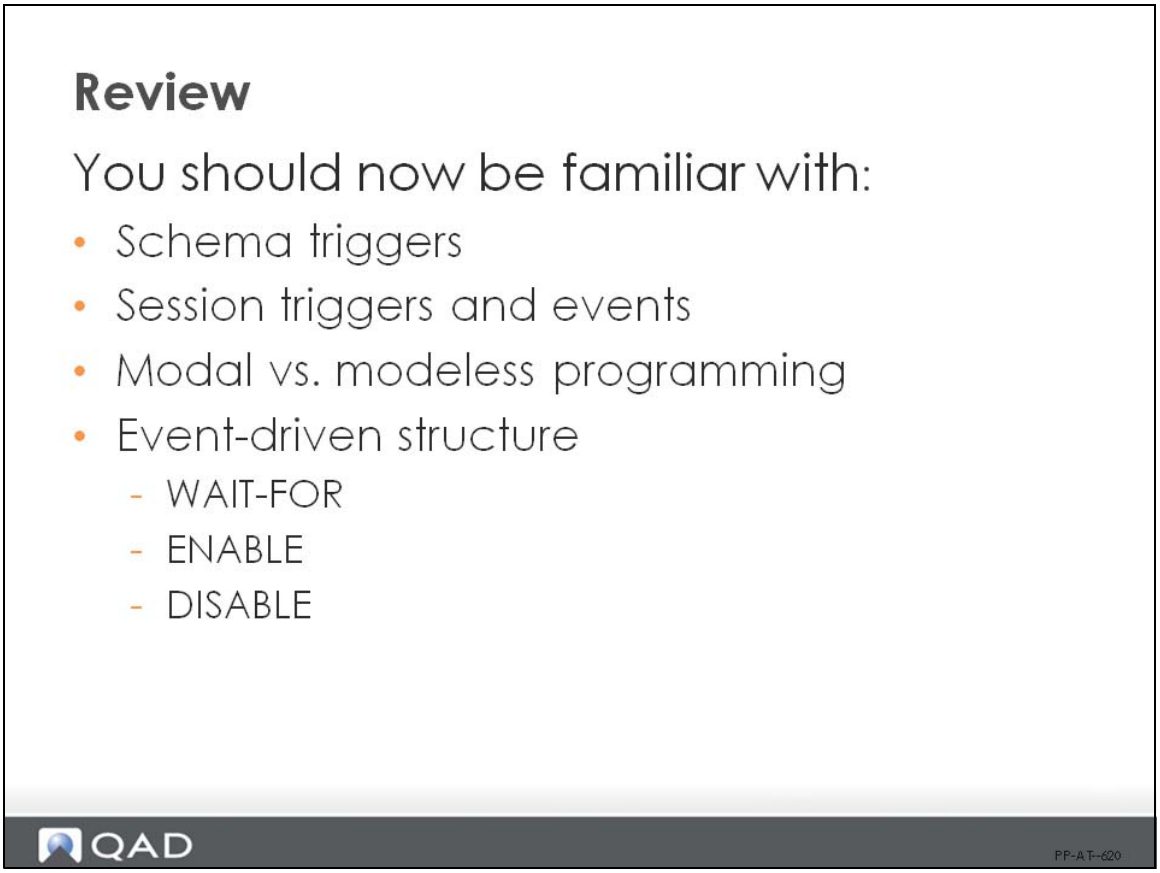
ENABLE


ENABLE
• Enables input for field-level widgets within a frame
• A "green light" for input
ENABLE {{ALL [EXCEPT field...]}|
{{field [format-phrase] [WHEN expression]}|
{TEXT({field [format-phrase] [WHEN
expression]}...)}|
{SPACE [(n)]}|{SKIP [(n)]}|
{constant [{AT|TO} n] [BGCOLOR expression]
[DCOLOR expression] [FGCOLOR expression]
[FONT expression]
[PFCOLOR expression] [VIEW-AS TEXT]}}...}
[IN WINDOW window] [frame-phrase]
QAD
PP-AT-600

DISABLE


DISABLE
Disables input for one or more field-level widgets within a frame that were previously enabled with the ENABLE statement
Prevents user from providing input to the widget, but does not remove it from the display
A "red light" for input
DISABLE {{ALL [EXCEPT field]...}|
{{field [WHEN expression]}...}} [frame-
phrase]
QAD
PP-AT-610

Review


Review
You should now be familiar with:
Schema triggers
Session triggers and events
Modal vs. modeless programming
Event-driven structure
WAIT-FOR
ENABLE
DISABLE
QAD
PP-AT-620

Lesson 15: Queries and Handles (Optional)

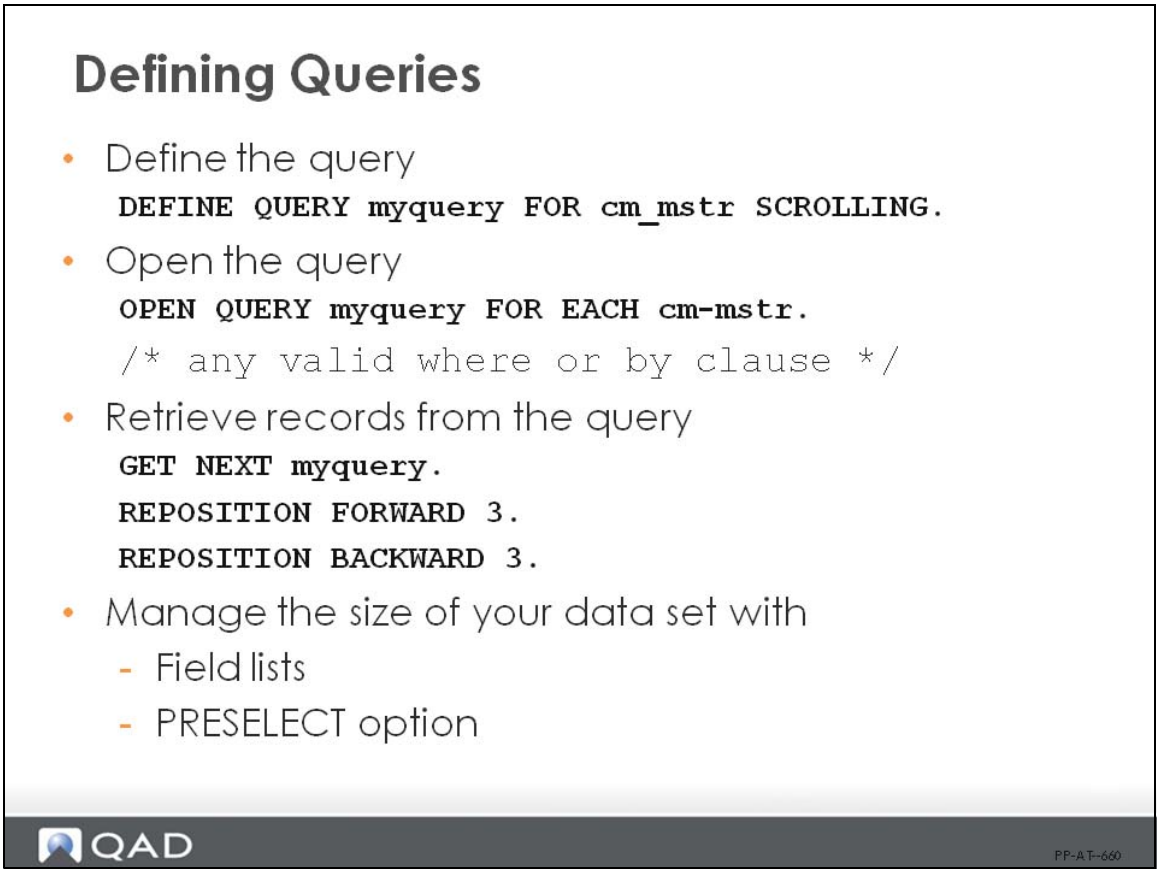
Upon completion of this lesson, you will be familiar with the following topics:

- DEFINE QUERY statement
- OPEN QUERY statement
- Dynamic queries and buffers
- Dynamic query keywords
- Buffer keywords

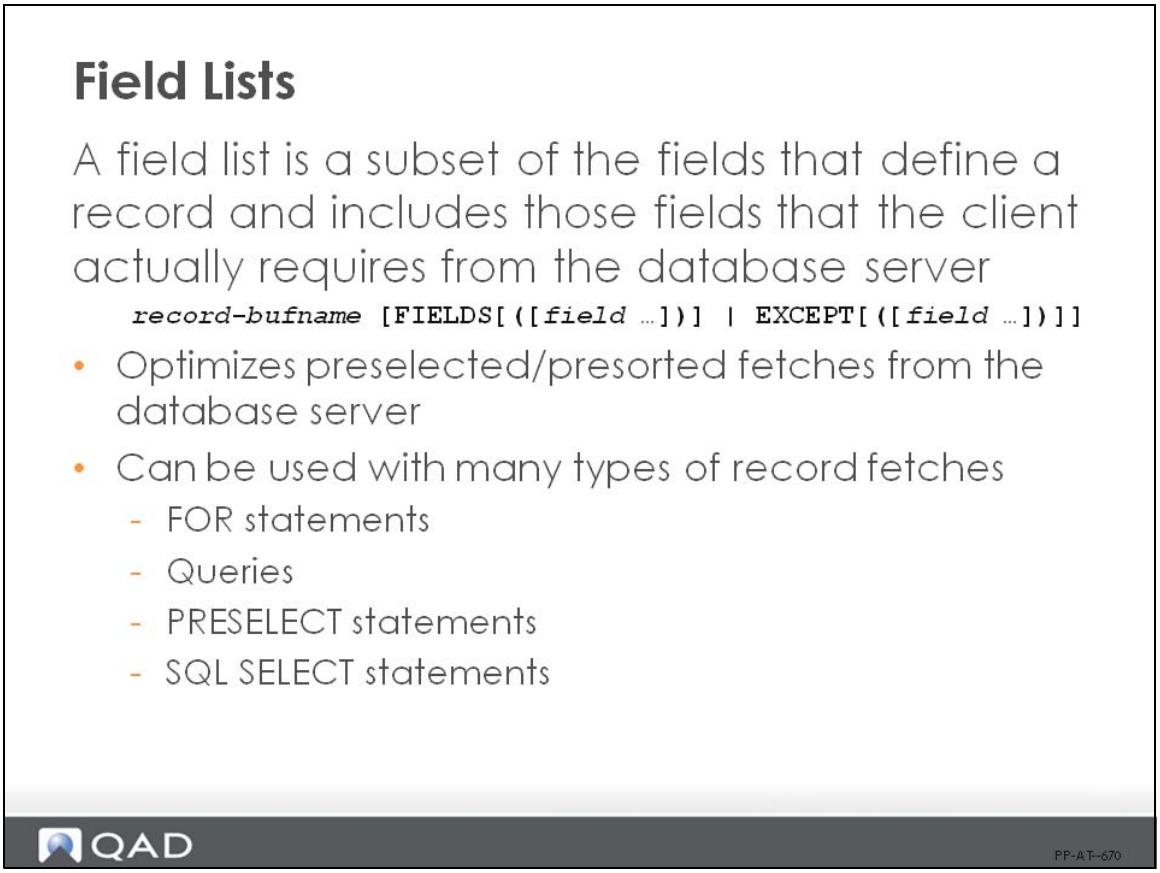
Overview


Overview
Queries: a list of database records from one or more tables, matching a selection criteria
By default: SHARE-LOCK, non-scrolling
Dynamic: populated from user input
QAD
PP-AT-650

Defining Queries

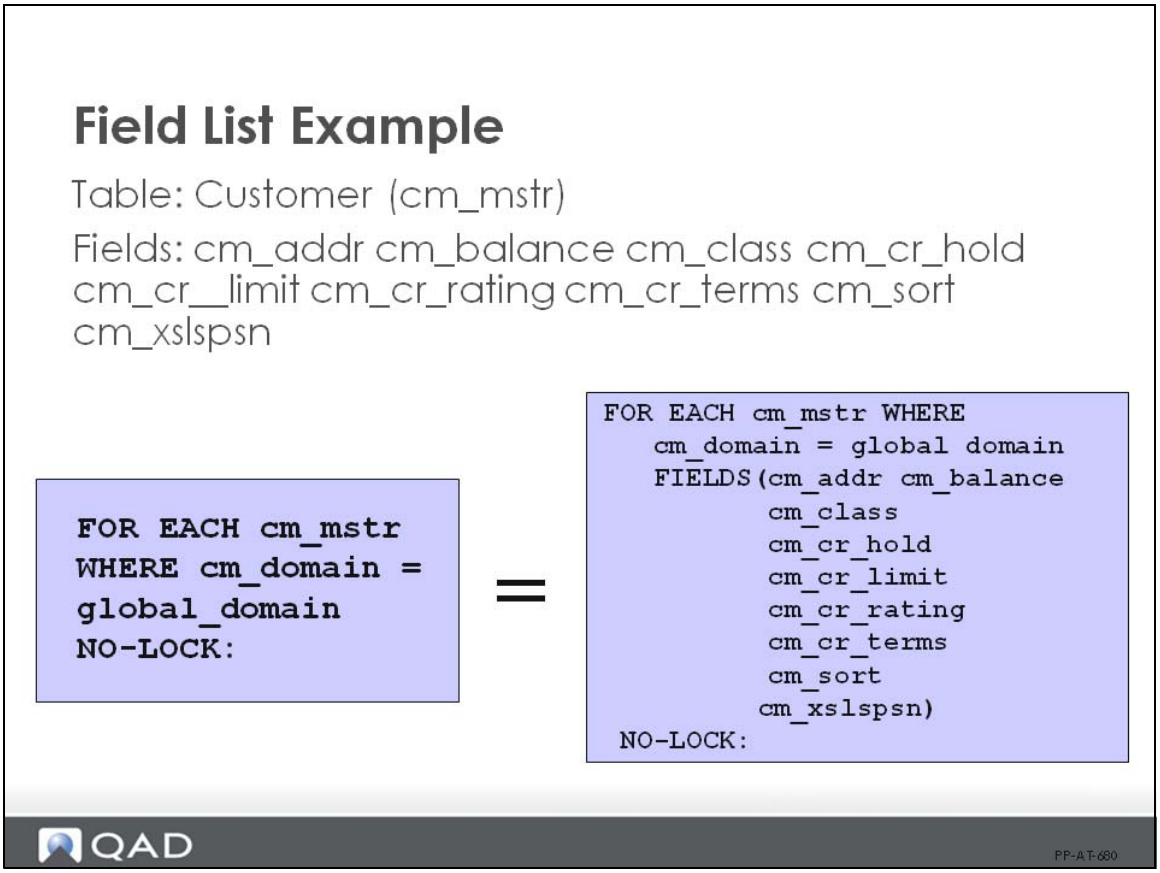


Field Lists



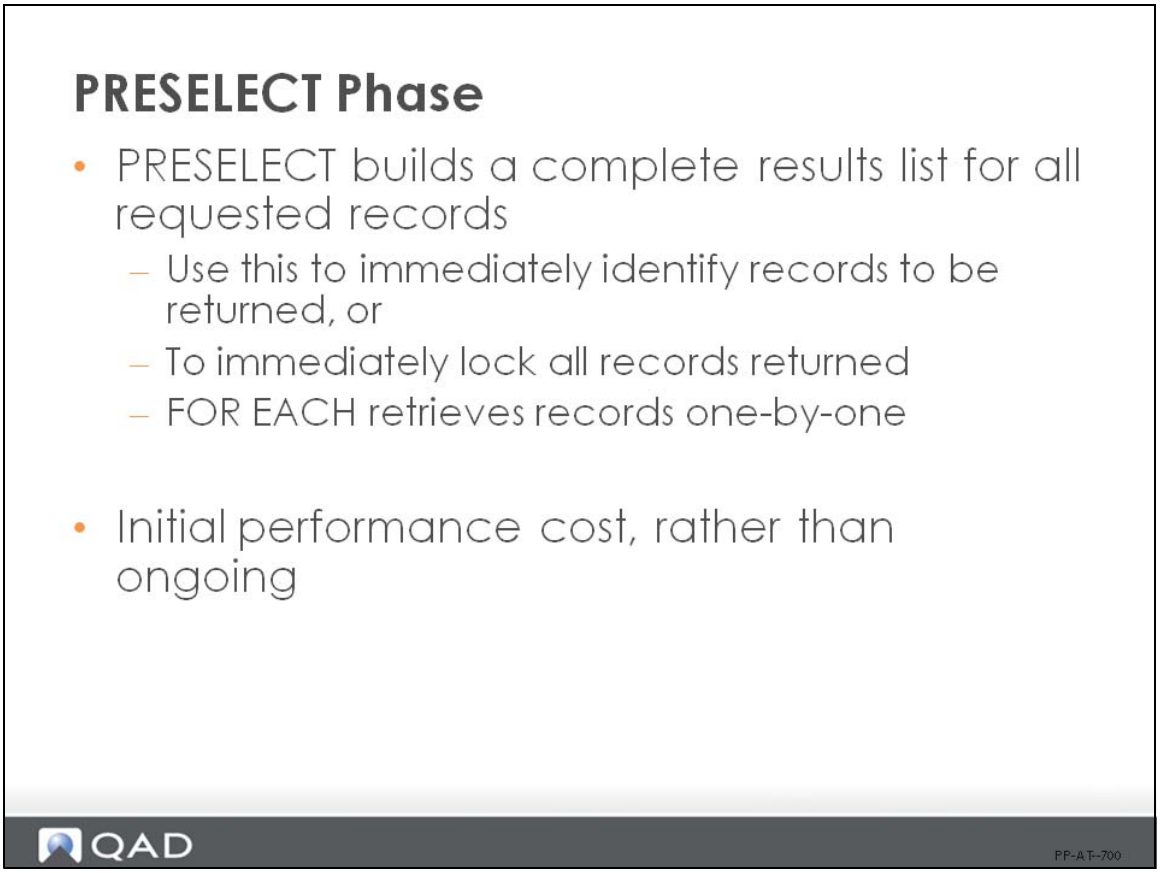
Important Field lists are no longer used in QAD code. While Progress supports this coding construct, creating the correct list is prone to error and difficult to keep up to date. With current broadband access, limiting the data set size is not a significant performance issue.

Field List Example

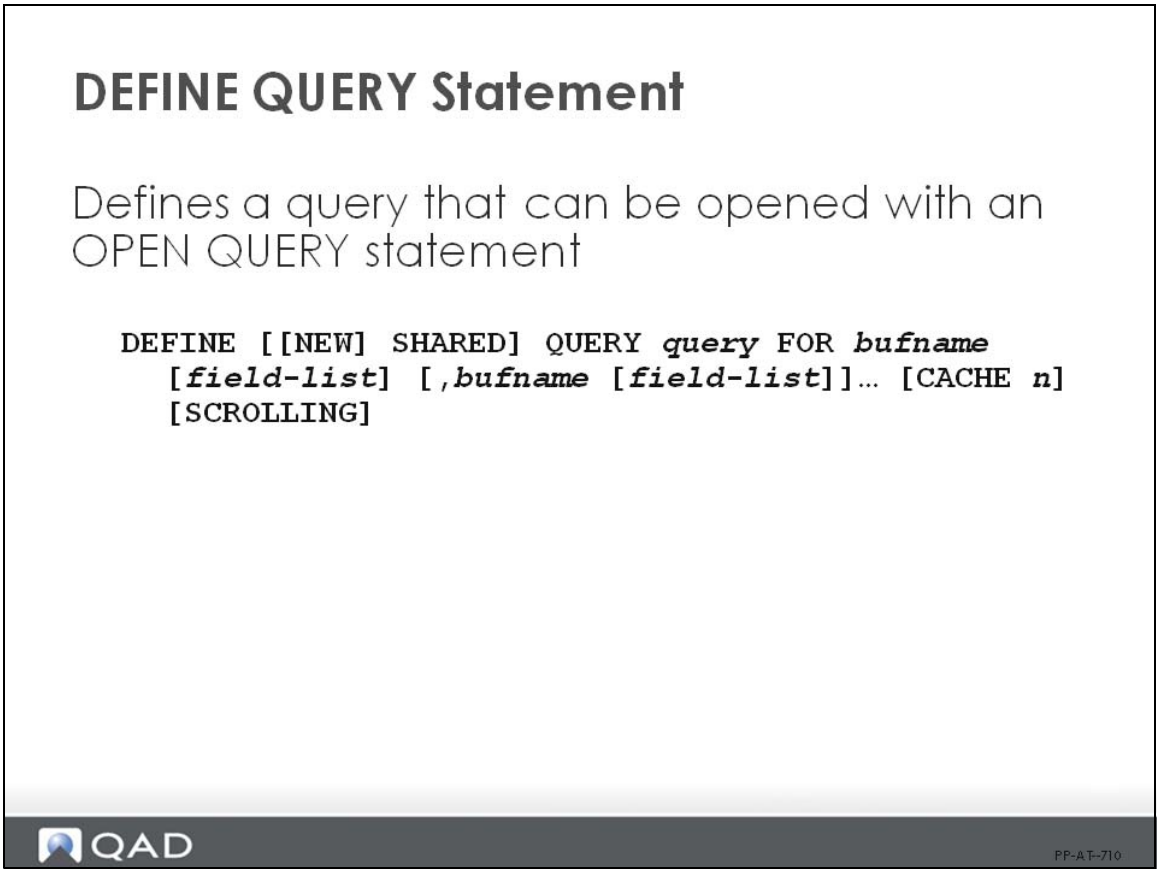


You can deactivate the field list using the `-fldisable` parameter. Progress then retrieves the whole record and overrides the field list in the procedure.

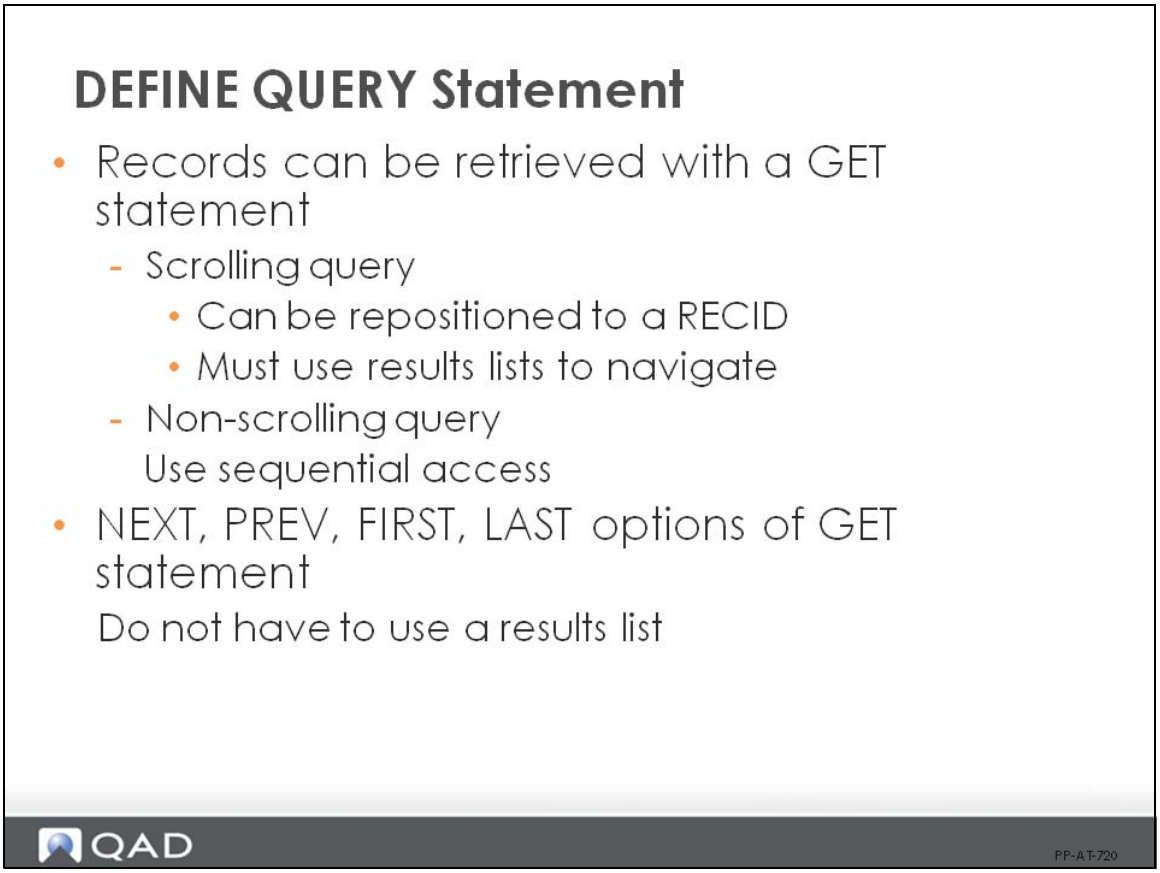
PRESELECT Phrase



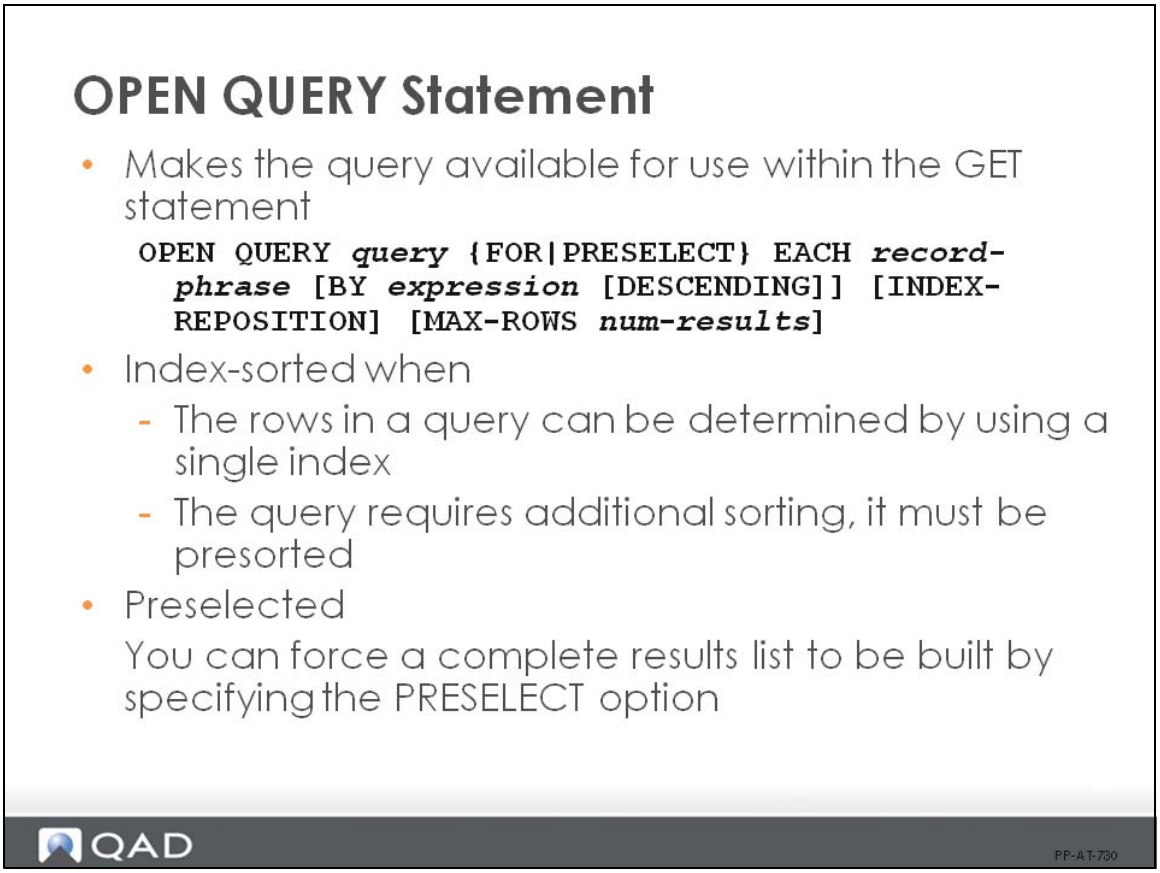
DEFINE QUERY Statement



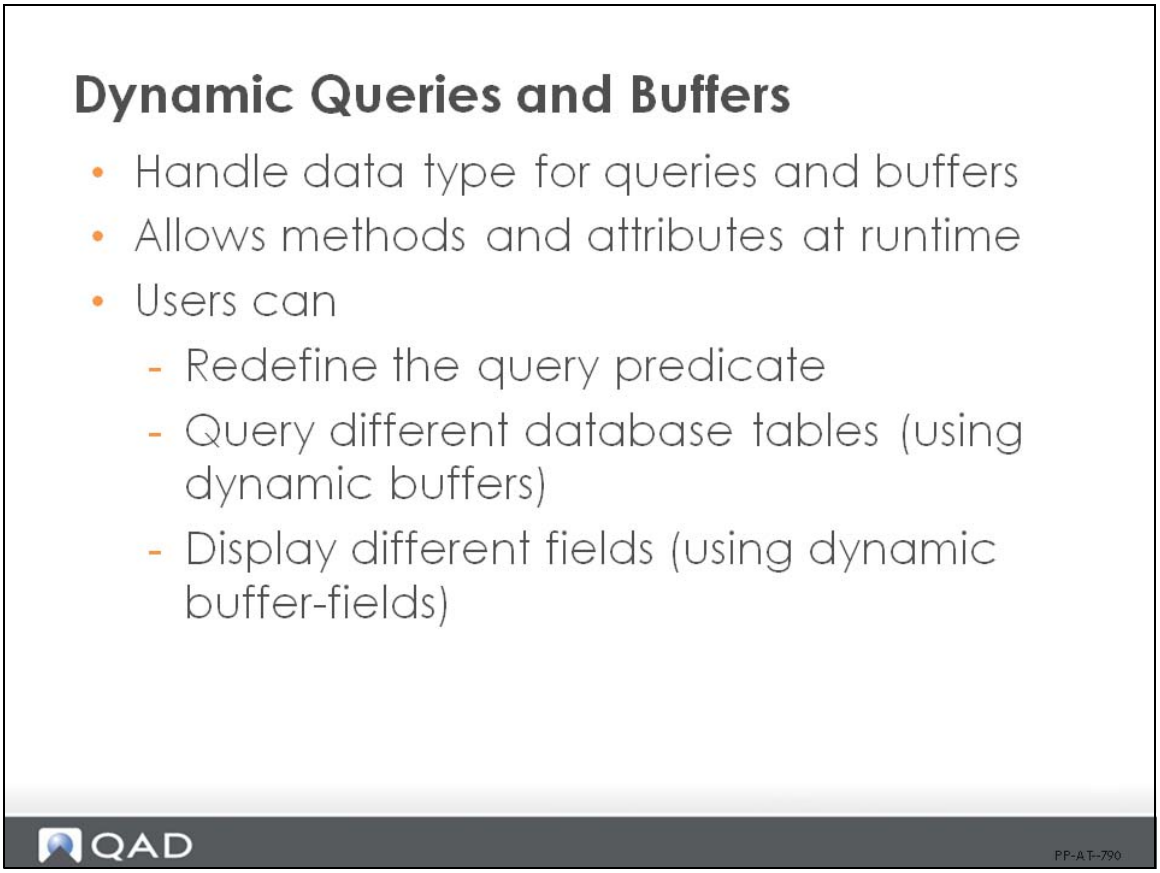
DEFINE QUERY Statement



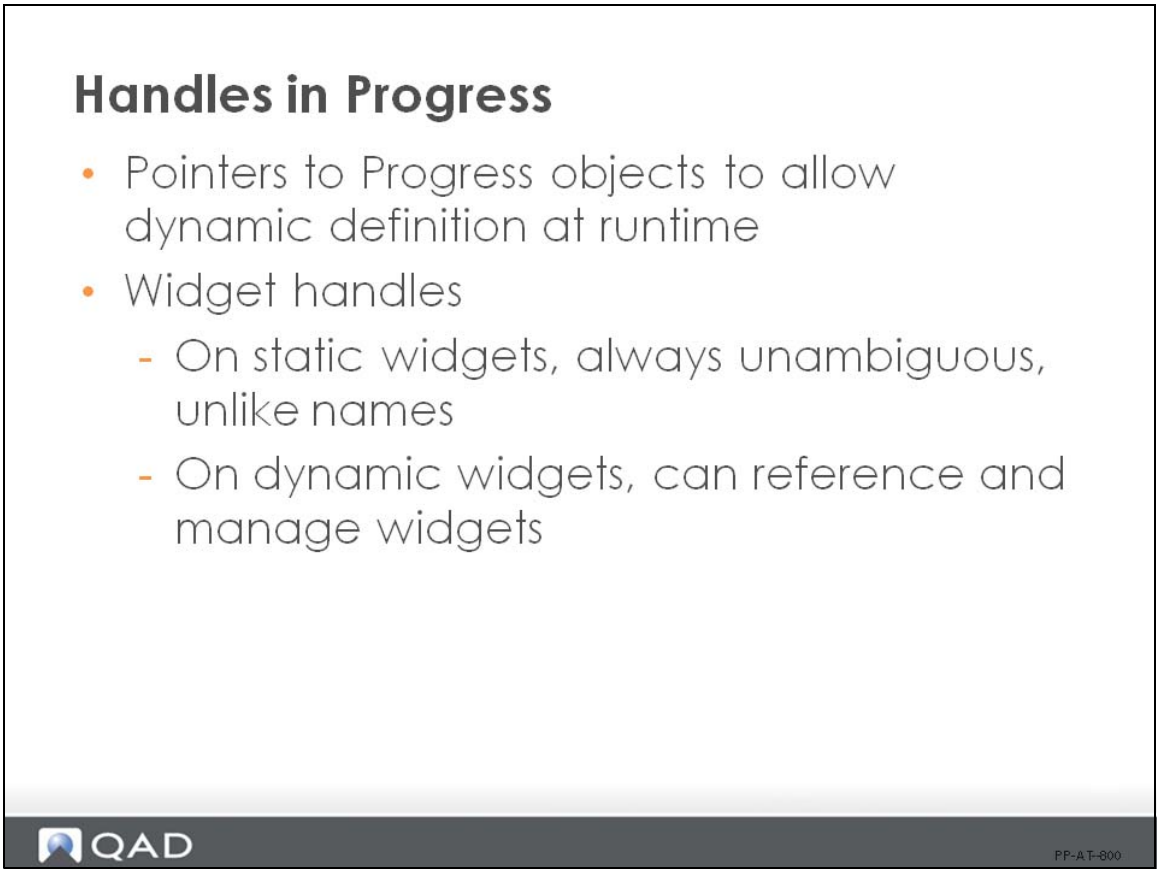
OPEN QUERY Statement



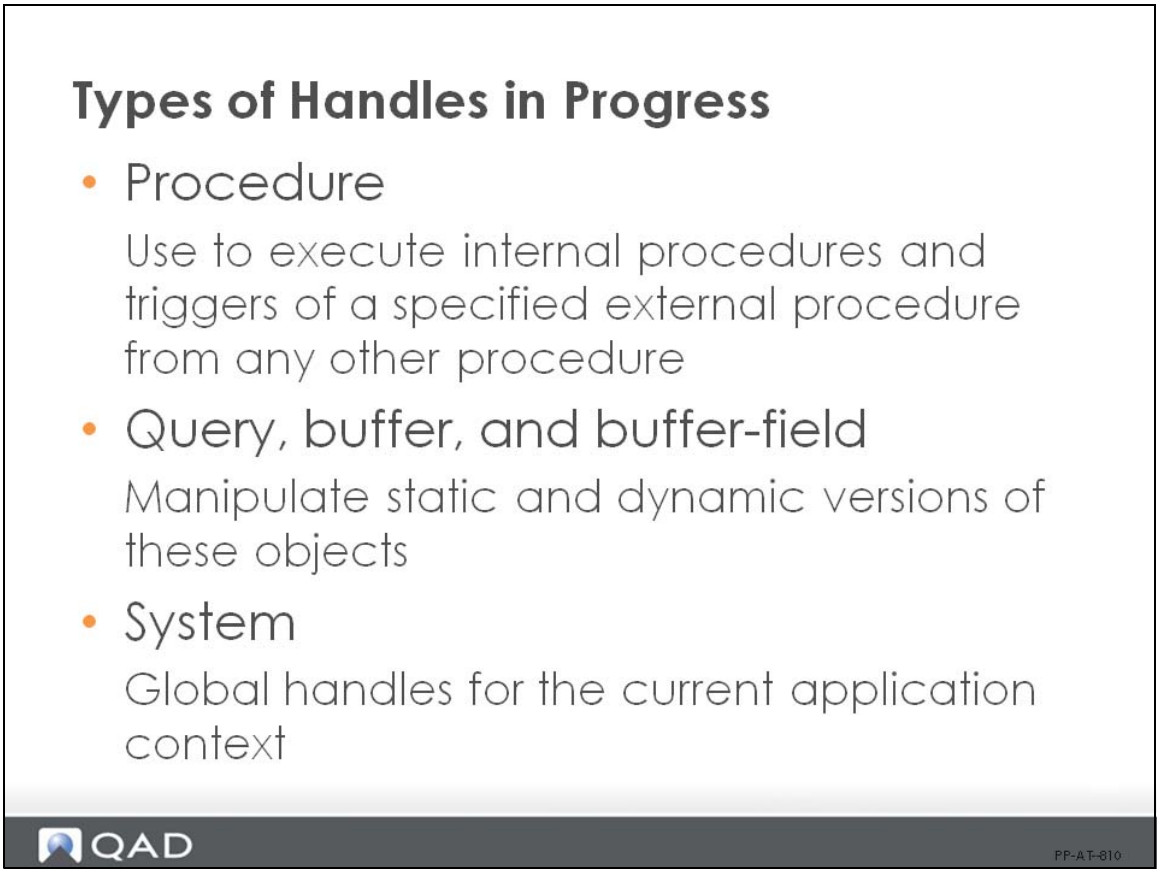
Dynamic Queries and Buffers



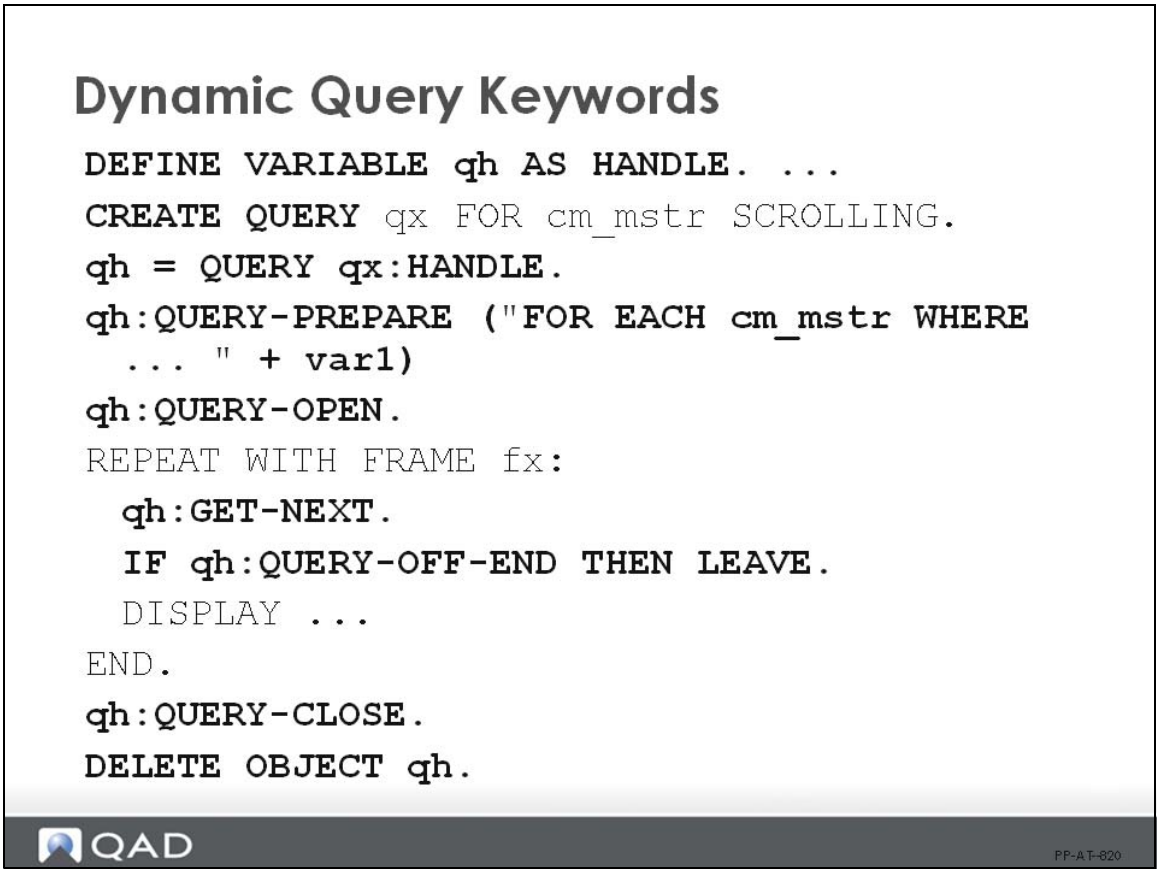
Handles in Progress



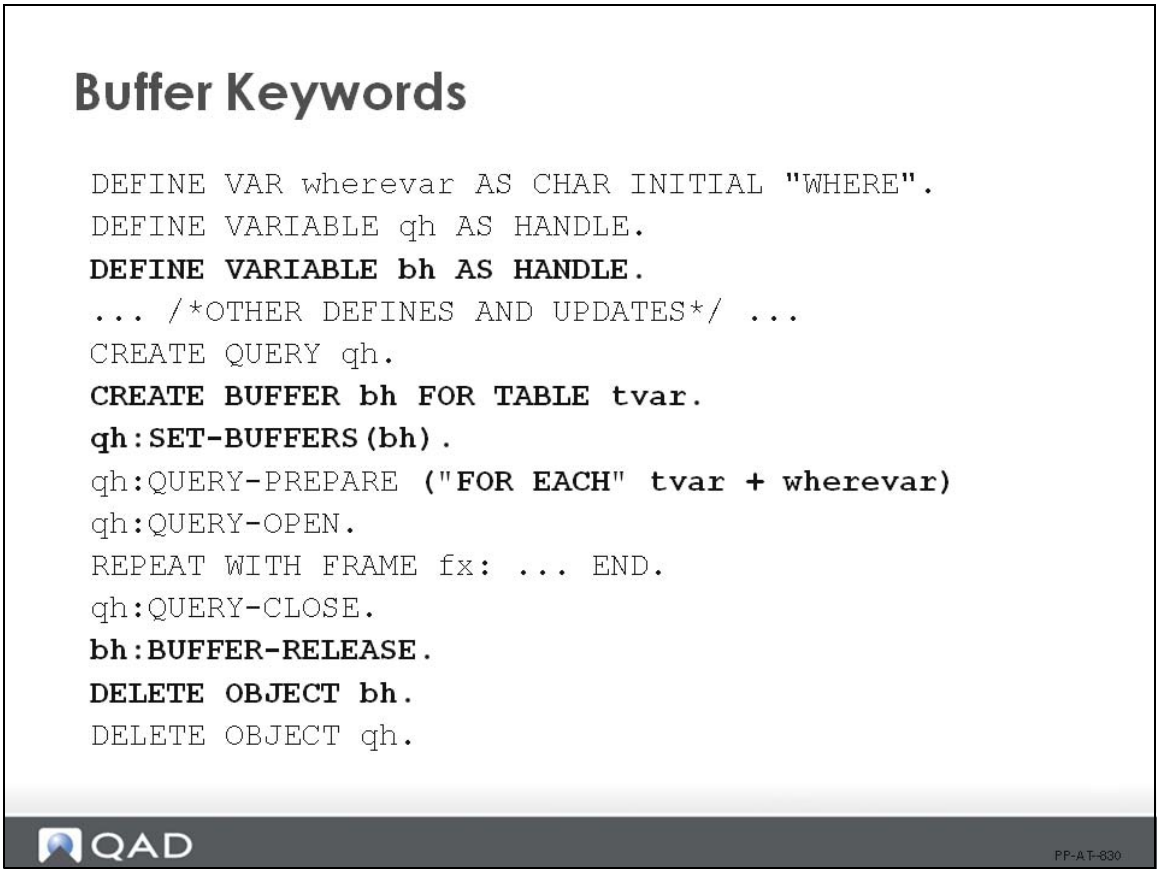
Types of Handles in Progress



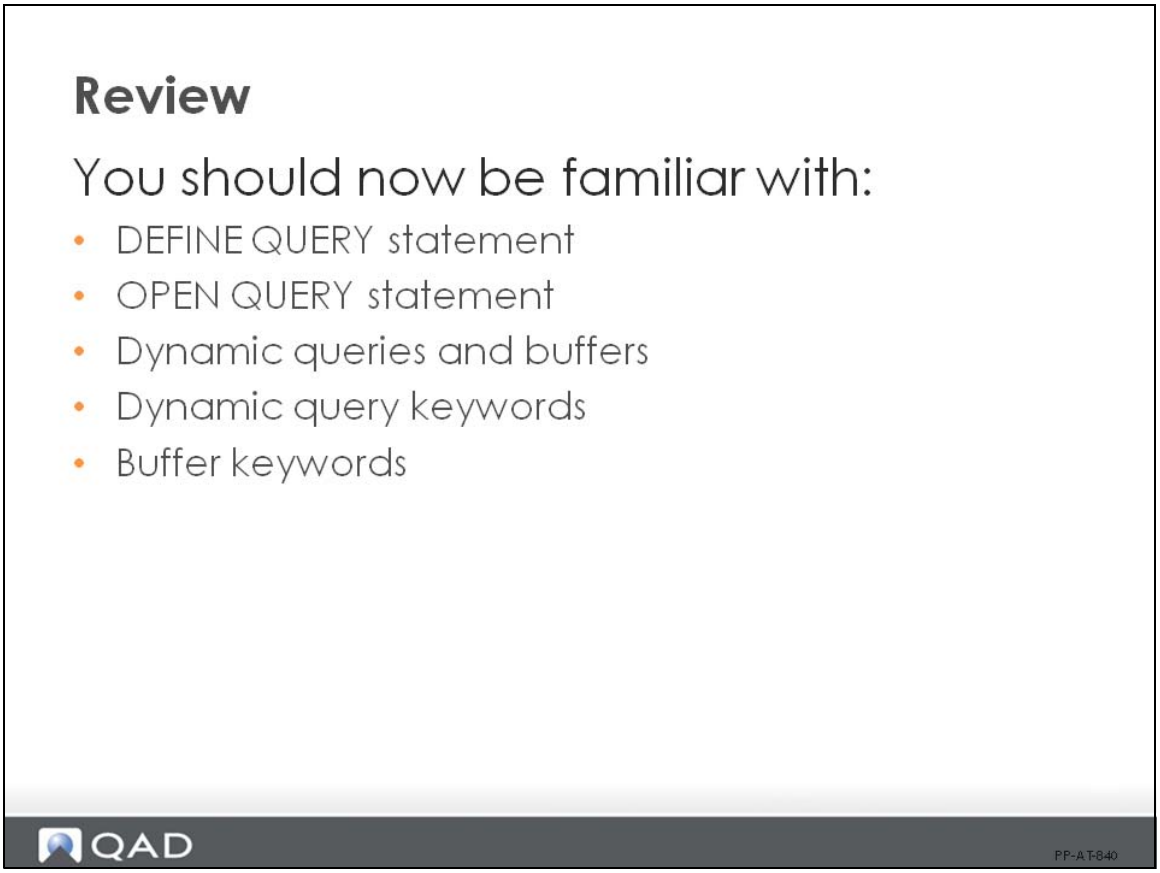
Dynamic Query Keywords



Buffer Keywords



Review


Review
You should now be familiar with:
• DEFINE QUERY statement
• OPEN QUERY statement
• Dynamic queries and buffers
• Dynamic query keywords
• Buffer keywords
QAD
PP-AT-840

Appendix A

Exercise Answers

Overview

This appendix includes answers to the exercise questions, except for the ones where you have simply been asked to change lines of code and rerun an existing program. Use these answers to check your work if you are having difficulty with an exercise.

The training environment includes a folder with executable programs for many of the exercises. It is found in this directory:

```
C:\_OE10\Exercises\Exercise Answers
```

Exercise Answers

Exercise 2-3: Naming Conventions

1 What is the database name of the following tables?

- Customer Master: **cm_mstr**
- Sales Order Master: **so_mstr**
- Sales Order Control: **soc_ctrl**
- Inventory Transaction History: **tr_hist**
- Item Master: p**t_mstr**

2 What is the database name of the following fields in the Item Master table?

- Item Number: **pt_part**
- Description: **pt_desc1 and pt_desc2**
- Revision: **pt_rev**
- Routing Code: **pt_route**
- Group: **pt_group**

Exercise 2-4: Field Definition

1 How big can an item number be? **pt_part, 18 characters**

2 Can an item number only contain numbers? **No**

3 How many decimals are allowed in the price? **pt_price, 10**

4 Is there any restriction on the value for the item's product line? **pt_prod_line Validation, Yes the value must be defined in the pl_mstr table**

5 What values can the Inspection Required field have? **Yes or No**

6 What is the initial value of Issue Policy? **Yes**

Exercise 2-5: Indexes

1 In what order are Sales Order Master records displayed by default?

Hint. What would sales orders logically be ordered by?

so_nbr is the Primary Unique index on the so_mstr table

By Sales Order Domain (so_domain) then Sales Order Number (so_nbr)

2 In what order are the Item Master records displayed by default?

pt_part is the Primary Unique index on the pt_mstr table

By Part Domain (pt_domain) then Part (pt_part)

Exercise 4-3: Display Customer Data

Write a procedure that displays the following:

- Customer number (address)
- Customer sort name
- Current balance
- Region
- Last sale date

Hint. Use table cm_mstr and set cm_domain to 10USA.

```
FOR EACH cm_mstr WHERE cm_domain = "10USA" NO-LOCK:
    DISPLAY cm_addr cm_sort cm_balance cm_region cm_sale_date WITH WIDTH 132.
END.
```

Exercise 5-1: Field Format

1 Open `pc05001.p` and run it without making any changes. Note the unit of measure for part 01011 says EACH.

2 Make the following change and run again:

From: `pt_um  FORMAT "x(4)"  LABEL "Unit"`

To: `pt_um  LABEL "Unit"`

Why is the Unit column four characters wide but part 01011 only shows EA instead of EACH?

The label on the column is still four characters long.

3 Now make this change and run to see the effect:

From: `pt_um  LABEL "Unit"`

To: `pt_um  NO-LABEL`

Why does the unit of measure column still show only two characters?

The column size is using the format from the Data Dictionary of x(2).

4 And finally, make this change and notice the effect:

From: `pt_desc1 pt_desc2  NO-LABEL`

To: `pt_desc1 + pt_desc2`

What happened to the Description column? Why?

The two description fields were strung together but show only the first eight characters from pt_desc1. The concatenation of the two fields results in an implied new character field. A character field has a default format of x(8) unless otherwise overridden. The column has no label because none was specified for this new implied field.

Exercise 5-2: Sorting Records

1 Write a procedure to report location detail records (ld_det) sorted by item number, then site, and then location.

```
FOR EACH ld_det WHERE ld_domain = "10USA" NO-LOCK
BY ld_part BY ld_site by ld_loc:
   DISPLAY ld_part ld_site ld_loc ld_qty_oh.
END.
```

2 Write a procedure to report inventory transactions (tr_hist) sorted by transaction type.

```
FOR EACH tr_hist WHERE tr_domain = "10USA" NO-LOCK
BY tr_type:
   DISPLAY tr_type tr_nbr tr_part tr_qty_chg.
END.
```

Exercise 5-3: Sort using BY

1 Retrieve `pc05003.p` and modify:

From:
```
BY sod_price BY sod_line
```
To:
```
BY sod_price * sod_qty_ord
DESCENDING BY sod_part
```

2 Write a procedure to sort the Sales Order Master table by customer.

Hint. Use so_cust and sod_doman = "10USA".

Display the sold-to customer code, sales order number, order date, and due date.

```
FOR EACH so_mstr WHERE so_domain = "10USA" NO-LOCK
BY so_cust:
   DISPLAY so_cust so_nbr so_ord_date so_due_date.
END.
```

3 Modify the above procedure to sort the sales orders by order date and then by the due date.

Hint. Use so_ord_date and so_due_date.

```
FOR EACH so_mstr WHERE so_domain = "10USA" NO-LOCK
BY so_ord_date BY so_due_date:
   DISPLAY so_cust so_nbr so_ord_date so_due_date.
END.
```

Exercise 5-4: Report Totals

1 Write a procedure to report sales order detail records (sod_det) by item number (sod_part), and give a total of the quantity shipped (sod_qty_ship).

Hint. Use sod_domain = "10USA" in the WHERE clause.

2 Display the following:

- sales order line

- item number
- quantity ordered
- quantity shipped

```
FOR EACH sod_det WHERE sod_domain = "10USA" NO-LOCK
BY sod_part:
   DISPLAY sod_line sod_part sod_qty_ord sod_qty_ship(TOTAL).
END.
```

Exercise 5-5: Sort Breaks and Subtotals

1 Modify the procedure `pc05005.p` to subtotal the quantity ordered for each sales order.

```
FOR EACH sod_det WHERE sod_domain = "10USA" NO-LOCK
BREAK BY sod_nbr BY sod_line:
   DISPLAY sod_nbr sod_line sod_part sod_price sod_qty_ord (TOTAL by sod_nbr)
        sod_price * sod_qty_ord (TOTAL by sod_nbr)
        COLUMN-LABEL "Ext Price" WITH WIDTH 132.
END.
```

2 Write a procedure to report location detail records (ld_det) by site and location.

- Display item number, site, location, quantity on hand, and quantity allocated
- Display subtotal of quantity on hand for each location
  Hint. Use ld_domain = "10USA" in the WHERE clause.

```
FOR EACH ld_det WHERE ld_domain = "10USA" NO-LOCK
BREAK BY ld_site BY ld_loc:
   DISPLAY ld_part ld_site ld_loc ld_qty_oh (TOTAL BY ld_loc) ld_qty_all
      ld_lot WITH WIDTH 132.
END.
```

Exercise 5-6: Sort Break

1 Write a procedure to display the customer number on the first line of a list of sales orders, then SKIP a line after the last order of that customer.

Hint. Use domain "10USA" in the WHERE clauses.

```
FOR EACH so_mstr WHERE so_domain = "10USA"
BREAK BY so_cust:
   IF FIRST-OF(so_cust) THEN
      DISPLAY so_cust.
   DISPLAY so_nbr so_ord_date.
   IF LAST-OF(so_cust) THEN
      DOWN 1.
END.
```

2 Write a procedure to display the different transaction types that have been written to the tr_hist table.

```
FOR EACH tr_hist WHERE tr_domain = "10USA"
BREAK BY tr_type:
   IF FIRST-OF(tr_type) THEN
      DISPLAY tr_type.
END.
```

Exercise 5-7, Question 5

Create a procedure to produce the following display.

Hint. Use the AT or COLON format phrase options to position information on the screen. Use pt_domain = "10USA" in the WHERE clause.

```
                          I T E M S   C A T A L O G

Item Number   :   XXXXXXXXXXXXXXXXXX           Price:   $99.99
Description   :   XXXXXXXXXXXXXXXXXXXXXXX
              :   XXXXXXXXXXXXXXXXXXXXXXX

Item Number   :   XXXXXXXXXXXXXXXXXX           Price:   $99.99
Description   :   XXXXXXXXXXXXXXXXXXXXXXX
              :   XXXXXXXXXXXXXXXXXXXXXXX

Item Number   :   XXXXXXXXXXXXXXXXXX           Price:   $99.99
Description   :   XXXXXXXXXXXXXXXXXXXXXXX
              :   XXXXXXXXXXXXXXXXXXXXXXX
```

```
FOR EACH pt_mstr WHERE pt_domain = "10USA" NO-LOCK:
   DISPLAY
      pt_part  COLON 13
           pt_price COLON 50 FORMAT "$->>>>,>>>,>>9.99<<<" SKIP
           pt_desc1 COLON 13          SKIP
           pt_desc2 COLON 13 LABEL "" SKIP(2)
   WITH FRAME a TITLE "  I T E M S  C A T A L O G  "
      SIDE-LABELS 3 DOWN.
END.
```

Exercise 6-1: Retrieving Specific Records

1 Modify procedure `pc06001.p`:

From: `pt_price = 0`
To: `pt_price <> 0`

From: `pt_price <> 0 AND pt_pm_code = "P"`
To: `pt_price = 0 AND pt_pm_code = "M"`

2 Modify this procedure to display only items in the product line 10.

Hint. You must enclose character constants within quotation marks.

```
FOR EACH pt_mstr NO-LOCK WHERE pt_domain = "10USA"
   AND pt_prod_line = "10":
   DISPLAY pt_part pt_desc1 pt_price pt_group WITH WIDTH 132.
END.
```

Exercise 6-2: BEGINS and MATCHES

1 Modify the procedure `pc06002.p` and run:

From: `BEGINS "01"`
To: `BEGINS "0"`

From: `BEGINS "0"`
To: `MATCHES "*01*"`

2 Often companies have the same or similar item description with very different item numbers. A useful application of the MATCHES function is to display all items with a certain word in their description.

Modify the procedure to display all items with the word “PACK” in their description.

```
FOR EACH pt_mstr NO-LOCK WHERE pt_domain = "10USA"
   AND pt_desc1 MATCHES "*PACK*":
   DISPLAY pt_part
           pt_desc1 + " " + pt_desc2 FORMAT "x(48)"
           LABEL "Description" WITH WIDTH 80.
END.
```

Exercise 6-3: Conditional Expressions

Evaluate the following statements as True or False. Use the following values for the respective field names:

```
pt_run = 2          pt_setup = 1   pt_sfty_time = 3
```

1. `pt_run > 0 AND (pt_run > pt_setup AND pt_run > pt_sfty_time)` **False**
2. `pt_run > 0 AND (pt_run > pt_setup OR pt_run > pt_sfty_time)` **True**
3. `pt_run LE 0 OR pt_setup < pt_sfty_time` **True**
4. `(pt_run > 0 AND pt_run > pt_setup) OR pt_run > pt_sfty_time` **True**
5. `((pt_run > 0 and pt_run < pt_setup) AND (pt_setup < pt_sfty_time)) AND pt_run > pt_sfty_time` **False**

Exercise 6-4: Using Indexes

Modify the procedure `pc06003.p` to sort the output by item number.

```
FOR EACH tr_hist NO-LOCK WHERE tr_domain = "10USA" AND tr_type = "ISS-SO"
USE-INDEX tr_part_eff:
   DISPLAY tr_part tr_effdate tr_qty_chg tr_so_job
           tr_type WITH WIDTH 132.
END.
```

Exercise 6-5: IF...THEN...ELSE

Procedure `pc06004.p` highlights items that do not belong to a group.

1. Modify this procedure to highlight those items with a price = 0.

```
FOR EACH pt_mstr WHERE pt_domain = "10USA" NO-LOCK:
   IF pt_price = 0 THEN
      DISPLAY pt_part pt_desc1 pt_prod_line "***" WITH WIDTH 132.
   ELSE
      DISPLAY pt_part pt_desc1 pt_prod_line pt_price WITH WIDTH 132.
END.
```

2. Modify the procedure by deleting the ELSE statement.

```
FOR EACH pt_mstr WHERE pt_domain = "10USA" NO-LOCK:
   IF pt_price = 0 THEN
      DISPLAY pt_part pt_desc1 pt_prod_line "***" WITH WIDTH 132.
END.
```

3 Add a title for this inquiry.

```
FOR EACH pt_mstr WHERE pt_domain = "10USA" NO-LOCK:
   IF pt_price = 0 THEN
      DISPLAY pt_part pt_desc1 pt_prod_line "***" WITH WIDTH 132
         TITLE "Items With No Price".
END.
```

Exercise 6-6: PROMPT-FOR

1 Modify the procedure `pc06006.p`:

From: `USING pt_prod_line`
To: `WHERE pt_prod_line = INPUT pt_prod_line`

2 Modify this procedure to use item number input to create the same inquiry.

```
PROMPT-FOR pt_part.
FOR EACH pt_mstr NO-LOCK WHERE pt_domain = "10USA" AND pt_part = INPUT pt_part:
   DISPLAY pt_part pt_desc1 + " " + pt_desc2 FORMAT "x(40)" LABEL "Description"
   pt_price WITH TITLE "Item Number: " + pt_part WIDTH 132.
END.
```

3 Write a procedure to input a location, then report all of the items and quantities on hand in that location.

Hint. Use ld_det and remember to set ld_domain = "10USA" in the WHERE clause.

```
PROMPT-FOR ld_loc.
FOR EACH ld_det NO-LOCK WHERE ld_domain = "10USA" AND ld_loc = INPUT ld_loc:
   DISPLAY ld_part ld_site ld_lot ld_qty_oh WITH TITLE "Location: " + ld_loc.
END.
```

Exercise 7-1: FIND

1 Modify procedure `pc07001.p`. Notice that you can position the NO-LOCK phrase anywhere.

From:
```
FIND pt_mstr NO-LOCK WHERE pt_domain = "10USA"
AND pt_part = INPUT pt_part.
```
To:
```
FIND pt_mstr WHERE pt_domain = "10USA"
AND pt_part = INPUT pt_part NO-LOCK.
```

From:
```
 FIND pt_mstr WHERE pt_domain = "10USA"
AND pt_part = INPUT pt_part NO-LOCK.
```
To:
```
FIND pt_mstr WHERE pt_domain = "10USA"
AND pt_part > INPUT pt_part NO-LOCK.
```

Hint. Enter 01010 when prompted for an item.

2 Modify `pc07001.p` to look for item 04-5025, which does not exist in the database. What happens?

A Progress error appears.

3 Create a program that prompts for a customer number and then displays the customer name and address information.

Hint. Use ad_mstr and remember to use ad_domain = "10USA" in the WHERE clause.

Note The field labeled Address (ad_addr) contains the customer and supplier number, not their physical location.

```
PROMPT-FOR ad_addr LABEL "Customer" COLON 20 WITH SIDE-LABELS WIDTH 80.

FIND ad_mstr NO-LOCK WHERE ad_domain = "10USA" AND ad_addr = INPUT ad_addr.
DISPLAY
   ad_name  NO-LABEL
   ad_line1 LABEL "Street" COLON 20
   ad_city                 COLON 20
   ad_state ad_zip
WITH WIDTH 80.
```

Exercise 7-2: AVAILABLE

1 Modify procedure `pc07002.p`.

From: `MESSAGE "Item" + INPUT pt_part + " Not Found".`
To: `MESSAGE "Item" + pt_part + " Not Found".`

2 Run the modified program with a valid item (01010).

3 Run the program with an invalid item (your initials).

4 Why did removing INPUT from pt_part cause a Progress error instead of the message?

A Progress error occurred because no pt_mstr record was found. The MESSAGE statement is trying to display something from an empty record buffer. The MESSAGE statement needs to reference the INPUT buffer for pt_part to avoid the Progress error.

5 Modify the program as follows and rerun first with a valid and then an invalid item.

From: `ELSE DO:`
To: `IF NOT AVAILABLE pt_mstr THEN DO:`

Exercise 7-5: User-Defined Variables

1 Note any advantages or differences between using this:

```
"UPDATE part zero-ship"
```

Or this:

```
"PROMPT-FOR part zero-ship".
```

2 (Optional) Modify the program `pc07006.p` to round the value of ext-margin using the ROUND function.

Hint. ROUND (expression, precision)

expression = ext-margin, precision = the number of decimal places for rounding

```
DEFINE VARIABLE part LIKE pt_part NO-UNDO.
DEFINE VARIABLE zero-ship AS LOGICAL INITIAL yes NO-UNDO
       LABEL "Suppress Line with Zero Parts Shipped".
DEFINE VARIABLE ext-margin AS DECIMAL FORMAT "$->>>,>>9.999"
       LABEL "Ext Margin" NO-UNDO.

REPEAT:
   UPDATE part zero-ship WITH SIDE-LABELS WIDTH 80.
   FOR EACH sod_det NO-LOCK WHERE sod_domain = "10USA"
      AND sod_part = part OR part = "":
      IF zero-ship = NO OR (zero-ship = YES AND sod_qty_ship <> 0)
      THEN DO:
        ext-margin = sod_qty_ord * (sod_price - sod_std_cost).
        DISPLAY sod_nbr sod_line sod_price sod_qty_ord
              ROUND(ext-margin, 2) LABEL "Ext Margin" sod_qty_ship
          WITH WIDTH 132 TITLE "SO Margin Inquiry".
      END.
   END.
END.
```

3 Write a procedure that uses user-defined variables and the UPDATE statement. Ask a user for a location. Then list all of the items in that location.

Hints. Use the location detail table (ld_det) to find the items in a location. Display the lot number, item number, and quantity on hand.

Remember to use ld_domain = "10USA" in the WHERE clause. When prompted for location, specify 1000.

```
DEFINE VARIABLE loc LIKE ld_loc NO-UNDO.

UPDATE loc.
FOR EACH ld_det NO-LOCK WHERE ld_domain = "10USA" AND ld_loc = loc:
   DISPLAY ld_lot ld_part ld_qty_oh.
END.
```

Exercise 8-1: Entity Relationships

1 Review the Customer Master relationships in the *QAD EA Entity Diagrams* manual.

2 How many purchase order lines (pod_det) records can a purchase order (po_mstr) have?

Zero, one or multiple

Exercise 8-2: FIND/FIND

Write a program that prompts the user for a sales order number. Then display the order date, customer number, customer name, and address. Use the following hints:

- In the Sales Order Master table (so_mstr) display the so_ord_date field.
- Use 10USA as a valid domain.
- Link the tables using the so_cust and ad_addr fields.
- Locate the address in the Address Master table ad_mstr and display the fields ad_name and ad_line1.
- When prompted for a sales order, specify SO011204.

```
DEFINE VARIABLE nbr LIKE so_nbr NO-UNDO.

REPEAT:
    UPDATE nbr WITH WIDTH 80 SIDE-LABELS.
    FIND so_mstr NO-LOCK WHERE so_domain = "10USA" AND so_nbr = nbr NO-ERROR.
    IF AVAILABLE so_mstr THEN DO:
      DISPLAY so_ord_date so_cust.
      FIND ad_mstr NO-LOCK WHERE ad_domain = so_domain
         AND ad_addr = so_cust NO-ERROR.
    IF AVAILABLE ad_mstr THEN
        DISPLAY
            ad_name  NO-LABEL AT 12
            ad_line1          COLON 10
            ad_city + ", " + ad_state + " " + ad_zip FORMAT "x(40)" AT 12
            SKIP(1).
   END.
   ELSE
      MESSAGE "The Sales Order Number Entered was not found".
END.
```

Exercise 8-3: FIND, FOR EACH

1 Write a new procedure that:

- Prompts for an item number
- Displays the item description and product line number (pt_desc1 and pt_prod_line)

Hint. Remember to use "10USA" as a valid domain in the WHERE clause. Item 01010 is a valid item number for this exercise.

```
DEFINE VARIABLE part LIKE pt_part NO-UNDO.

REPEAT:
    UPDATE part WITH SIDE-LABELS WIDTH 80.
    FIND pt_mstr NO-LOCK WHERE pt_domain = "10USA" AND pt_part = part NO-ERROR.
    IF AVAILABLE pt_mstr THEN
        DISPLAY pt_desc1 pt_prod_line.
    ELSE MESSAGE "Part not found".
END.
```

2 Then display sales order line item information, sales order number, sales order line number, quantity ordered, and order price (sod_nbr sod_line sod_qty_ord sod_price).

```
DEFINE VARIABLE part LIKE pt_part NO-UNDO.

REPEAT:
    UPDATE part WITH SIDE-LABELS WIDTH 80.
    FIND pt_mstr NO-LOCK WHERE pt_domain = "10USA" AND pt_part = part NO-ERROR.
    IF AVAILABLE pt_mstr THEN DO:
        DISPLAY pt_desc1 pt_prod_line.
        FOR EACH sod_det WHERE sod_domain = pt_domain
            AND sod_part = pt_part NO-LOCK:
            DISPLAY sod_nbr sod_line sod_qty_ord sod_price.
        END.
    END.
    ELSE MESSAGE "Part not found".
END.
```

Exercise 8-4: Outer Join

Write a program that lists:

- The customer name, region, and last sale date
- Then list all sales orders along with the items, quantities sold, and prices of these items

Hint. Set the domain to "10USA". You can use customer 10C1001 to test this program.

```
FOR EACH cm_mstr WHERE cm_domain = "10USA" NO-LOCK:
    DISPLAY cm_addr cm_sort cm_region cm_sale_date WITH WIDTH 80.
    FOR EACH so_mstr WHERE so_domain = cm_domain
        AND so_cust = cm_addr NO-LOCK:
        DISPLAY so_nbr WITH WIDTH 80.
        FOR EACH sod_det WHERE sod_domain = so_domain
            AND sod_nbr = so_nbr NO-LOCK:
            DISPLAY
                sod_line
                sod_part
                sod_qty_ord
                sod_price
                (sod_qty_ord * sod_price) LABEL "Sales Amount"
            WITH WIDTH 132.
        END.   /* for each sod_det */
    END.   /* for each so_mstr */
END.   /* for each cm_mstr */
```

Exercise 8-5: Inner Join

1 Modify procedure `pc08005.p` to sort by sales order date.

Note When you run the program, use 10C1000 for Customer and 10C1001 for To Customer.

```
DEFINE VARIABLE cust  LIKE so_cust NO-UNDO.
DEFINE VARIABLE cust1 LIKE so_cust LABEL "To" NO-UNDO.
DEFINE VARIABLE desc1 LIKE pt_desc1 NO-UNDO.

REPEAT:
    UPDATE cust cust1 WITH SIDE-LABELS WIDTH 80.

    FOR EACH so_mstr NO-LOCK WHERE so_domain = "10USA" AND
        so_cust >= cust AND so_cust <= cust1,
    EACH sod_det NO-LOCK
        WHERE sod_domain = so_domain AND sod_nbr = so_nbr
        BY so_ord_date WITH WIDTH 132:
        DISPLAY so_cust so_nbr sod_part sod_qty_ord so_ord_date.
    END.
END.
```

2 Write a procedure that displays purchase order detail information (pod_det) sorted by supplier.

Hint. Supplier is field po_vend. Set pod_domain = "10USA" in the WHERE clause.

```
FOR EACH po_mstr NO-LOCK WHERE po_domain = "10USA",
EACH pod_det NO-LOCK WHERE pod_domain = po_domain AND pod_nbr = po_nbr
BY po_vend:
    DISPLAY po_vend po_nbr pod_part pod_qty_ord po_ord_date WITH WIDTH 132.
END.
```

3 Write a procedure that uses user-defined variables and the UPDATE statement to ask a user for a location; then list all items in that location.

Hint. Use the location detail table (ld_det) to find items in a location. Display the lot number, item number, and quantity on hand. Remember to set ld_domain = "10USA" in the WHERE clause.

```
DEFINE VARIABLE loc LIKE ld_loc NO-UNDO.

REPEAT:
    UPDATE loc WITH WIDTH 80.
    FOR EACH ld_det NO-LOCK WHERE ld_domain = "10USA" AND ld_loc = loc:
        DISPLAY ld_part ld_qty_oh ld_lot WITH WIDTH 80.
    END.
END.
```

Exercise 9-3: Framing Properties

1 What is the effect of specifying ROW 4?

The ROW phrase indicates the row number on the screen where the output from the associated DISPLAY statement should be positioned. If that screen row is already occupied, Progress uses the next available screen row.

Exercise 9-4: FORM

1 Modify procedure `pc09004.p` and add so_ship to the DISPLAY statement.

2 Modify this procedure so that it also displays the following sales order detail information: line number, item number, item price, order quantity, and quantity shipped.

```
FOR EACH so_mstr WHERE so_domain = "10USA" NO-LOCK WITH FRAME a:
    FIND FIRST ad_mstr WHERE ad_domain = so_domain
        AND ad_addr = so_cust NO-LOCK NO-ERROR.

    FORM HEADER "Sales Order Inquiry"
        so_nbr          COLON 15
        so_cust         COLON 15  ad_name NO-LABEL               AT 28
        so_ord_date     COLON 15  so_req_date LABEL "Req Date" COLON 36
        so_cr_terms     COLON 15
        SKIP(1)
    WITH SIDE-LABELS WIDTH 132 FRAME a.

    DISPLAY so_nbr so_cust so_ord_date so_req_date so_cr_terms.
    IF AVAILABLE ad_mstr THEN DISPLAY ad_name.
    FOR EACH sod_det WHERE sod_domain = so_domain
        AND sod_nbr = so_nbr NO-LOCK:
        DISPLAY sod_line sod_part sod_price sod_qty_ord sod_qty_ship
        WITH WIDTH 132.
    END.
END.
```

3 Notice the difference between HEADER and TITLE.

TITLE displays on the box. HEADER displays in main part of frame; used if no box (to save space).

Exercise 9-5: FORM Placement

1 Modify procedure `pc09005.p`:

From: `WITH SIDE-LABELS WIDTH 80 FRAME a 3 DOWN.`
To: `WITH SIDE-LABELS WIDTH 80 FRAME a.`

2 Comment out the DOWN 1 statement.

3 Rename the frame for the FOR EACH statement. Does this produce a better result?

4 Describe the easiest way to avoid the problems illustrated by these changes.

Scope the form statement to the block that modifies it.

Exercise 9-6: Flashing

Revise procedure `pc09006.p` to prevent the records from flashing.

Hint. Use customer 10C1001 when testing the program.

```
REPEAT:
    PROMPT-FOR so_cust WITH FRAME A.
    FIND FIRST cm_mstr NO-LOCK WHERE cm_domain = "10USA"
        AND cm_addr = INPUT so_cust NO-ERROR.
    IF AVAILABLE cm_mstr THEN DO:
        FOR EACH so_mstr NO-LOCK WHERE so_domain = cm_domain
            AND so_cust = INPUT so_cust WITH FRAME A:
                DISPLAY so_cust so_nbr so_ord_date so_po WITH FRAME A DOWN.
            DOWN 1.
        END.
    END.
    ELSE DO:
        MESSAGE " Invalid Customer - Please Reenter ".
        UNDO, RETRY.
    END.
END.
```

Exercise 11-2: Shared Variables

1 Create a new procedure called `main1.p` that:

- Asks the user to UPDATE an integer type variable.
- Then displays the value of the variable.
- Then call subprocedure `upd2.p`.
- After running `upd2.p`, redisplay the value of the variable in a different frame.

```
DEFINE  NEW SHARED VARIABLE numberx AS INTEGER NO-UNDO.

UPDATE  numberx LABEL "INITIAL NUMBER"
    WITH FRAME num1 SIDE-LABELS.
DISPLAY numberx LABEL "NEW  NUMBER" WITH FRAME num2 SIDE-LABELS.
RUN upd2.p.
DISPLAY numberx LABEL "FINAL NUMBER" WITH FRAME num3 SIDE-LABELS.
```

2 Create a new program called `upd2.p` that:

- Takes in a numeric variable passed to it.
- Adds the value of ten to the variable.

```
DEFINE SHARED VARIABLE numberx AS INTEGER NO-UNDO.

numberx = numberx + 10.
```