Oracle Forms Developer 10*g*: Build Internet Applications

Instructor Guide • Volume 3

D17251GC10

Edition 1.0

June 2004

D39560

Author

Pam Gamer

Technical Contributors and Reviewers

Alena Bugarova
Purjanti Chang
Laurent Dereac
Punita Handa
Mark Pare
Jasmin Robayo
Bryan Roberts
Divya Sandeep
Raza Siddiqui
John Soltani
Lex van der Werff

Editors

Nishima Sachdeva
Elizabeth Treacy

Publisher

Giri Venugopal

Contents

Preface

Profile

Before you begin this course

Before you begin this course, you should be able to:

- Create SQL statements.
- Create PL/SQL constructs, including conditional statements, loops, procedures and functions.
- Create PL/SQL stored (server) procedures, functions, and packages.
- Use a graphical user interface (GUI).
- Use a Web browser.

Prerequisites

Either

- Oracle Database 10*g*: SQL Fundamentals I
- or the following CBT Library:
  - Oracle SQL: Basic SELECT statements
  - Oracle SQL: Data Retrieval Techniques
  - Oracle SQL: DML and DDL
- or Introduction to Oracle Database 10*g* for Experienced SQL Users (InClass)
- or Oracle Database 10*g*: Introduction to SQL (InClass)

And either

- Oracle Database 10*g*: Program with PL/SQL (InClass)
- or the following CBT Library:
  - PL/SQL: Basics
  - PL/SQL: Procedures, Functions, and Packages
  - PL/SQL: Database Programming
- Or both:
  - Oracle Database 10*g*: PL/SQL Fundamentals (InClass)
  - Oracle Database 10*g*: Develop PL/SQL Program Units (InClass)

Suggested prerequisites

- Oracle Database 10*g*: SQL Fundamentals II (InClass) (if you attended the Oracle Database 10*g*: SQL Fundamentals I (InClass))
- Oracle Database 10*g*: Advanced PL/SQL (InClass)
- Oracle Forms Developer 10*g*: Move to the Web (eStudy)

How this course is organized

*Oracle Forms Developer 10*g*: Build Internet Applications* is an instructor-led course featuring lecture and hands-on exercises. Online demonstrations and written practice sessions reinforce the concepts and skills introduced.

Related Publications

Oracle publications

Title	Part Number
*Oracle Forms Developer, Release 6*i*: Getting Started (Windows 95/NT)*	A73154-01
*Oracle Forms Developer and Reports Developer, Release 6*i*: Guidelines for Building Applications*	A73073-02
Oracle Application Server Forms Services Deployment Guide 10g (9.0.4)	B10470-01

Additional publications

Release notes: `<ORACLE_HOME\doc\welcome\release_notes\chap_forms.htm`

Typographic Conventions

Typographic conventions in text

Convention	**Element**	**Example**
Bold italic	Glossary term (if there is a glossary)	The ***algorithm*** inserts the new key.
Caps and lowercase	Buttons, check boxes, triggers, windows	Click the Executable button. Select the Can't Delete Card check box. Assign a When-Validate-Item trigger to the ORDERS block. Open the Master Schedule window.
Courier new, case sensitive (default is lowercase)	Code output, directory names, filenames, passwords, pathnames, URLs, user input, usernames	Code output: `debug.set ('I", 300);` Directory: `bin` (DOS), `$FMHOME` (UNIX) Filename: Locate the `init.ora` file. Password: User `tiger` as your password. Pathname: Open `c:\my_docs\projects` URL: Go to `http://www.oracle.com` User input: Enter `300` Username: Log on as `scott`
Initial cap	Graphics labels (unless the term is a proper noun)	Customer address (*but* Oracle Payables)
Italic	Emphasized words and phrases, titles of books and courses, variables	Do *not* save changes to the database. For further information, see *Oracle7 Server SQL Language Reference Manual.* Enter `user_id@us.oracle.com`, where *user_id* is the name of the user.
Quotation marks	Interface elements with long names that have only initial caps; lesson and chapter titles in cross-references	Select "Include a reusable module component" and click Finish. This subject is covered in Unit II, Lesson 3, "Working with Objects."
Uppercase	SQL column names, commands, functions, schemas, table names	Use the SELECT command to view information stored in the LAST_NAME column of the EMP table.

Typographic Conventions (continued)

Typographic conventions in text (continued)

Convention	Element	Example
Right arrow	Menu paths	Select File > Save.
Brackets	Key names	Press [Enter].
Commas	Key sequences	Press and release keys one at a time: [Alternate], [F], [D]
Plus signs	Key combinations	Press and hold these keys simultaneously: [Ctrl]+[Alt]+[Del]

Typographic conventions in code

Convention	Element	Example
Caps and lowercase	Oracle Forms triggers	`When-Validate-Item`

Lowercase	Column names, table names	`SELECT last_name`
		`FROM s_emp;`

	Passwords	`DROP USER scott`
		`IDENTIFIED BY tiger;`

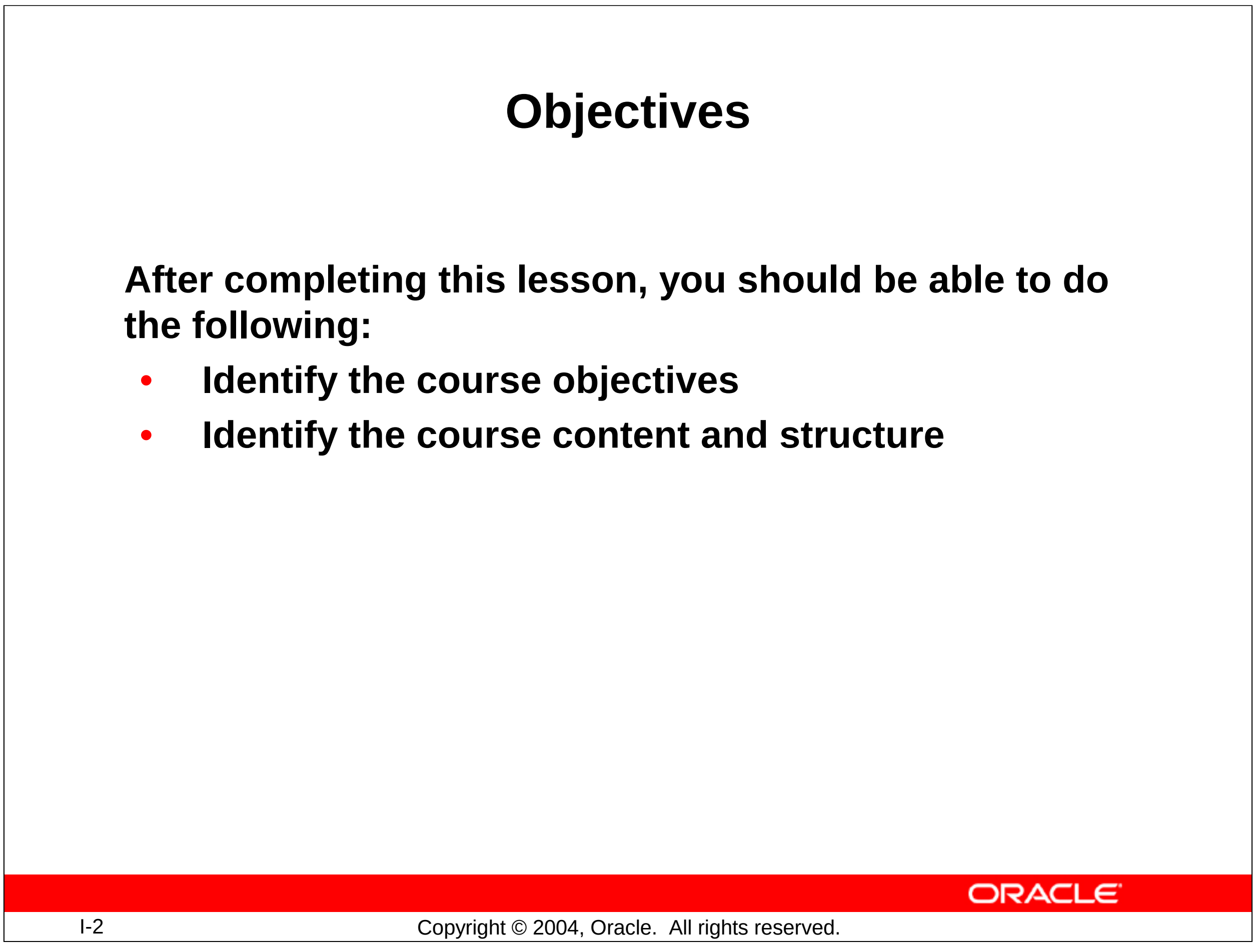
	PL/SQL objects	`OG_ACTIVATE_LAYER`
		`  (OG_GET_LAYER (‘prod_pie_layer’))`

Lowercase italic	Syntax variables	`CREATE ROLE` *`role`*

Uppercase	SQL commands and functions	`SELECT userid`
		`FROM emp;`

Introduction

Schedule:	**Timing**	**Topic**
	15 minutes	Lecture
	15 minutes	Total



Objectives

After completing this lesson, you should be able to do the following:

- **Identify the course objectives**
- **Identify the course content and structure**

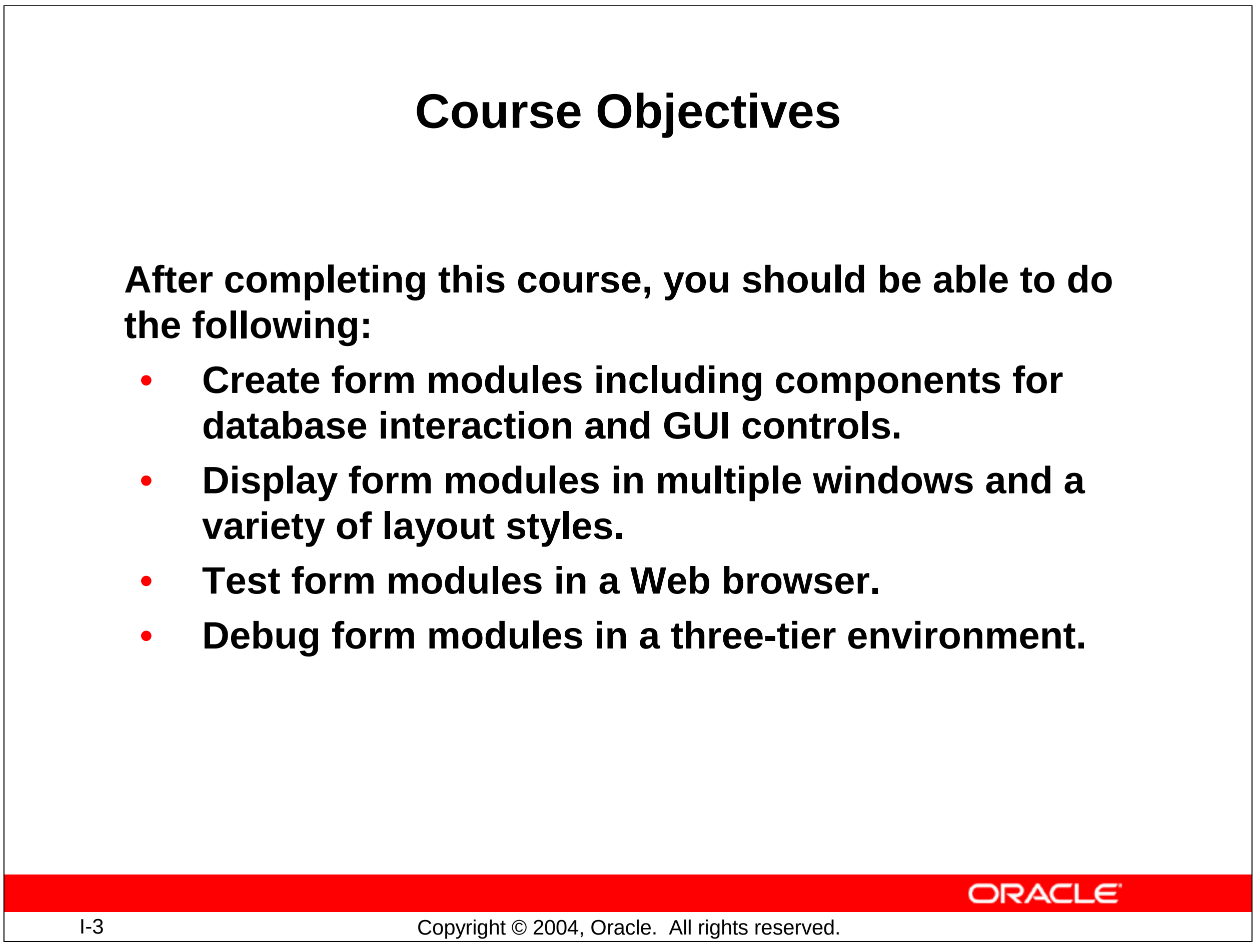
I-2

Introduction

Overview

This lesson introduces you to the *Oracle Forms Developer 10*g*: Build Internet Applications* course:

- The objectives that the course intends to meet
- The topics that it covers
- How the topics are structured over the duration of the course



Course Objectives

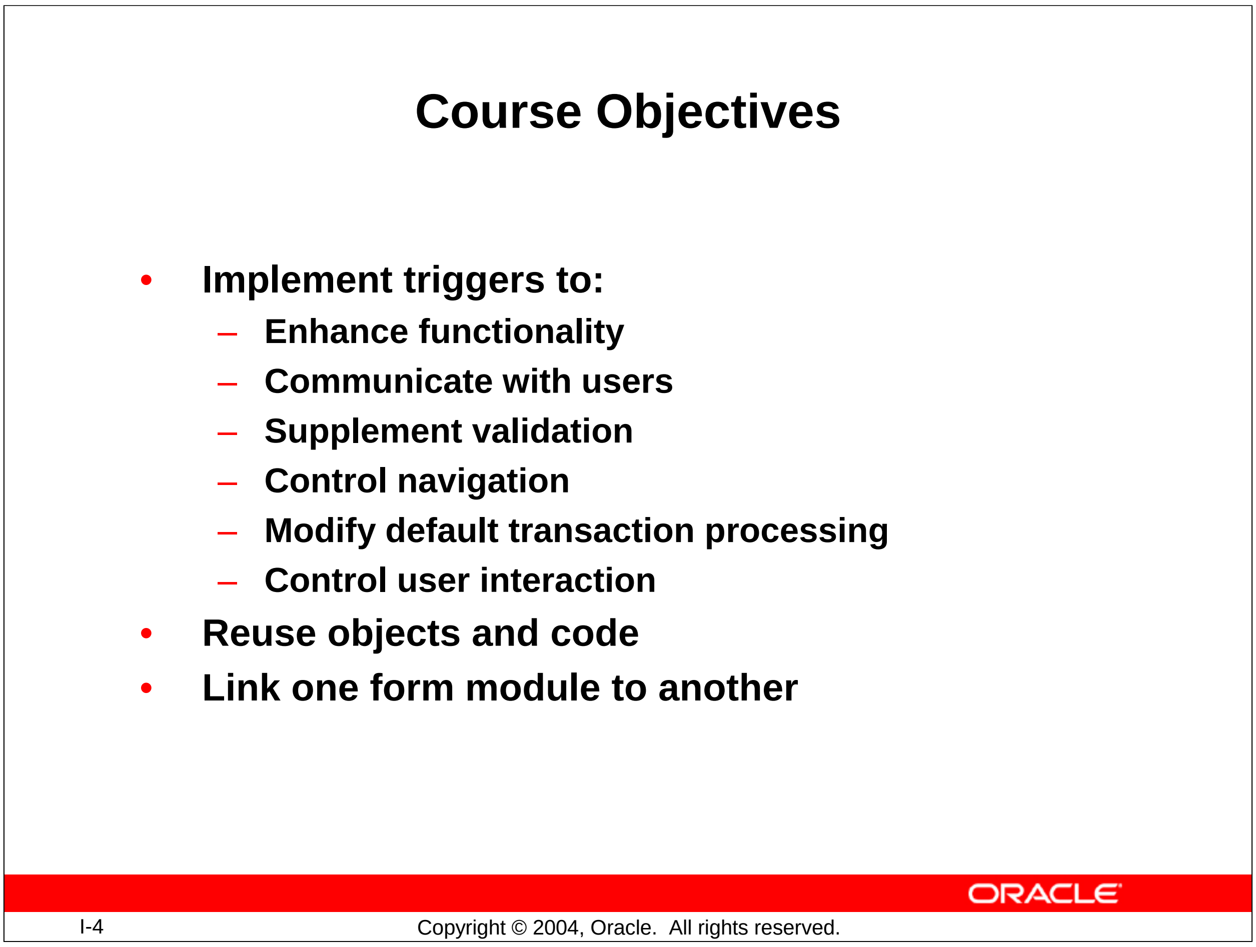
After completing this course, you should be able to do the following:

- Create form modules including components for database interaction and GUI controls.
- Display form modules in multiple windows and a variety of layout styles.
- Test form modules in a Web browser.
- Debug form modules in a three-tier environment.

Course Objectives

Course Description

In this course, you will learn to build, test, and deploy interactive Internet applications. Working in a graphical user interface (GUI) environment, you will learn how to create and customize forms with user input items such as check boxes, list items, and radio groups. You will also learn how to modify data access by creating event-related triggers, and you will display Forms elements and data in multiple canvases and windows.

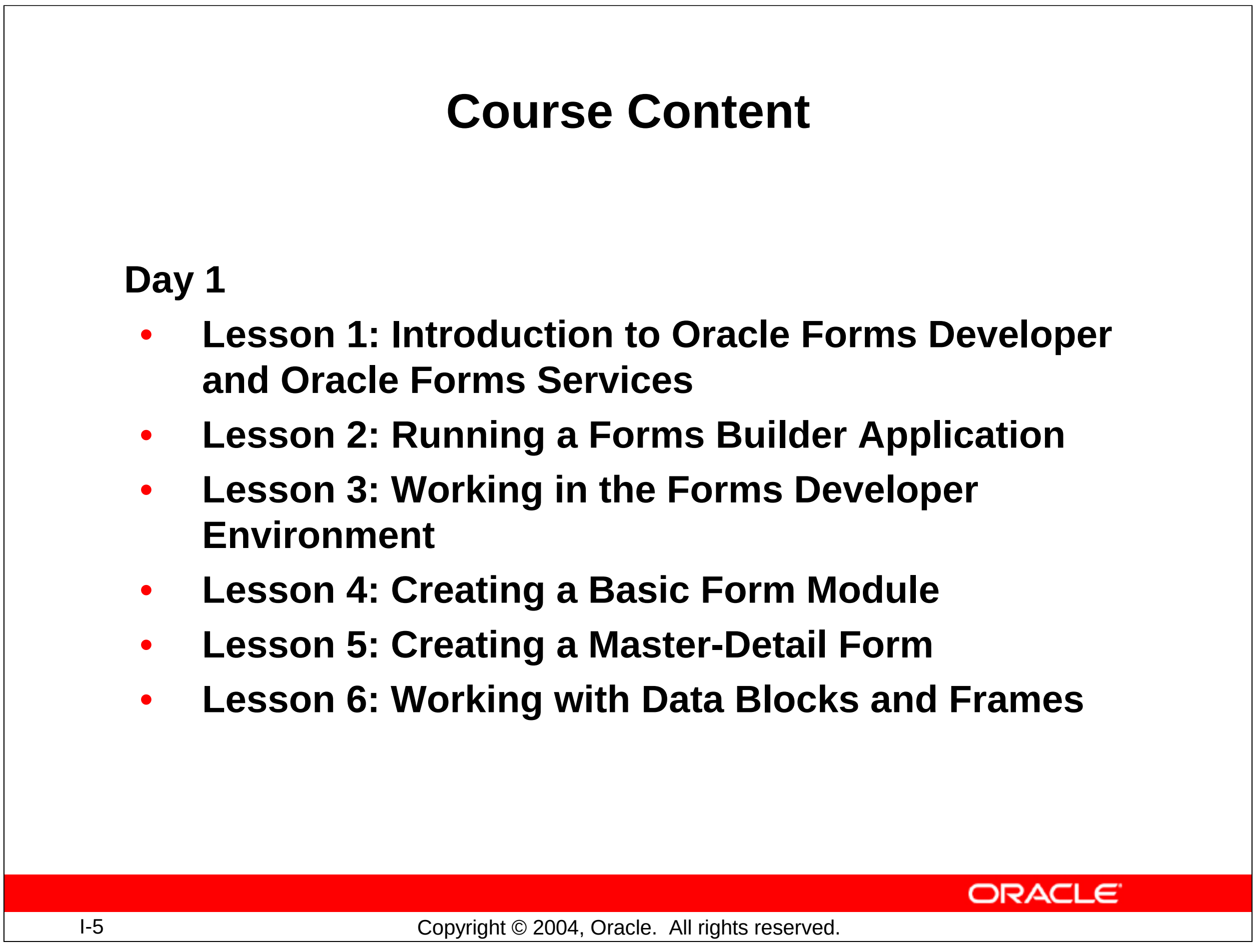


Course Objectives

- Implement triggers to:
  - Enhance functionality
  - Communicate with users
  - Supplement validation
  - Control navigation
  - Modify default transaction processing
  - Control user interaction
- Reuse objects and code
- Link one form module to another

ORACLE

I-4



Course Content

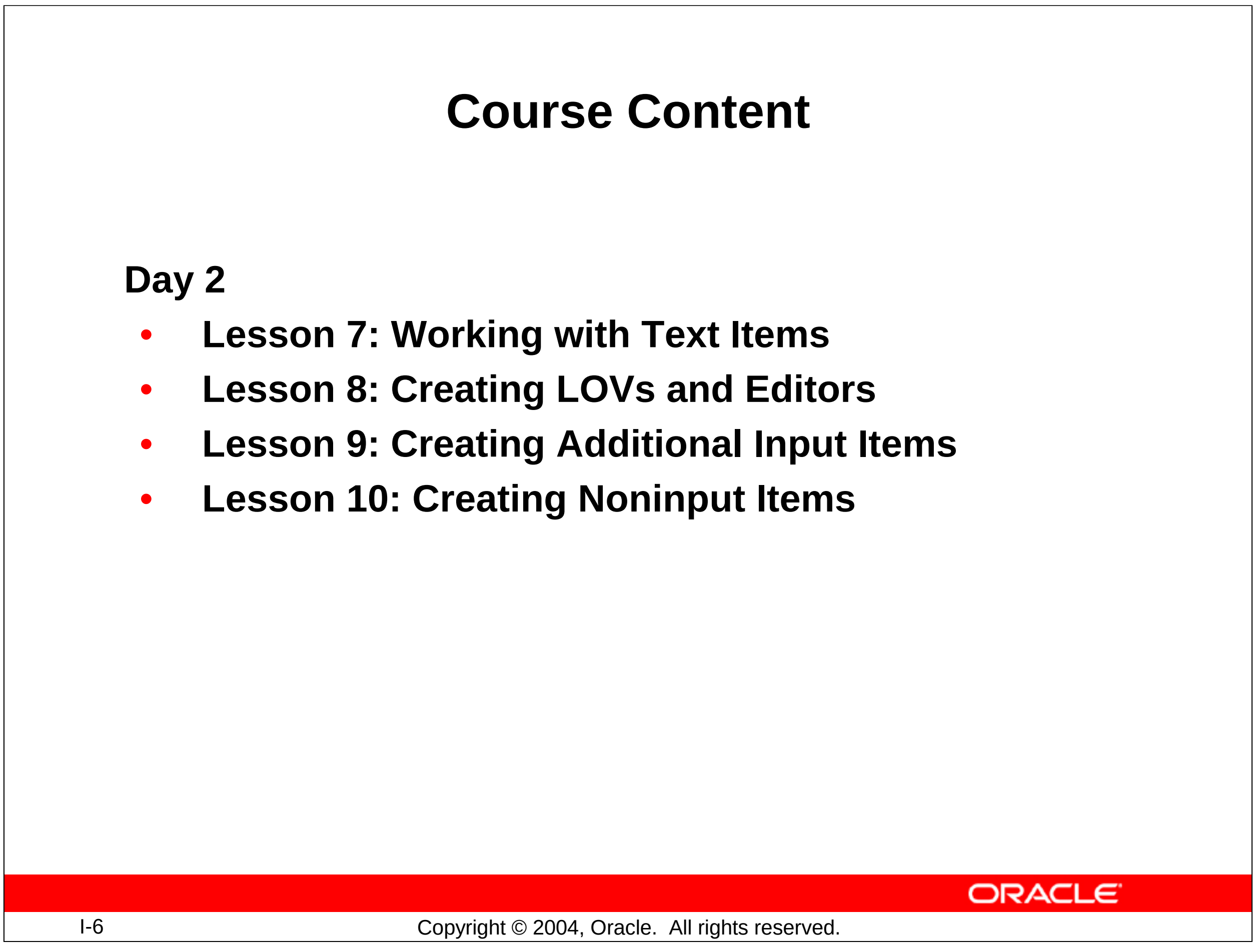
Day 1

- Lesson 1: Introduction to Oracle Forms Developer and Oracle Forms Services
- Lesson 2: Running a Forms Builder Application
- Lesson 3: Working in the Forms Developer Environment
- Lesson 4: Creating a Basic Form Module
- Lesson 5: Creating a Master-Detail Form
- Lesson 6: Working with Data Blocks and Frames

ORACLE

Course Content

The lesson titles show the topics that is covered in this course, and the usual sequence of lessons. However, the daily schedule is an estimate, and may vary for each class.



Course Content

Day 2

- **Lesson 7: Working with Text Items**
- **Lesson 8: Creating LOVs and Editors**
- **Lesson 9: Creating Additional Input Items**
- **Lesson 10: Creating Noninput Items**

ORACLE

I-6

Course Content

Day 3

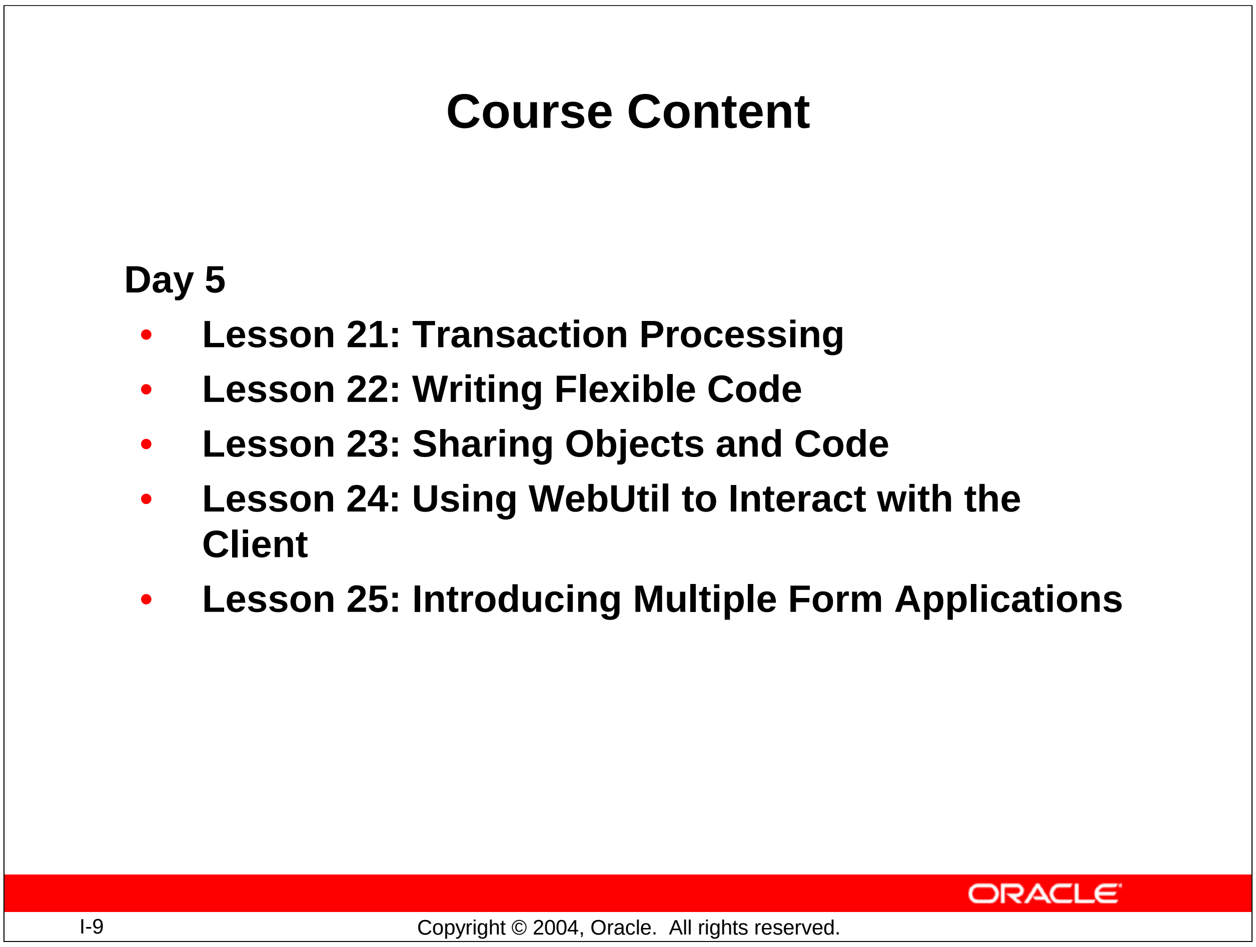
- **Lesson 11: Creating Windows and Content Canvases**
- **Lesson 12: Working with Other Canvas Types**
- **Lesson 13: Introduction to Triggers**
- **Lesson 14: Producing Triggers**
- **Lesson 15: Debugging Triggers**

ORACLE

Course Content

Day 4

- **Lesson 16: Adding Functionality to Items**
- **Lesson 17: Run-time Messages and Alerts**
- **Lesson 18: Query Triggers**
- **Lesson 19: Validation**
- **Lesson 20: Navigation**



Course Content

Day 5

- Lesson 21: Transaction Processing
- Lesson 22: Writing Flexible Code
- Lesson 23: Sharing Objects and Code
- Lesson 24: Using WebUtil to Interact with the Client
- Lesson 25: Introducing Multiple Form Applications

I-9

Practice Solutions

Practice 2 Solutions

1. Start an instance of OC4J.
   Double-click the desktop shortcut labeled Start OC4J Instance. You can minimize the window when it displays the message: Oracle Application Server Containers for J2EE 10g (9.0.4.0.0) initialized
2. Invoke Internet Explorer and enter the following URL:
   http://<machine>:<port>/forms90/f90servlet?form=customers.fmx
   Your instructor will tell you the machine name and port number to use, as well as the username, password, and database for connection.
   No formal solution.
3. Select Help > Keys from the menu.
   No formal solution.
5. Click OK to close the Keys window. Browse through the records that were returned by the unrestricted query that executed automatically when the form started.
   Select Query > Execute, or press [Ctrl]+[F11], or click Execute Query.
   Press [Up] and [Down] to browse through the records returned.
6. Execute a restricted query to retrieve information about the customer with the ID of 212.
   Put the form module in Enter-Query mode (press [F11] or select Query > Enter from the menu or click Enter Query).
   Notice that the status line displays the mode Enter-Qu… (for Enter-Query mode).
   Move to the Customer: ID item and enter the search value 212.
   Execute the query (press [Ctrl]+[F11] or select Query > Execute from the menu or click Execute Query).
   Notice that only one record is retrieved.
7. Try each of these restricted queries:
   a. Retrieve all cities starting with San.
      Select Query > Enter.
      Click the Contact Information Tab.
      Type San% in the City item.
      Select Query > Execute.
   b. Retrieve all customers based in the USA with a low credit limit.
      Select Query > Enter.
      Click the Contact Information Tab.
      Type US in the Country_Id item.
      Click the Account Information Tab.
      Select the Low credit limit.
      Select Query > Execute.
8. Display the customer details for Harrison Sutherland and click Orders to invoke the Orders form module.
   Execute an unrestricted query (select Query > Execute).
   Press [Next Record] until you see Harrison Sutherland.
   Click Orders.

9. Click Image Off and notice that the image item is no longer displayed. Click Image On and notice that the image item is displayed.
   No formal solution.
10. Query only those orders that were submitted online.
    Select Query > Enter.
    Check the Online checkbox.
    Select Query > Execute.
11. Move to the fourth record (Product ID 2322) in the Item block of Order 2355 and click Stock.
    The Inventory block is displayed in a separate window with stock information for that item.
    No formal solution.
12. Close the Stock Levels window. For the customer Harrison Sutherland, insert a new record in the ORDER block, as detailed below.
    Click the X in upper right of Stock Levels window. Move to the ORDER block and select Record > Insert, or click Insert Record on the toolbar.
    Notice that some items are already populated with default values. Enter the following:

Item	**Value**
Online	Unchecked
Status	New Credit Order (poplist)
Sales Rep ID	151 (Enter the ID, or click the List button and select David Bernstein)

13. Insert a new record in the ITEM block with the following values:
    Move to the ITEM block and enter the following:

Item	**Value**
Product ID	2289 (Enter the ID, or click the List button and select KB 101/ES.
Quantity	2

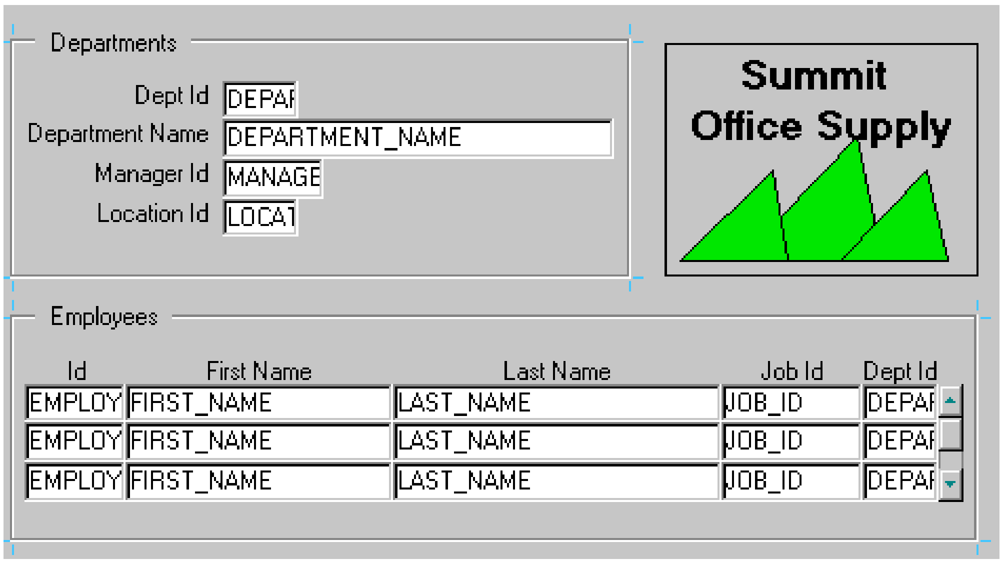
14. Save the new records.
    Select Action > Save or click Save.
15. Update the order that you have just placed and save the change.
    Note: You may receive a message indicating that there are no changes to save. This message is generated by the Customers form, because both forms are saved at the same time. Changes to the Orders form should be saved successfully, so you can acknowledge the message and then ignore it.
    Change the Order Date to last Monday and click Save.
16. Attempt to delete the order that you have just placed. What happens?
    Move to the Orders block and select Record > Remove. You are not able to delete the order because there are detail (item) records.

Practice 2 Solutions (continued)

17. Delete the line item for your order and save the change.
    Move to the Item block and select Record > Remove. Click Save.
18. Now attempt to delete your order and save the change.
    Move to the ORDER block and select Record > Remove. Click Save.
19. Exit the run-time session and close the browser window.
    Choose Action > Exit from the menu, or click Exit on the toolbar.
    Close the browser window.

Practice 3 Solutions

1. Invoke Forms Builder. If the Welcome page is displayed, select "Open an existing form". If the Welcome page is not displayed, select File > Open.
   No formal solution.
2. Open the `Orders.fmb` form module from the Open Dialog window.
   No formal solution.
3. Set your preferences so that Welcome dialogs display when you first open Forms Builder and when you use any of the wizards.
   Select Edit > Preferences from the default menu system. Click the Wizards tab in the Preferences dialog box.
   Check all boxes. Click OK.
4. Close the Orders form.
   With the Orders form selected, choose File > Close from the menu.
5. Open the `Summit.fmb` form module.
   No formal solution.
6. Expand the Data Blocks node.
   No formal solution.
7. Expand the Database Objects node. If you cannot expand the node, connect to the database and try again. What do you see below this node?
   No formal solution.
8. Collapse the Data Blocks node.
   No formal solution.
9. Change the layout of the `Summit.fmb` form module to match the following screenshot. At the end, save your changes.



a. Invoke the Layout Editor.
   With the Summit form selected, select Tools > Layout Editor from the menu.
b. Move the three summit shapes to the top-right corner of the layout. Align the objects along the bottom edge.
   Shift-click each of the three shapes to select them together. Move them to the top-right corner of the layout. With all three shapes still selected, select Layout > Align Components, and select the option to Align Bottom. Click OK.
c. Select the summit shape in the middle and place it behind the other two shapes.
   Click outside the shapes to deselect them, then select the middle summit shape and select Layout > Send to Back.
d. Draw a box with no fill around the summit shapes.
   Select the Rectangle tool from the Tool Palette and draw a rectangle around the three summit shapes. With the rectangle still selected, click the Fill Color tool and select No Fill.
e. Add the text Summit Office Supply in the box. If necessary, enlarge the box.
   Select the text tool from the Tool Palette and enter the text within the rectangle. Choose a suitable font size and style.
   Click outside the text, then from the menu select Layout > Justify > Center.
f. Move the Manager_Id and Location_Id items to match the screenshot.
   Select and move the Manager_Id and Location_Id items below the Department_Name item. Shift-click the Dept_Id, Department_Name, Manager_Id, and Location_Id items to select them together. Click Align Left to align these items. To distribute them evenly, select Layout > Align Components, then select the Distribute option under the Vertically column and click OK.
g. Move the First_Name item up to align it at the same level as the Last_Name item.
   Select the First_Name and Last_Name items together, and click Align Top.
h. Resize the scroll bar to make it the same height as the three records in the Employees block.
   Select the scroll bar and resize it with the mouse.
i. Save the form module.
   In the Object Navigator, select File > Save (or click Save).

Practice 3 Solutions (continued)

10. Set the run-time preferences for Forms Builder to use OC4J to test your applications. Set the Application Server URL by pressing Reset to Default, which will enter the following settings:

URL Component	Value
Machine name	`127.0.0.1` (or your local machine name)
Port	`8889` (for OC4J)(or the OC4J port on your local machine)
Pointer to Forms Servlet	`forms90/f90servlet`

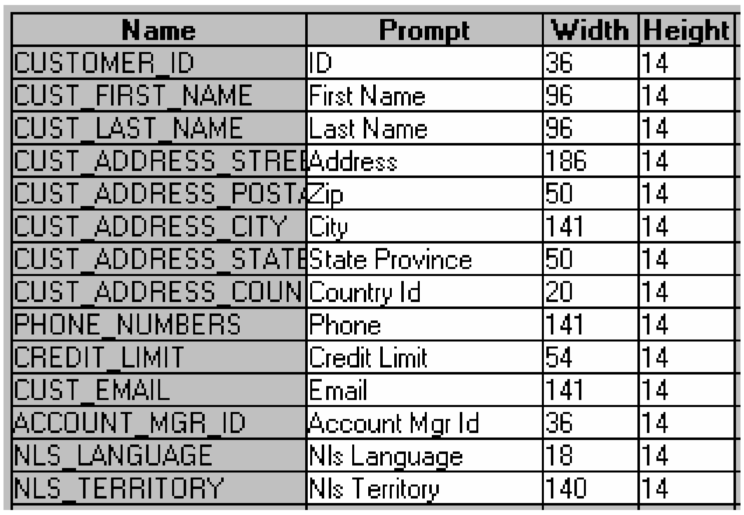
    From the Forms Builder menu, choose Edit > Preferences.
    Select the Runtime tab.
    Click Reset to Default.
11. In Forms Builder, open and run the Customers form located in your local directory (you must have OC4J running first). **Note:** Run-time fonts may look different than the fonts used in Forms Builder because Forms Builder uses operating system specific fonts, but at run time only Java fonts are used. In addition, the appearance is different from the Layout Editor because the Oracle (rather than the Generic) look and feel is used by default.
    Click Open, or choose File > Open from the menu.
    Open `customers.fmb`.
    Click Run Form, or choose Program > Run Form from the menu.
    Enter the connect information in the Logon dialog and click Connect.
12. Click the Account Information tab. You should be able to see the image of a flashlight on the List button. Exit the run-time session and close the browser window.
    No formal solution.
13. In Forms Builder, open the Layout Editor for the CV_Customer canvas by expanding the Canvases node in the Object Navigator and double-clicking the CV_Customer canvas icon. In the Layout Editor, click the Account Information tab. What do you observe about the List button?
    The List button displays without an iconic image.
14. From the Windows Start menu, choose Run, type `regedit`, and click OK. Expand the registry nodes `HKEY_LOCAL_MACHINE` > `SOFTWARE` > `ORACLE`. Click into the `ORACLE` node, or into one of the `HOME` nodes beneath it; your instructor will tell you which node to open. Ensure that you have opened the correct node by verifying that the key `FORMS90` exists in that node.
    No formal solution.
15. Set the path for Forms Builder to locate icons:
    a) Double-click the `UI_ICON` key to open it for editing.
    b) For the value data, append the path to the `.gif` file that you will use for the button icon, which is the `\icons` subdirectory of your lab directory. Separate this path from the remainder of the string with a semicolon; for example: `;e:\labs\lab\icons`, then click OK.
    No formal solution.

Practice 3 Solutions (continued)

16. In a similar fashion, set the value for Forms Builder to use for the icon extension, then close the registry editor.
    If the key `UI_ICON_EXTENSION` exists, ensure that it is set to `"gif"`. If it does not exist, from the registry menu, select Edit > New > String Value. Enter the name `UI_ICON_EXTENSION`. Double-click the string to open it for editing. For the value data, enter `gif`, then click OK.
17. Close and reopen Forms Builder. Open the Customers form and verify that the flashlight icon now displays in the Layout Editor.
    No formal solution.

Practice 4 Solutions

1. Create a new form module.
   Create a new single block by using the Data Block Wizard.
   Base it on the `CUSTOMERS` table and include all columns.
   Display the CUSTOMERS block on a new content canvas called `CV_CUSTOMER` and show just one record at a time. Set the frame title to Customers. Set column names and widths as shown in the following table:



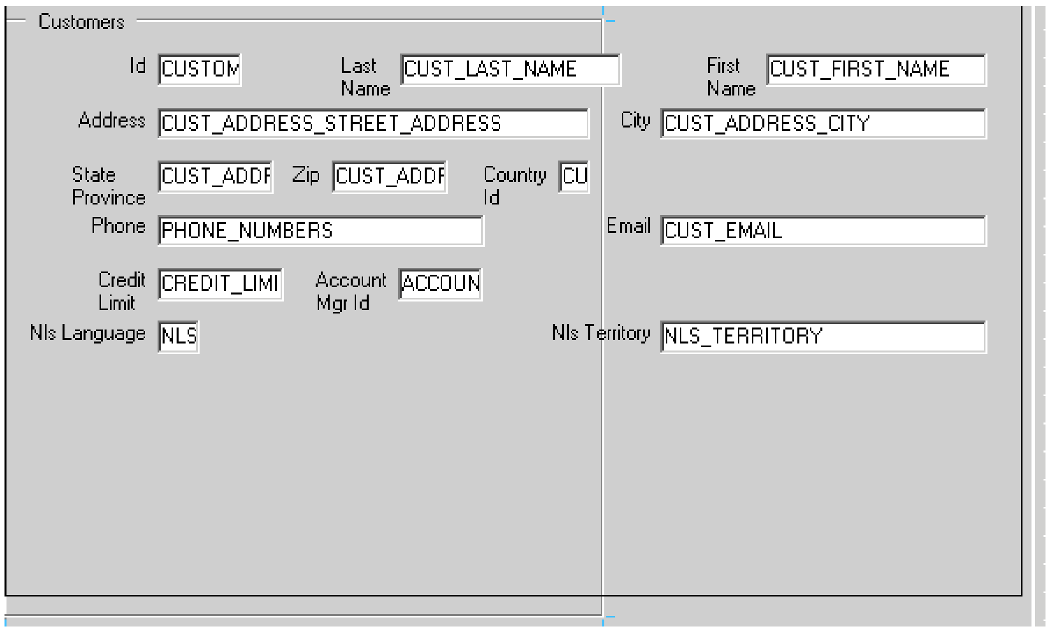
Name	Prompt	Width	Height
CUSTOMER_ID	ID	36	14
CUST_FIRST_NAME	First Name	96	14
CUST_LAST_NAME	Last Name	96	14
CUST_ADDRESS_STRE	Address	186	14
CUST_ADDRESS_POST	Zip	50	14
CUST_ADDRESS_CITY	City	141	14
CUST_ADDRESS_STATE	State Province	50	14
CUST_ADDRESS_COUN	Country Id	20	14
PHONE_NUMBERS	Phone	141	14
CREDIT_LIMIT	Credit Limit	54	14
CUST_EMAIL	Email	141	14
ACCOUNT_MGR_ID	Account Mgr Id	36	14
NLS_LANGUAGE	Nls Language	18	14
NLS_TERRITORY	Nls Territory	140	14

If you are not already in Forms Builder, run Forms Builder and create a new form module by selecting "Use the Data Block Wizard" from the Welcome Wizard.
If you are already in Forms Builder, then create a new form module by selecting File > New > Form or by highlighting the Forms node in the Object Navigator and clicking Create. To begin creating a block, choose Tools > Data Block Wizard from the menu.
Select the block type as Table or View.
Set the Table or View field to CUSTOMERS.
Click Refresh. Click >> to include all columns, then click Next.
Click Next, and select the “Create the data block, then call the Layout Wizard” option, and click Finish.
In the Layout Wizard, select [New Canvas] and make sure the Type field is set to Content. Click Next.
Include all items and click Next.
Set values for prompt, width, and height as shown, then click Next.
Set the Style to Form and click Next.
Set the Frame Title to `Customers` and click Finish.
In Object Navigator, rename the canvas as CV_CUSTOMER:

- **Select the canvas.**
- **Click the name.**
- **The cursor changes to an I-beam; edit the name and press [Enter].**

Practice 4 Solutions (continued)

2. Save the new module to a file called `CUSTG`*XX*, where *XX* is the group number that your instructor has assigned to you.
   No formal solution
3. Run your form module and execute a query.
   Navigate through the fields. Exit the run-time session and return to Forms Builder.
   No formal solution.
4. Change the form module name in the Object Navigator to CUSTOMERS.
   Select the form module. Click the name. The cursor changes to an I-beam. Edit the name, and then press [Enter].
5. In the Layout Editor, reposition the items and edit item prompts so that the canvas resembles the following:
   Hint: First resize the canvas. Do not attempt to resize the frame, or the items will revert to their original positions.
   Reposition the items by dragging and dropping them.
   Use the Align Left and Align Right buttons to line the items up with one another. Edit the following item prompts to include a carriage return as pictured: Last Name, First Name, State Province, Country Id, Credit Limit, and Account Mgr Id. (Click twice in the item prompt to edit it – the cursor changes to an I-beam.)



Customers

Id	CUSTOM	Last Name	CUST_LAST_NAME	First Name	CUST_FIRST_NAME
Address	CUST_ADDRESS_STREET_ADDRESS			City	CUST_ADDRESS_CITY
State Province	CUST_ADDR	Zip	CUST_ADDR	Country Id	CU
Phone	PHONE_NUMBERS			Email	CUST_EMAIL
Credit Limit	CREDIT_LIMI	Account Mgr Id	ACCOUN		
Nls Language	NLS			Nls Territory	NLS_TERRITORY

6. Save and compile the form module.
   Click Run Form to run the form. Execute a query.
   No formal solution.
7. Exit the run-time session and close the browser window.
   No formal solution.

Practice 5 Solutions

1. Create a new form module.
   Create a new block by using the Data Block Wizard.
   Base it on the `ORDERS` table and include all columns except `ORDER_TOTAL` and `PROMOTION_ID`.
   Display the `ORDERS` block on a new content canvas called `CV_ORDER` and show just one record at a time. Use a form style layout. Set the frame title to Orders.
   Create a new form module by selecting File > New > Form or by clicking Create.
   Use Tools > Data Block Wizard to create a block.
   Select the block type as Table or View.
   Set the Table or View field to `ORDERS`.
   Click Refresh and include all columns except Order_Total and Promotion_Id.
   Click Next twice, select "Create the data block, then call the Layout Wizard" option, and click Finish.
   In the Layout Wizard, select a new canvas and make sure the Type field is set to Content.
   Include all items.
   Set Style to Form.
   Set Frame Title to `Orders`, and click Finish.
   In the Object Navigator, rename the canvas as `CV_ORDER`.
2. Create a new block by using the Data Block Wizard.
   Base the block on the `ORDER_ITEMS` table and include all columns.
   Create a relationship and select the master block as `ORDERS`.
   Display all items except `ORDER_ID` on the `CV_ORDER` canvas.
   Display six records in this detail block on the same canvas as the master block.
   Use a tabular style layout and include a scroll bar.
   Change the order of the blocks in the Object Navigator, moving the `ORDER_ITEMS` block after the `ORDERS` block. Set the frame title to Items.
   In the same module, create a new block by using Tools > Data Block Wizard. (Ensure that you do not have a frame selected in the Object Navigaor when you invoke the Data Block Wizard, or it will be in reentrant mode to modify that frame. If this happens, click Cancel, select a different object in the form, and invoke the Data Block Wizard again.)
   Select block type as Table or View.
   Set the Base Table to `ORDER_ITEMS`.
   Include all columns.
   Click Create Relationship.
   Select `ORDERS` block as the master block and click OK.
   Click Finish.
   Use the Layout Wizard to create a layout.
   Select Canvas as `CV_ORDER`.

Include all items except `ORDER_ID`.
Do not change any prompts.
Set the Style to Tabular.
Set the Frame Title to `Items`.
Set the Records Displayed to 6.
Select the Display Scrollbar check box.
In the Object Navigator, if `ORDER_ITEMS` is displayed first, drag and drop the `ORDER_ITEMS` block to a position below the `ORDERS` block.

3. Save the new module to a file called `ORDG`*XX*, where *XX* is the group number that your instructor has assigned to you.
   No formal solution.
4. Create a new block based on `INVENTORIES` (do not create any relationships with other blocks at this time) to display on a different canvas.
   Base it on the `INVENTORIES` table.
   Display four records in this block and ensure that they are displayed on a new content canvas called `CV_INVENTORY`.
   Use a tabular style layout, and include a scroll bar.
   In the Object Navigator, move the `INVENTORIES` block after the `ORDER_ITEMS` block. Set the frame title to Stock.
   Do not create any relationships between blocks at this stage.
   In the same module, create a new block by using Tools > Data Block Wizard. (Ensure that you do not have a frame selected in the Object Navigaor when you invoke the Data Block Wizard, or it will be in reentrant mode to modify that frame. If this happens, click Cancel, select a different object in the form, and invoke the Data Block Wizard again.)
   Select block type as Table or View.
   Set the Base Table to `INVENTORIES`.
   Include all columns.
   Use the Layout Wizard to create a layout.
   Select a New Canvas.
   Include all items.
   Do not change any prompts.
   Set the Style to Tabular.
   Set the Frame Title to `Stock`.
   Set the Records Displayed to 4.
   Select the Display Scrollbar check box.
   In the Object Navigator, rename the canvas to `CV_INVENTORY`.
   In the Object Navigator, if `INVENTORIES` is not displayed last in the block list, move the `INVENTORIES` block after the `ORDER_ITEMS` block.

Practice 5 Solutions (continued)

5. Explicitly create a relation called Order_Items_Inventories between the `ORDER_ITEMS` and `INVENTORIES` blocks.
   Ensure that line item records can be deleted independently of any related inventory.
   Set the coordination so that the Inventories block is not queried until you explicitly execute a query.
   Create the relation: Select the Relations node in the `ORDER_ITEMS` block in the Object Navigator and click Create. The New Relation dialog box appears.
   Select `INVENTORIES` as the detail block. Select the Isolated option. Check Deferred and uncheck Auto Query. Enter the join condition `order_items.product_id = inventories.product_id` and click OK.
6. On the `ORDER_ITEMS` block, change the prompt for the Line Item ID item to Item# by using the reentrant Layout Wizard. First select the relevant frame in the Layout Editor, and then use the Layout Wizard.
   Select the frame for the `ORDER_ITEMS` block under the `CV_ORDER` canvas in the Object Navigator or in the Layout Editor, and select Tools > Layout Wizard from the menu.
   Select the Items tab page.
   Change the prompt for the Line Item ID item to Item#, and click Finish.
7. In the `INVENTORIES` data block, change the prompt for Quantity on Hand to In Stock by using the Layout Wizard.
   In the Object Navigator or in the Layout Editor, select the frame that is associated with the `INVENTORIES` data block.
   Select Tools > Layout Wizard from the menu.
   Select the Items tab page and change the prompt for Quantity on Hand to In Stock, and click Finish.
8. Save and compile your form module.
   Click Run Form to run your form module.
   Execute a query.
   Navigate through the blocks so that you see the `INVENTORIES` block.
   Exit the run-time session, close the browser, and return to Forms Builder.
   To navigate through the blocks, choose Block > Next from the menu or click Next Block on the Toolbar.
9. Change the form module name in the Object Navigator to `ORDERS` and save.
   No formal solution.

Practice 6 Solutions

CUSTG*XX* Form

1. Create a control block in the CUSTG*XX* form.
   Create a new block manually, and rename this block `CONTROL`.
   Set the Database Data Block, Query Allowed, Insert Allowed, Update Allowed, and Delete Allowed Database properties to No. Set the Query Data Source Type property to None. Set the Single Record property to Yes. Leave other properties as default.
   Move the `CONTROL` block after the `CUSTOMERS` block.
   Select the Data Blocks node in the Object Navigator.
   Click the Create icon in the Object Navigator, or select Edit > Create option from menu to create a new data block.
   Select the "Build a new data block manually" option.
   Rename this new data block as `CONTROL`.
   Right-click this block, and open the Property Palette. Find the Database category in the Property Palette.
   Set the Database Data Block, Query Allowed, Insert Allowed, Update Allowed, and Delete Allowed properties to No.
   Set the Query Data Source Type property to None.
   Find the Records category.
   Set the Single Record property to Yes.
   Leave other properties as default.
   In the Object Navigator, if the `CONTROL` block is not displayed last, move the `CONTROL` block after the `CUSTOMERS` block.
2. Ensure that the records retrieved in the `CUSTOMERS` block are sorted by the customer's ID.
   In the Property Palette for the `CUSTOMERS` block, set the `ORDER BY` Clause property to `customer_id`.
3. Set the frame properties for the `CUSTOMERS` block as follows:
   Remove the frame title, and set the Update Layout property to Manually. Once you have done this, you may resize the frame if desired without having the items revert to their original positions.
   In the Layout Editor for the CV_Customer canvas, select the frame that covers the `CUSTOMERS` block and open the Property Palette. Remove the Frame Title property value and set the Update Layout property to Manually. You may resize the frame if desired.
4. Save and run the CUSTG*XX* form.
   Test the effects of the properties that you have set.
   Ensure that records are sorted by Customer_Id and that the frame title is no longer displayed.

   Note: The Compilation Errors window displays a warning that advises you that the `CONTROL` block has no items. This is expected (until you add some items to the `CONTROL` block in a later lesson).

ORDG*XX* Form

5. Create a `CONTROL` block in the ORDG*XX* form.
   Create a new block manually, and rename this block `CONTROL`.
   Set the Database Data Block, Query Allowed, Insert Allowed, Update Allowed, and Delete Allowed database properties to No. Set the Query Data Source Type property to None. Set the Single Record property to Yes. Leave other properties as default.
   Position the `CONTROL` block after the `INVENTORIES` block in the Object Navigator.
   Select the Data Blocks node in the Object Navigator.
   Click the Create icon in the Object Navigator, or select the Edit > Create option from the menu to create a new data block.
   Select the "Build a new data block manually" option.
   Rename this new data block `CONTROL`.
   Right-click this block, and open the Property Palette.
   Find the Database category in the Property Palette.
   Set the Database Data Block, Query Allowed, Insert Allowed, Update Allowed, and Delete Allowed properties to No.
   Set the Query Data Source Type property to None.
   Find the Records category.
   Set the Single Record property to Yes.
   Leave other properties as default.
   In the Object Navigator, move the `CONTROL` block after the `INVENTORIES` block.
6. Ensure that the records retrieved in the `ORDERS` block are sorted by the `ORDER_ID`.
   For the `ORDERS` data block, set the `ORDER BY` Clause property to `ORDER_ID`.
7. Ensure that the current record is displayed differently from the others in both the `ORDER_ITEMS` and `INVENTORIES` blocks.
   Create a Visual Attribute called Current_Record.
   Using the Color Picker, set the foreground color to white and the background color to gray. Using the Pattern Picker, set the pattern to a light and unobtrusive pattern. Using the Font Picker, set the font to MS Serif italic 10 point.
   Use the multiple selection feature on both data blocks to set the relevant block property to use this Visual Attribute.
   In the Object Navigator, select the Visual Attributes node, and create a new Visual Attribute.
   In the Property Palette, set the Name property to `CURRENT_RECORD`.
   Select the Foreground Color property and click the More button, which is labeled "…".

Practice 6 Solutions (continued)

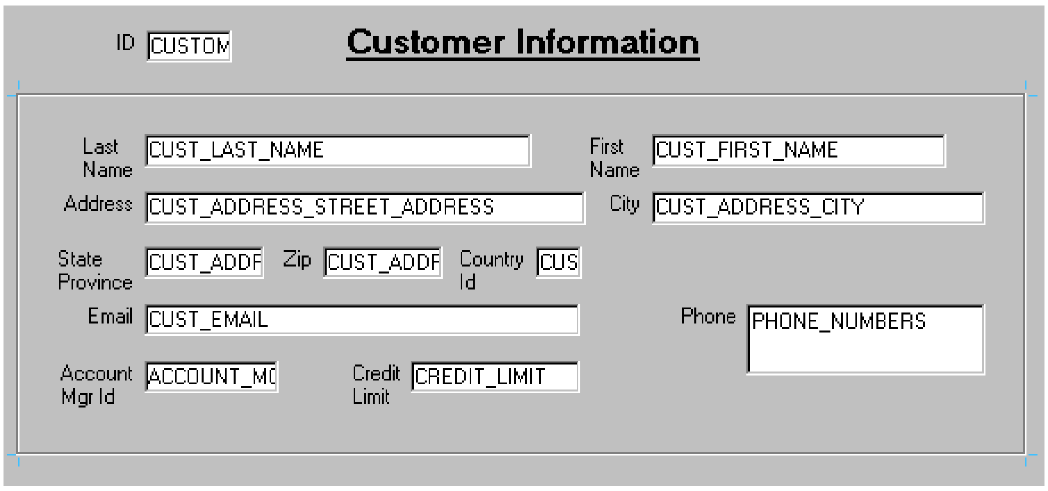
The Foreground Color color picker dialog box is displayed. Set Foreground Color to White. Repeat the process to set the Background Color to gray.
In the Property Palette, select the Fill Pattern property. Click More (…). Select the third pattern from the left in the top row, which sets the pattern to gray3.3 (you could type this value if you did not want to use the Pattern Picker).
In the Property Palette, select the Font category heading. (Do not select any of the properties under the Font category heading.) Click More… and the Font dialog box appears. Select MS Serif, Italic, 10 point, and click OK.
In the Object Navigator, to use the multiple selection feature, select both of the `ORDER_ITEMS` and the `INVENTORIES` blocks by pressing the [Shift] key and the left mouse button, and then open the Property Palette.
Set the Current Record Visual Attribute Group property to `CURRENT_RECORD`.

8. For the `ORDER_ITEMS` block, change the number of records displayed to 4 and resize the scroll bar accordingly.
   In the Object Navigator, select the `ORDER_ITEMS` block and open the Property Palette. Set the Number of Records Displayed property
   to 4. In the Layout Editor for the CV_ORDER canvas, resize the scroll bar to match the number of records displayed.
9. Ensure that the records retrieved in the `ORDER_ITEMS` block are sorted by the `LINE_ITEM_ID`.
   For the `ORDER_ITEMS` data block, set the ORDER BY Clause property to `LINE_ITEM_ID`.
10. Set the `ORDER_ITEMS` block to automatically navigate to the next record when the user presses [Next Item] while the cursor is in the last item of a record.
    For the `ORDER_ITEMS` block, set the Navigation Style to Change Record.
11. Set the frame properties for all blocks as follows:
    Remove the frame title and set the Update Layout property to Manually.
    In the Object Navigator, expand all nodes under the Canvases node. Multi-select all frames under the Graphics nodes and open the Property Palette. Remove the Frame Title property value and set the Update Layout property to Manually.
12. Save and compile the ORDG*XX* form.
    Click Run Form to run your form.
    Test the effects of the properties that you have set.
    Note: The Compilation Errors window displays a warning that advises you that the `CONTROL` block has no items. This is expected (until you add some items to the `CONTROL` block in a later lesson).
    No formal solution.

Practice 7 Solutions

CUSTG*XX* Form

1. Remove the NLS_Language and NLS_Territory items.
   In the Layout Editor, select and delete the two items.
2. Make sure that the Phone_Numbers item accepts multiline text to display. The database column is long enough to accept two phone numbers if the second one is entered without “+1” in front of the number.
   For the Phone Numbers item, set Multi-line to Yes. Set Height to 30 and Width to 100.
3. Automatically display a unique, new customer number for each new record and ensure that it cannot be changed.
   Use the `CUSTOMERS_SEQ` sequence.
   In the Property Palette for Customer_Id, set Initial Value to **`:sequence.customers_seq.nextval`.**
   Set the properties Insert Allowed and Update Allowed to No.
4. In the CUSTG*XX* form, resize and reposition the items. Add the boilerplate text Customer Information. Reorder the items in the Object Navigator. Use the screenshot as a guide.



5. Save and compile your form.
   Test the changes by clicking Run Form to run the form.
   Note: The entire form may not be visible at this time. This will be addressed in a later lesson.
   No formal solution.

Practice 7 Solutions (continued)

ORDG*XX* Form

6. In the `ORDERS` block, create a new text item called Customer_Name.
   Ensure that Customer_Name is not associated with the `ORDERS` table.
   Do not allow insert, update, or query operations on this item, and make sure that navigation is possible only by means of the mouse. Set the Prompt text to Customer Name. Display this item on `CV_ORDER` canvas.
   Create a text item in the ORDERS block and name it `Customer_Name`.
   Item Type should be set to Text Item.
   Set the Database Item, Insert Allowed, Update Allowed, Query Allowed, and Keyboard Navigable properties to No.
   Set the Prompt to `Customer Name`.
   Set the Prompt Attachment Offset to `5`.
   Set the Canvas property to `CV_ORDER`.
7. In the `ORDERS` block, create a new text item called Sales_Rep_Name.
   Ensure that Sales_Rep_Name is not associated with the `ORDERS` table.
   Do not allow insert, update, or query operations on this item and make sure that navigation is possible only by means of the mouse. Set the Prompt text to Sales Rep Name. Display this item on the `CV_ORDER` canvas.
   Create a text item in the ORDERS block and name it `Sales_Rep_Name`.
   Item Type should be set to Text Item.
   Set the Database Item, Insert Allowed, Update Allowed, Query Allowed, and Keyboard Navigable properties to No.
   Set the Prompt to `Sales Rep Name`.
   Set the Prompt Attachment Offset to `5`.
   Set the Canvas property to `CV_ORDER`.
8. Set the relevant property for Order_Date, so that it displays the current date whenever a new record is entered.
   For Order_Date, set Initial Value to: `$$date$$`
9. In the `ORDER_ITEMS` block, create a new text item called Item_Total.
   Ensure that Item_Total is not associated with the `ORDER_ITEMS` table.
   Do not allow insert, update, or query operations on this item and make sure that navigation is possible only by means of the mouse.
   Allow numeric data only and display it by using a format of 999G990D99.
   Set the Prompt text to Item Total. Display this item on the `CV_ORDER` canvas.
   Create a text item in the ORDER_ITEMS block and name it `Item_Total`.
   Set the Item Type to Text Item.
   Set the Database Item, Insert Allowed, Update Allowed, Query Allowed, and Keyboard Navigable properties to No.
   Set the Data Type to Number.
   Set the Format Mask to: `999G990D99`

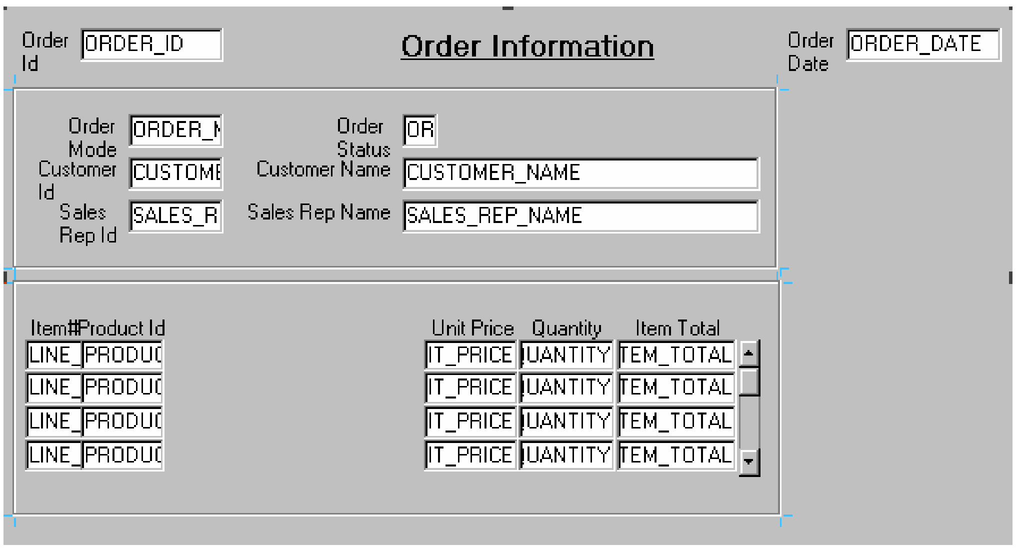
Practice 7 Solutions (continued)

Set the Prompt to `Item Total`.
Set the Prompt Attachment Edge to Top.
Set the Prompt Alignment to Center.
Set the Height to 14.
Set the Canvas property to `CV_ORDER`.

10. Justify the values of Unit_Price, Quantity, and Item_Total to the right.
    For each of the items, set Justification to Right.
11. Alter the Unit_Price item, so that navigation is possible only by means of the mouse, and updates are not allowed. Set its format mask to be the same as that used for Item_Total.
    For Unit_Price, set Keyboard Navigable and Update Allowed to No.
    Set the Format Mask to: `999G990D99`
12. In the ORDG*XX* form, resize and reposition the items according to the screenshot and the following table.
    Resize items by setting the width in the corresponding property palette.
    Drag and drop items to reposition in navigation order.

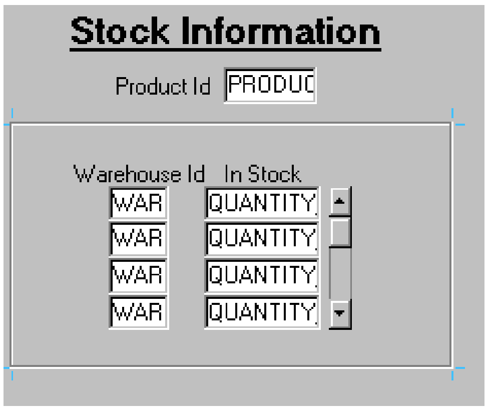
ORDERS Block Items	**Suggested Width**
Order_Id	60
Order_Date	65
Order_Mode	40
Customer_ID	40
Customer_Name	150
Order_Status	15
Sales_Rep_ID	40
Sales_Rep_Name	150

ORDER_ITEMS Block Items	**Suggested Width**
Line_Item_Id	25
Product_ID	35
Unit_Price	40
Quantity	40
Item_Total	50



Practice 7 Solutions (continued)

13. In the `INVENTORIES` block, alter the number of instances of the Product_ID, so that it is displayed just once. Make its prompt display to the left of the item.
    In the property palette for Product_ID, set Number of Items Displayed to `1`.
    Set the Prompt Attachment Edge to Start.
    Set the Prompt Attachment Offset to `5`.
14. Arrange the items and boilerplate on `CV_INVENTORY`, so that it resembles the screenshot.
    Hint: Set the Update Layout property for the frame to Manually.
    No formal solution.



15. Save, compile, and run the form to test the changes.
    No formal solution.

Instructor Note

Customer_Name and Sales_Rep_Name are created as text items in this course. They could also be created as display items. Display items use less memory than text items. Students may find that text items created in this practice are different in appearance from those created with the Data Block Wizard. This is due to bug 2020548. If you create objects on a canvas and then click on the canvas, the next object you create on the canvas has a different background color.

Practice 8 Solutions

1. In the ORDG*XX* form, create an LOV to display product numbers and descriptions to be used with the Product_Id item in the `ORDER_ITEMS` block.
   Use the `PRODUCTS` table and select the Product_Id and Product_Name columns. Assign a title of Products to the LOV. Sort the list by the product name. Assign a column width of 25 for Product_Id, and assign the LOV a width of 200 and a height of 300. Position the LOV 30 pixels below and to the right of the upper-left corner. For the Product_Id column, set the return item to `ORDER_ITEMS.PRODUCT_ID`. Attach the LOV to the Product_Id item in the `ORDER_ITEMS` block. Change the name of the LOV to PRODUCTS_LOV and the name of the record group to PRODUCTS_RG.
   Create a new LOV: In the Object Navigator, select the LOVs node and click Create. Click OK to use the LOV Wizard.
   Select the "New Record Group based on a query" option, and click Next.
   In the SQL Query Statement, enter or import the following SQL query from `pr8_1.txt`:

   ```
   select PRODUCT_ID, PRODUCT_NAME
   from PRODUCTS
   order by PRODUCT_NAME
   ```

   and click Next.

   Click >> and then Next to select both of the record group values. With the Product_ID column selected, click Look up return item. Select ORDER_ITEMS.PRODUCT_ID and click OK.

   Set the display width for PRODUCT_ID to 25 and click Next.

   Enter the title `Products`. Assign the LOV a width of `200` and a height of `300`. Select the "No, I want to position it manually" option, and set both Left and Top to `30`. Click Next.

   Click Next to accept the default advanced properties.

   Click > and then Finish to create the LOV and attach it to the Product_Id item.

   In the Object Navigator, change the name of the new LOV to `PRODUCTS_LOV` and change the name of the new record group to `PRODUCTS_RG`.

2. In the ORDG*XX* form, use the LOV Wizard to create an LOV to display sales representatives' numbers and their names. Use the `EMPLOYEES` table, Employee_Id, First_Name, and Last_Name columns. Concatenate the First_Name and the Last_Name columns and give the alias of Name. Select employees whose Job_ID is SA_REP.

Practice 8 Solutions (continued)

Assign a title of Sales Representatives to the LOV. Assign a column width of 20 for ID, and assign the LOV a width of 200 and a height of 300. Position the LOV 30 pixels below and to the right of the upper-left corner. For the ID column, set the return item to `ORDERS.SALES_REP_ID`; for the Name column, set the return item to `ORDERS.SALES_REP_NAME`. Attach the LOV to the Sales_Rep_Id item in the `ORDERS` block.
Change the name of the LOV to `SALES_REP_LOV` and the record group to `SALES_REP_RG`.
Create a new LOV. Click OK to use the LOV Wizard.
Select the "New Record Group based on a query" option, and click Next.
In the SQL Query Statement, enter or import the following SQL query from `pr8_2.txt`:

```
select employee_id, first_name || ' '||last_name name
from employees
where job_id = 'SA_REP'
order by last_name
```

and click Next.
Click >> and then Next to select both of the record group values.
With the Employee_ID column selected, click Look up return item. Select `ORDERS.SALES_REP_ID` and click OK.
With the Name column selected, the Look up return item. Select `ORDERS.SALES_REP_NAME` and click OK. Set the display width for Employee_ID to `20` and click Next.
Enter the title `Sales Representatives`. Assign the LOV a width of `200` and a height of `300`. Select the "No, I want to position it manually" option, and set both Left and Top to `30`. Click Next.
Click Next to accept the default advanced properties.
Select `ORDERS.SALES_REP_ID` and click > and then Finish to create the LOV and attach it to the Sales_Rep_Id item.
In the Object Navigator, change the name of the new LOV to `SALES_REP_LOV` and the new record group to `SALES_REP_RG`.

3. Save and compile your form.
   Click Run Form to run the form and test the changes.
   No formal solution.

Practice 8 Solutions (continued)

4. In the CUSTG*XX* form, use the LOV Wizard to create an LOV to display account managers' numbers and their names. Use the `EMPLOYEES` table, Employee_Id, First_Name, and Last_Name columns. Concatenate the First_Name and the Last_Name columns and give the alias of Name. Select employees whose Job_ID is SA_MAN.
   Assign a title of Account Managers to the LOV. Assign a column width of 20 for ID, and assign the LOV a width of 200 and a height of 300. Position the LOV 30 pixels below and to the right of the upper-left corner. For the ID column, set the return item to `CUSTOMERS.ACCOUNT_MGR_ID`. Attach the LOV to the Account_Mgr_Id item in the `CUSTOMERS` block.
   Change the name of the LOV to `ACCOUNT_MGR_LOV` and the record group to `ACCOUNT_MGR_RG`.
   Create a new LOV. Click OK to use the LOV Wizard.
   Select the "New Record Group based on a query" option, and click Next.
   In the SQL Query Statement, enter or import the following SQL query from `pr8_4.txt`:

```
select employee_id, first_name || ' '||last_name name
from employees
where job_id = 'SA_MAN'
order by last_name
```

   and click Next.
   Click >> and then Next to select both of the record group values.
   With the Employee_ID column selected, click Look up return item. Select `CUSTOMERS.ACCOUNT_MGR_ID` and click OK.
   Set the display width for ID to `20` and click Next.
   Enter the title `Account Managers`. Assign the LOV a width of `200` and a height of `300`. Select the "No, I want to position it manually" option, and set both Left and Top to `30`. Click Next.
   Click Next to accept the default advanced properties.
   Click > and then Finish to create the LOV and attach it to the Account_Mgr_Id item.
   In the Object Navigator, change the name of the new LOV to `ACCOUNT_MGR_LOV` and the new record group to `ACCOUNT_MGR_RG`.

Instructor Note

If students try to include an object type column in the query, they will receive an error. See Bug 2338467.

Practice 8 Solutions (continued)

5. In the CUSTG*XX* form, create an editor and attach it to the Phone_Numbers item. Set the title to Phone Numbers, the bottom title to Max 30 Characters, the background color to gray, and the foreground color to yellow.
   Create a new editor. Change the name to Phone_Number_Editor.
   Set the X Position and Y Position properties to `175`.
   Set the Width property to `100` and the Height property to `100`.
   Set the title to `Phone Numbers`, the bottom title to `Max 30 Characters`, Background Color property to `gray`, and the Foreground Color property to `yellow`.
   In the Property Palette of the Phone_Numbers item, set the Editor property to `Phone_Number_Editor`.
6. Save, compile, and run the form to test the changes. Resize the window if necessary.
   No formal solution.

Practice 9 Solutions

1. In the ORDG*XX* form, convert the Order_Status item into a pop-up list item.
   Add list elements shown in the table below.
   Display any other values as Unknown.
   Ensure that new records display the initial value New CASH order.
   Resize the poplist item in the Layout Editor, so that the elements do not truncate at run time.

List Element	List Item Value
New CASH order	0
CASH order being processed	1
CASH Backorder	2
CASH order shipped	3
New CREDIT order	4
CREDIT order being processed	5
CREDIT Backorder	6
CREDIT order shipped	7
CREDIT order billed	8
CREDIT order past due	9
CREDIT order paid	10
Unknown	11

   For Order_Status, set Item Type to List Item.
   Set the Initial Value to `0`.
   Set the List Style to Poplist (this is the default).
   Set the Mapping of Other Values to `11`.
   Select the Elements in List property and click More to invoke the List Elements dialog box.
   Enter the elements shown in the table. Enter the corresponding database values in the List Item Value box.
   Click OK to accept and close the dialog.
   Open the Layout Editor and resize the item so that elements do not truncate.

2. In the ORDG*XX* form, convert the Order_Mode text item to a check box.
   Set the checked state to represent the base table value of `online` and the unchecked state to represent `direct`.
   Ensure that new records are automatically assigned the value `online`.
   Display any other values as unchecked.
   Remove the existing prompt and set label to: Online?
   In the Layout Editor, resize the check box so that its label is fully displayed to the right. Resize it a little longer than needed in Forms Builder so that the label does not truncate at run-time.

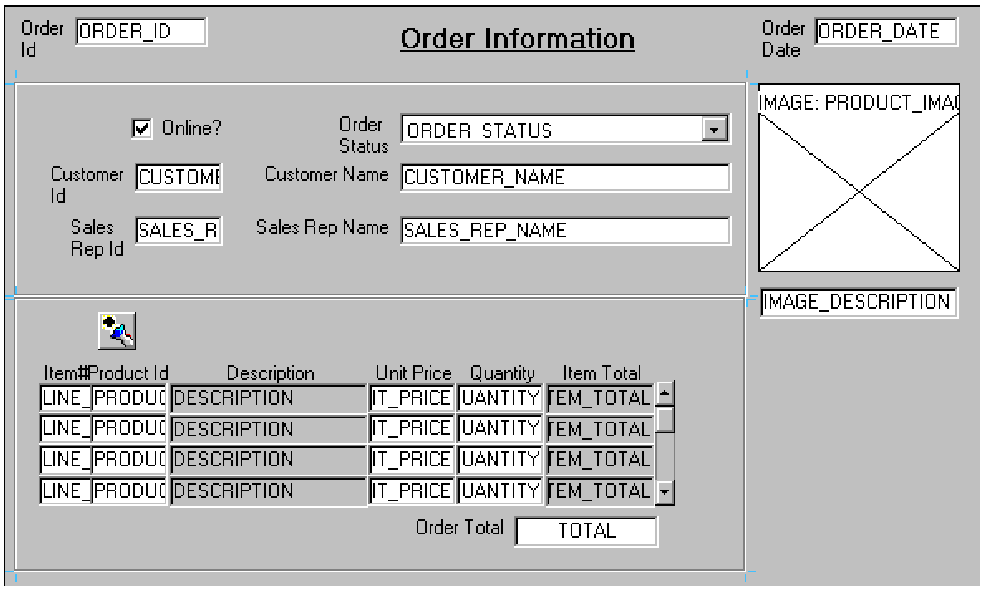
Practice 9 Solutions (continued)

For Order_Mode, set Item Type to Check Box.
Set the Value when Checked to: `online`
Set the Value when Unchecked to: `direct`
Set the Check Box Mapping of Other Values to Unchecked.
Set the Initial Value to: `online`
Delete the Prompt property; set the Label property to: `Online?`
Open the Layout Editor and resize the check box so that its label is fully displayed.

3. Save and compile the form.
   Click Run Form to run your form and test the changes.
   No formal solution.
4. In the CUSTG*XX* form, convert the Credit_Limit text item into a radio group.
   Add radio buttons for Low, Medium, and High to represent database values of 500, 2000, and 5000.
   Define access keys of L for Low, M for Medium, and H for High.
   Add text Credit Limit to describe the radio group's purpose.
   Set Label to Low for the Low radio button, Medium for the Medium radio button, and High for the High radio button.
   Ensure that new records display the default of Low, and that existing records with other values display as Medium.
   For Credit_Limit, set Item Type to Radio Group.
   Set the Initial Value to `500` and the Mapping of Other Values to `2000`.
   In the Object Navigator, expand the Credit_Limit node.
   The Radio Buttons node is displayed.
   Create three buttons; name them `Low_Button`, `Medium_Button`, and `High_Button`.
   For the first button, set Access Key to `L`, Label to `Low`, and Radio Button Value to `500`. For the second button, set Access Key to `M`, Label to `Medium`, and Radio Button Value to `2000`. For the third button, set Access Key to `H`, Label to `High`, and Radio Button Value to `5000`.
   Use the Layout Editor to position the radio buttons so that all can be seen.
   In the Layout Editor, create boilerplate text to identify radio buttons as `Credit Limit`.
5. Save, compile, and run the form to test the changes.
   No formal solution.

Practice 10 Solutions

1. In the `ORDER_ITEMS` block of the ORDG*XX* form, create a display item called Description. Set the Prompt property to Description and display the prompt centered above the item. Rearrange the items in the layout so that all are visible.
   Open the Layout Editor, ensure that the block is set to `ORDER_ITEMS`, and select the Display Item tool.
   Place the Display Item to the right of the Product_Id. Move the other items to the right to accommodate the display item.
   Set the Name property of the display item to `DESCRIPTION`.
   Set the Maximum Length to `50`.
   Set the Width to `80`. Set Height to `14`.
   Set the Database Item to No.
   Set the Prompt property to `Description`, the Prompt Attachment Edge property to Top, and the Prompt Alignment to Center.
2. Create an image item called Product_Image in the `CONTROL` block of the ORDG*XX* form. (Use the Control block because you do not want to pick up the Current Record Visual Attribute Group of the `ORDER_ITEMS` block, and this is a non base table item.) Position this and the other items you create in this practice so that your form looks like the screenshot.
   Note: The image will not display in the image item at this point; you will add code to populate the image item in Practice 20.



Display the Layout Editor; ensure that the block is set to `CONTROL`.
Select the Image Item tool.
Click and drag a box and place it so that it matches the screenshot.
Display the Property Palette for the image.
Change the name to `PRODUCT_IMAGE`.
Include these properties for the image item:
Keyboard Navigable: No, Sizing Style: Adjust, Database Item: No.

Practice 10 Solutions (continued)

3. Create another display item, Image_Description, in the `ORDER_ITEMS` block. This should synchronize with the Description item. Set the Maximum Length property to the same value as the Description item.
   Display the Layout Editor.
   Ensure that the block is set to `ORDER_ITEMS`.
   Select the Display Item tool.
   Place the Display Item to just below the Product_Image.
   Set the Name to `IMAGE_DESCRIPTION`.
   Set the Synchronize with Item to Description.
   Set the Maximum Length property to `0`.
   Set the Database Item to No.
   Set the Number of Items Displayed to `1`.
   Set the Width to `90`. Set the Height to `14`.
4. In the `CONTROL` block of the ORDG*XX* form, create an iconic button called Product_LOV_Button. Use the `list` file (do not include the `.ico` or `.gif` extension). Set the Keyboard Navigable property and the Mouse Navigate property to No.
   Display the Layout Editor and ensure that the block is set to `CONTROL`.
   Select the Button tool.
   Create a push button and place it close to the Product_Id.
   Set the Name to `Product_LOV_Button`.
   Set the Iconic property to Yes, and set the Icon Filename to: `list`
   Set the Keyboard Navigable property to No.
   Set the Mouse Navigate property to No.
   Set the Width and Height to `17`.
   Note: If you completed Practice 3 successfully, you should now be able to see the flashlight icon on the Product_LOV_Button button in the Layout Editor.
5. To display item total information, set the following properties for the Item_Total item in the `ORDER_ITEMS` block:
   Change the Item Type to Display Item.
   Set the Calculation Mode property to Formula.
   Set the Formula property to: `nvl(:ORDER_ITEMS.quantity,0) * nvl(:ORDER_ITEMS.unit_price,0)`
   In the Object Navigator, select the Item_Total item in the `ORDER_ITEMS` block, and open the Property Palette.
   Change the Item Type to Display Item.
   Set the Calculation Mode property to Formula.
   Set the Formula property to: `nvl(:ORDER_ITEMS.quantity,0) * nvl(:ORDER_ITEMS.unit_price,0)`

Practice 10 Solutions (continued)

6. To display the total of the item totals, create a new nondatabase display item in the `CONTROL` block.
   Set the Position, Size, and Prompt properties according to the screenshot.
   Set the Format Mask property to 9G999G990D99.
   Set the Justification property to Right.
   Set the Number of Items Displayed property to 1.
   Set the Keyboard Navigable property to No.
   Make `CONTROL`.total a summary item and display summaries of the item_total values in the `ORDER_ITEMS` block. Ensure that you have set the Query All Records property to Yes for the `ORDER_ITEMS` block.
   In the Object Navigator, create a new text item in the `CONTROL` block.
   Change the Item Type to Display Item.
   Set the Name property to `TOTAL`, and the Prompt property to `Order Total`, with a Prompt Attachment Offset of `5`.
   Set the Canvas property to `CV_ORDER`.
   Set the Number of Items Displayed property to `1`.
   Set the Data Type property to Number.
   Set the Format Mask property to: `9G999G990D99`
   Set the Justification property to Right.
   Set the Database Item property to No, the Calculation Mode property to Summary, and the Summary Function property to Sum.
   Set the Summarized Block property to `ORDER_ITEMS` and Summarized Item property to `Item_Total`.
   Set the Keyboard Navigable property to No.
   Resize and reposition the item according to the screenshot.
   Set the Query All Records property to Yes for the `ORDER_ITEMS` block; that is necessary for summary items to compute the sum of all the records.
7. Save the form and click Run Form to run it. Perform a query in the ORDG*XX* form to ensure that the new items do not cause an error. Did you remember to switch off the Database Item property for items that do not correspond to columns in the base table? Also, check that the calculated items function correctly.
   No formal solution.

Practice 10 Solutions (continued)

8. In the CUSTG*XX* form, create an iconic button similar to the one created in question 4, in the `CONTROL` block. Use the `list` file (do not include the `.ico` or `.gif` extension). Name the push button Account_Mgr_Lov_Button, and place it next to Account_Mgr_ID.
   Display the Layout Editor and ensure that the block is set to `CONTROL`.
   Select the Button tool.
   Create a push button and place it close to Account_Mgr_ID.
   Resize the push button to a Width of `17` and a Height of `17`.
   Set Name to `Account_Mgr_LOV_Button`.
   Set Keyboard Navigable to No.
   Set Mouse Navigate to No.
   Set Iconic to Yes.
   Set Icon File Name to: `list`
   Note: If you completed Practice 3 successfully, you should now be able to see the flashlight icon on the Account_Mgr_LOV_Button button in the Layout Editor.
9. In the CUSTG*XX* form, create a bean area in the `CONTROL` block and name it Colorpicker. This bean has no visible component on the form, so set it to display as a tiny dot in the upper left corner of the canvas. Ensure that users cannot navigate to the bean area item with either the keyboard or mouse. You will not set an implementation class; in Lesson 19 you learn how to programmatically instantiate the JavaBean at run time.
   Display the Layout Editor and ensure that the block is set to `CONTROL`.
   Select the Bean Area tool and click the upper left of the canvas.
   Set the NAME to `COLORPICKER`.
   Set X Position and Y Position to `0`.
   Set Width and Height to `1`.
   Set Keyboard Navigable and Mouse Navigate to No.
10. Save and compile the form. Click Run Form to run your form and test the changes.
    No formal solution.

Instructor Note

Students may ask questions about setting the Maximum Length property to 0 in question 3. They would not be wrong if they set it to the same value as Description. However, setting it to 0 makes it pick up the same Maximum Length as the item with which it is synchronized, so that if the Maximum Length of the Description item changes, the Maximum Length of the Image_Description item does not have to be changed manually.

Practice 11 Solutions

1. Modify the window in the CUSTG*XX* form. Change the name of the window to WIN_CUSTOMER, and change its title to Customer Information. Check that the size and position are suitable.
   Set the Name property of the existing window to `WIN_CUSTOMER`.
   Change the Title property to `Customer Information`.
   In the Layout Editor, look at the lowest and rightmost positions of objects on the canvas, and plan the height and width for the window.
   Change the height and width of the window in the Property Palette.
   The suggested size is Width `450`, Height `210`.
   The suggested X, Y positions are `10`, `10`.
2. Save, compile, and run the form to test the changes.
   No formal solution.
3. Modify the window in the ORDG*XX* form. Ensure that the window is called WIN_ORDER. Also change its title to Orders and Items. Check that the size and position are suitable.
   Set the Name property of the existing window to `WIN_ORDER`.
   Set the Title property to `Orders and Items`.
   In the Layout Editor, look at the lowest and rightmost positions of objects on the canvas, and plan the height and width for the window.
   Change the height and width of the window in the Property Palette.
   The suggested size is Width `440`, Height `265`.
   The suggested X, Y positions are `10`, `10`.
4. In the ORDG*XX* form, create a new window called WIN_INVENTORY suitable for displaying the CV_INVENTORY canvas. Use the rulers in the Layout Editor to help you plan the height and width of the window. Set the window title to Stock Levels. Place the new window in a suitable position relative to WIN_ORDER. Ensure that the window does not display when the user navigates to an item on a different window.
   Create a new window called `WIN_INVENTORY`.
   Set the Title property to `Stock Levels`.
   In the Layout Editor, look at the lowest and rightmost positions of objects on the canvas, and plan the height and width for the window. Look at the CV_ORDERS canvas to plan the X and Y positions for the window.
   Set the size and position in the Property Palette.
   The suggested size is Width `200`, Height `160`.
   The suggested position is X `160`, Y `100`.
   Set Hide on Exit to Yes.

Practice 11 Solutions (continued)

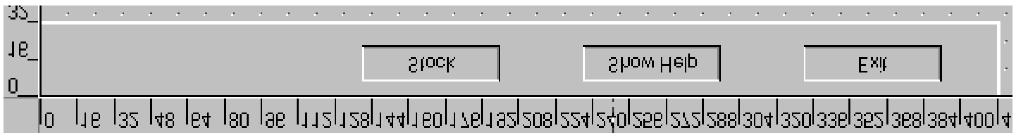
5. Associate the CV_INVENTORY canvas with the window WIN_INVENTORY. Compile the form. Click Run Form to run the form. Ensure that the INVENTORIES block is displayed in WIN_INVENTORY when you navigate to this block. Also make sure that the WIN_INVENTORY window is hidden when you navigate to one of the other blocks.
   Set the canvas Window property to WIN_INVENTORY.
   Click Run Form to compile and run the form.
6. Save the form.
   No formal solution.

Practice 12 Solutions

Toolbar Canvases

1. In the ORDG*XX* form, create a horizontal toolbar canvas called Toolbar in the WIN_ORDER window, and make it the standard toolbar for that window. Suggested height is 30.
   Create a new canvas. Set Name to `TOOLBAR` and Height to `30`. Set Canvas Type to Horizontal Toolbar and Window to WIN_ORDER.
   In the Property Palette of the WIN_ORDER window, set Horizontal Toolbar Canvas to Toolbar.
2. Save, compile, and run the form to test.
   Notice that the toolbar now uses part of the window space. Adjust the window size accordingly.
   Set the Height property of the WIN_ORDER window to add 30 to the height, to accommodate the horizontal toolbar.
3. Create three push buttons in the `CONTROL` block, as shown in the following table, and place them on the Toolbar canvas. Resize the Toolbar if needed.
   Suggested positions for the push buttons are shown in the illustration.

Push Button Name	**Details**
Stock_Button	Label: Stock Mouse Navigate: No Keyboard Navigable: No Canvas: Toolbar Background Color: white Height: 16 Width: 50
Show_Help_Button	Label: Show Help Mouse Navigate: No Keyboard Navigable: No Canvas: Toolbar Background Color: white Height: 16 Width: 50
Exit_Button	Label: Exit Mouse Navigate: No Keyboard Navigable: No Canvas: Toolbar Background Color: white Height: 16 Width: 50

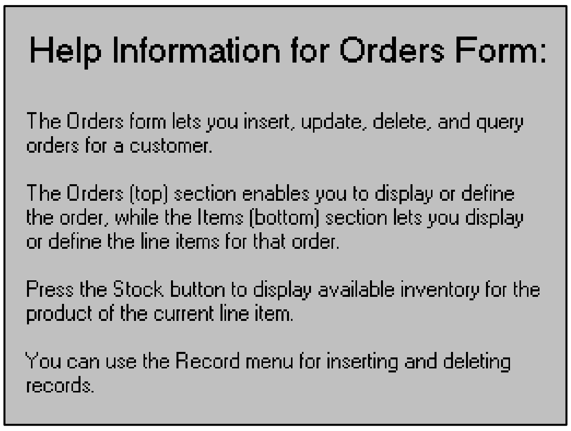


In the Layout Editor of the TOOLBAR canvas, with the CONTROL block selected, use the Button tool to create three push buttons. Select all three, invoke the Property Palette, and set the Mouse Navigate, Keyboard Navigable, Height, Width, and Background Color properties. Then select each button individually and set the Name and Label properties. In the Layout Editor, resize and reposition the buttons. Although gray in the Layout Editor, the buttons will be white at run time. Change the Width of the Toolbar if needed.

Practice 12 Solutions (continued)

Stacked Canvases

4. Create a stacked canvas named CV_HELP to display help in the WIN_ORDER window of the ORDG*XX* form. Suggested *visible* size is Viewport Width 250, Viewport Height 225 (points). Select a contrasting color for the canvas. Place some application help text on this canvas, similar to that shown:



Help Information for Orders Form:

The Orders form lets you insert, update, delete, and query orders for a customer.

The Orders (top) section enables you to display or define the order, while the Items (bottom) section lets you display or define the line items for that order.

Press the Stock button to display available inventory for the product of the current line item.

You can use the Record menu for inserting and deleting records.

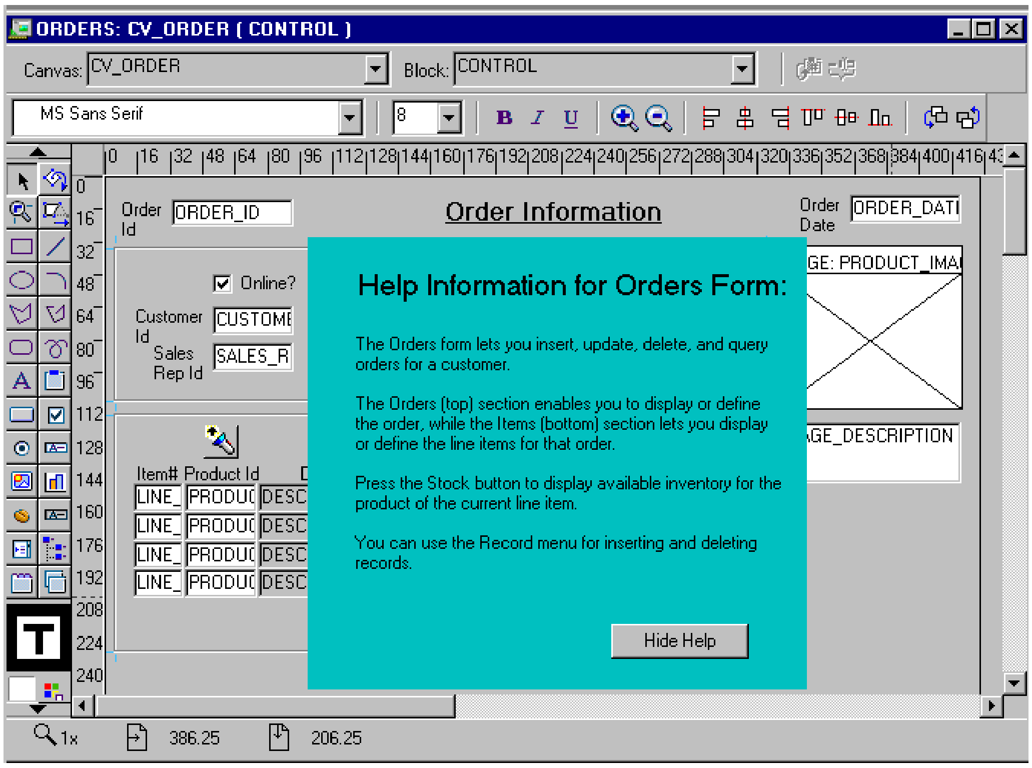
Create a new canvas.
If the Property Palette is not already displayed, click the new canvas object in the Object Navigator and select Tools > Property Palette.
Set the Canvas Type to Stacked. Name the canvas `CV_HELP`. Assign it to the WIN_ORDER window. Set the Viewport Width and Height and the Background Color properties.
Display the stacked canvas in the Layout Editor, and create some boilerplate text objects with help information about the form.

5. Position the view of the stacked canvas so that it appears in the center of WIN_ORDER. Ensure that it will not obscure the first enterable item.
Do this by planning the top-left position of the view in the Layout Editor, while showing CV_ORDER. Define the Viewport X and Viewport Y Positions in the Property Palette. You can move and resize the view in the Layout Editor to set the positions.
In the Layout Editor, display the CV_ORDER canvas, select View > Stacked Views from the menu, and select CV_HELP from the list. You can see both canvases in this way. Change Viewport X Position and Viewport Y Position properties in the Property Palette for CV_Help canvas. Suggested coordinates: `100`, `30`. You can also move and resize the view of the CV_Help canvas as you see it displayed on the CV_ORDER canvas.

Practice 12 Solutions (continued)

6. Organize CV_HELP so that it is the last canvas in sequence.
   Do this in the Object Navigator. (This ensures the correct stacking order at run time.)
   Drag the CV_Help canvas so that it is the last canvas displayed under the Canvases node in the Object Navigator.
7. Save and compile the form. Click Run Form to run your form and test the changes. Note that the stacked canvas is initially displayed until it obscures the current item in the form.
   No formal solution.
8. Switch off the Visible property of CV_HELP, and then create a push button in the CONTROL block to hide the Help information when it is no longer needed. You will add the code later. Display this push button on the CV_HELP canvas.
   Set the Visible Property to No for the CV_HELP canvas.
   Create a push button in the CONTROL block with the following properties:

Push Button Name	Details
Hide_Help_Button	Label: Hide Help Mouse Navigate: No Keyboard Navigable: No Canvas: CV_HELP Width, Height: 65, 16 X, Y Positions: 180, 200


ORDERS: CV_ORDER (CONTROL)

Canvas: CV_ORDER Block: CONTROL

MS Sans Serif 8

Order Id ORDER_ID

Order Information

Order Date ORDER_DATE

Online?

Customer Id CUSTOME

Sales Rep Id SALES_R

Item# Product Id

LINE_ PRODUC DESC

Help Information for Orders Form:

The Orders form lets you insert, update, delete, and query orders for a customer.

The Orders (top) section enables you to display or define the order, while the Items (bottom) section lets you display or define the line items for that order.

Press the Stock button to display available inventory for the product of the current line item.

You can use the Record menu for inserting and deleting records.

Hide Help

GE: PRODUCT_IMA

GE_DESCRIPTION

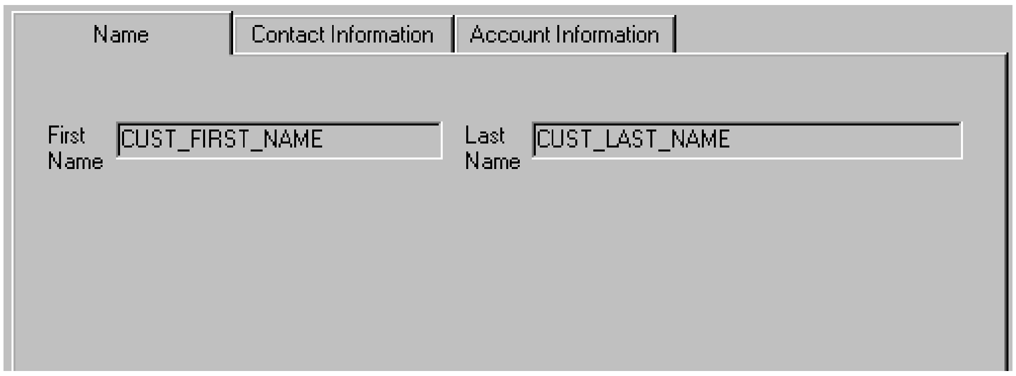
1x 386.25 206.25

Practice 12 Solutions (continued)

Tab Canvases

Modify the CUSTG*XX* form in order to use a tab canvas:

9. In the Layout Editor, delete the frame object that surrounds the `CUSTOMERS` block. Create a tab canvas (you may need to first enlarge the content canvas).
   In the Property Palette, set the Background Color property to gray, Corner Style property to Square, and Bevel property to None.
   In the Layout Editor, select the frame that surrounds the `CUSTOMERS` block and delete. Select the content canvas and resize it if necessary to make it large enough to accommodate the tab canvas you are about to create. Click Tab Canvas in the toolbar. Create a tab canvas by drawing a rectangle on the content canvas.
   In the Object Navigator, select the new tab canvas and open the Property Palette. Set the Background Color property to gray, Corner Style property to Square, and Bevel property to None.
10. Rename the TAB_CUSTOMER tab canvas. Create another tab page in addition to the two tab pages created by default. Name the tab pages as Name, Contact, and Account. Label them as Name, Contact Information, and Account Information.
    In the Property Palette, set the Name property to `TAB_CUSTOMER`.
    In the Object Navigator, expand this tab canvas and create one more tab page, for a total of three tab pages. Set the tab pages Name properties as `Name`, `Contact`, and `Account`. Set the Label properties as `Name`, `Contact Information`, and `Account Information`.
11. Design the tab pages according to the following screenshots. Set the item properties to make them display on the relevant tab pages.
    In the Object Navigator, select the CUST_LAST_NAME and CUST_FIRST_NAME items of the `CUSTOMERS` block and open the Property Palette for this multiple selection. Set the Canvas property to TAB_CUSTOMER. Set the Tab Page property to the NAME tab page.
    In the Layout Editor, arrange the items according to the screenshot.



Instructor Note

If students appear to “lose” items after setting their Canvas properties, have them set the items’ X and Y positions to 0 to move them to the upper left of the tab page, where they can be seen and then moved to the desired positions.

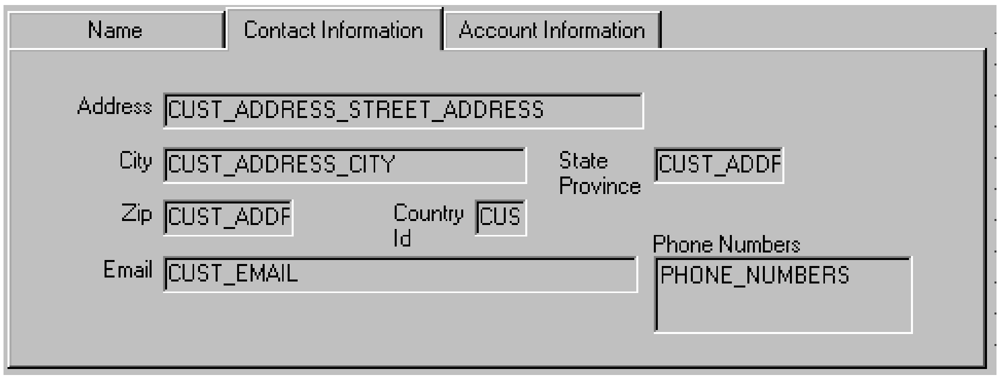
Practice 12 Solutions (continued)

In the Object Navigator, select the Cust_Address_Street_Address, Cust_Address_City, Cust_Address_State_Province, Cust_Address_Postal_Code, Cust_Address_Country_Id, Cust_Email, and Phone Number items of the `CUSTOMERS` block. Open the Property Palette for this multiple selection.

Set the Canvas property to TAB_CUSTOMER.

Set the Tab Page property to the CONTACT tab page.

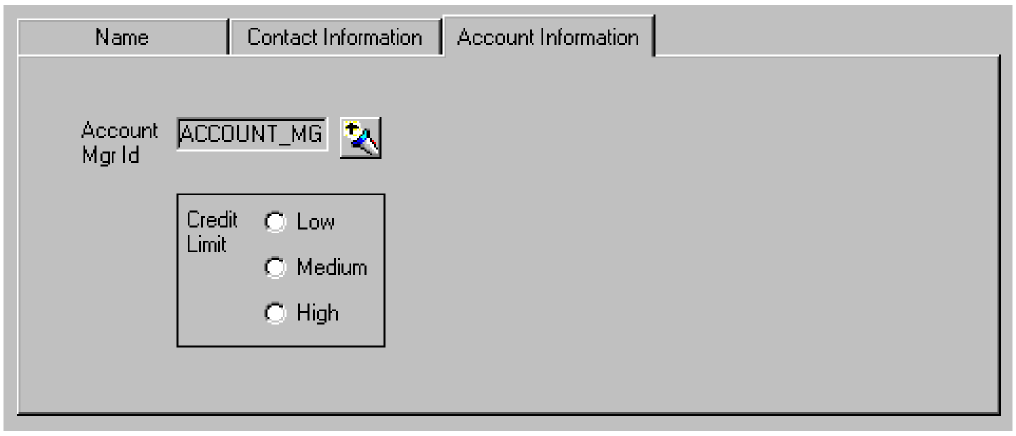
In the Layout Editor, arrange the items according to the screenshot. For the Phone_Numbers item, open the Property Palette and change the Prompt to Phone Numbers, the Prompt Attachment Edge to Top, and the Prompt Attachment Offset to 0.



Similarly, place the following items on the ACCOUNT tab page of the TAB_CUSTOMER canvas: Account_Mgr_ID and Credit_Limit from the `CUSTOMERS` block, and Account_Mgr_LOV_Button from the `CONTROL` block.

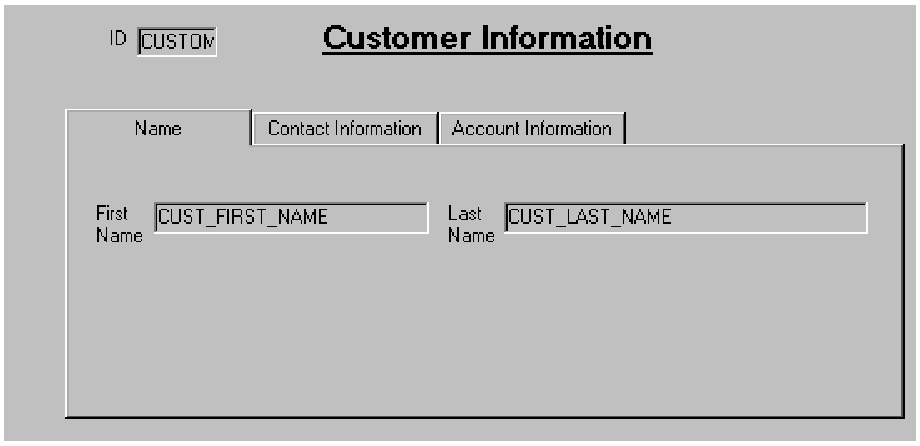
In the Object Navigator, select the text object under the Graphics node of the CV_Customer canvas that corresponds with the Credit Limit boilerplate text. Drag the text object to the Account tab page.

In the Layout Editor, arrange the items according to the screenshot and create a rectangle to surround the Credit Limit radio buttons.



Practice 12 Solutions (continued)

The content canvas should now look similar to the screenshot:



12. Reorder the items according to the tab page sequence. Ensure that the user moves smoothly from one tab page to another when tabbing through items. Set the Next Navigation Item and Previous Navigation Item properties to make it impossible to tab to the Customer_Id item once you tab out of that item initially.
    Note: Since Customer_Id is now the only item on the CV_CUSTOMER canvas, setting either its Enabled or Keyboard Navigable properties to No has the effect of making the item not visible. This is because Forms must be able to navigate to an item on a canvas in order to display that canvas' items.
    In the Object Navigator, reorder the items according to their order in the tab pages. Set the Previous Navigation Item and Next Navigation Items properties for the first and last items in the tab pages: For the Credit_Limit item, set Next Navigation Item to CUST_FIRST_NAME; for the Cust_First_Name item, set Previous Navigation Item to CREDIT_LIMIT.
13. Save and compile the form. Click Run Form to run your form and test the changes.
    No formal solution.

Practice 14 Solutions

1. In the CUSTG*XX* form, write a trigger to display the Account_Mgr_Lov when the Account_Mgr_Lov_Button is selected. To create the When-Button-Pressed trigger, use the Smart triggers feature. Find the relevant built-in in the Object Navigator under built-in packages, and use the “Paste Name and Arguments” feature.
   Right-click the Account_Mgr_Lov_Button in the Object Navigator, select SmartTriggers in the pop-up menu, and select the When-Button-Pressed trigger from the list.
   When-Button-Pressed on CONTROL.Account_Mgr_Lov_Button:
   ```
   IF SHOW_LOV(‘account_mgr_lov') THEN
        NULL;
   END IF;
   ```
2. Create a When-Window-Closed trigger at the form level in order to exit form.
   When-Window-Closed at the form level:
   ```
   EXIT_FORM;
   ```
3. Save, compile, and run the form. Test to see that the LOV is invoked when you press the Account_Mgr_Lov_Button and that the form exits when you close the Customer Information window.
   No formal solution.
4. In the ORDG*XX* form, write a trigger to display the Products_Lov when the Product_Lov_Button is selected.
   When-Button-Pressed on CONTROL.Product_Lov_Button:
   ```
   IF SHOW_LOV('products_lov') THEN
        NULL;
   END IF;
   ```
5. Write a trigger that exits the form when the Exit_Button is selected.
   When-Button-Pressed on CONTROL.Exit_Button:
   ```
   EXIT_FORM;
   ```
6. Save, compile, and run the form. Test to see that the LOV is invoked when you press the Product_Lov_Button and that the form exits when you press the Exit_Button.
   No formal solution.
7. Create a When-Button-Pressed trigger on CONTROL.Show_Help_Button that uses the SHOW_VIEW built-in to display the CV_HELP.
   When-Button-Pressed on CONTROL.Show_Help_Button:
   ```
   SHOW_VIEW('CV_HELP');
   ```
8. Create a When-Button-Pressed trigger on CONTROL.Hide_Help_Button that hides the CV_HELP. Use the HIDE_VIEW built-in to achieve this.
   When-Button-Pressed on CONTROL.Hide_Help_Button
   ```
   HIDE_VIEW('CV_HELP');
   ```

Practice 14 Solutions (continued)

9. Create a When-Button-Pressed trigger on `CONTROL`. Stock_Button that uses the `GO_BLOCK` built-in to display the `INVENTORIES` block.

   When-Button-Pressed on CONTROL.Stock_Button:

   ```
   GO_BLOCK('INVENTORIES');
   EXECUTE_QUERY;
   ```

10. Write a form-level When-Window-Closed trigger to hide the `WIN_INVENTORY` window if the user attempts to close it, and to exit the form if the user attempts to close the `WIN_ORDER` window.

    Hint: Use the system variable `:SYSTEM.TRIGGER_BLOCK` to determine what block the cursor is in when the trigger fires.

    When-Window-Closed trigger on form:

    ```
    IF :SYSTEM.TRIGGER_BLOCK = 'INVENTORIES' THEN
         GO_BLOCK('ORDERS');
    ELSE
         EXIT_FORM;
    END IF;
    ```

11. Save and compile the form. Click Run Form to run your form and test the changes.. The stacked canvas, `CV_HELP`, is displayed only if the current item will not be obscured. Ensure, at least, that the first entered item in the form is one that will not be obscured by `CV_HELP`.

    You might decide to advertise Help only while the cursor is in certain items, or move the stacked canvas to a position that does not overlay enterable items.

    The `CV_HELP` canvas, of course, could also be shown in its own window, if appropriate.

    No formal solution.

Practice 15 Solutions

1. Open your `CUSTG`*`XX`*.FMB file. In this form, create a procedure that is called List_Of_Values. Import code from the `pr15_1.txt` file:

   ```
   PROCEDURE list_of_values(p_lov in VARCHAR2,p_text in
   VARCHAR2) IS
     v_lov BOOLEAN;
   BEGIN
     v_lov:= SHOW_LOV(p_lov);
     IF v_lov = TRUE THEN
       MESSAGE('You have just selected a(n) '||p_text);
     ELSE
       MESSAGE('You have just cancelled the List of Values');
     END IF;
   END;
   ```

   Select the Program Units node in the Object Navigator. Click Create.
   In the New Program Unit window, type the name List_Of_Values, and click OK.
   In the PL/SQL Editor, select all the text and press [Delete] to delete it.
   From the menu, select File > Import PL/SQL Text.
   In the Import window, click the poplist in the "Files of type:" field to change the File type to All Files (*.*).
   Select `pr15_1.txt` and click Open.
   In the PL/SQL Editor, click Compile PL/SQL code, the icon at the upper left of the window.

2. Modify the When-Button-Pressed trigger of CONTROL.Account_Mgr_LOV_Button in order to call this procedure. Misspell the parameter to pass the LOV name.
   When-Button-Pressed on CONTROL.Account_Mgr_LOV_Button:
   `LIST_OF_VALUES('ACCOUNT_MGR_LO', 'Account Manager');`

3. Click Run Form to run your form and test the changes. Press the LOV button for the Account Manager. Notice that the LOV does not display, and you receive a message that 'You have just cancelled the List of Values'. **No formal solution.**

4. Now click Run Form Debug to run the form in debug mode. Set a breakpoint in your When-Button-Pressed trigger, and investigate the call stack. Try stepping through the code to monitor its progress. Look at the Variables panel to see the value of the parameters you passed to the procedure, and the value of the `p_lov` variable in the procedure. How would this information help you to figure out where the code was in error?
   Click Run Form Debug. I
   n Forms Builder, open the PL/SQL Editor for the When-Button-Pressed trigger on the CONTROL.Account_Mgr_LOV_Button. Double-click the gray area to the left of the code to create a breakpoint.

Practice 15 Solutions (continued)

In the running form, press the Account_Mgr_Button. The debugger takes control of the form because the breakpoint is encountered. The running form now appears as a blank applet.

In Forms Builder, the Debug Console is displayed. If the Stack panel and the Variables panel are not shown, click the appropriate icons of the Debug Console window to display them.

In the Forms Builder toolbar, click Step Into. This adds the List_Of_Values procedure to the Stack panel. In addition, some variables now appear in the Variables panel.

Resize the Value column of the Variables panel so that you can see the value of the variables. You can see that the `p_lov` value is incorrect.

Click Go in the Forms Builder toolbar to execute the remaining code. Control returns to the running form. Click Stop to dismiss the debugger.

Exit the form and the browser, then correct the code in the When-Button-Pressed trigger. You can then run the form again to retest it.

Practice 16 Solutions

1. In the CUSTG*XX* form, write a trigger that fires when the credit limit changes. The trigger should display a message warning the user if a customer's outstanding credit orders (those with an order status between 4 and 9) exceed the new credit limit. You can import the `pr16_1.txt` file.
   Create a When-Radio-Changed trigger on CUSTOMERS.Credit_Limit. With the the PL/SQL Editor open, select File > Import PL/SQL Text.
   In the Import window, click the poplist in the "Files of type:" field to change the File type to All Files (*.*).
   Select `pr16_1.txt` and click Open.
   In the PL/SQL Editor, click Compile PL/SQL code, the icon at the upper left of the window.
   When-Radio-Changed on CUSTOMERS.Credit_Limit:
```
DECLARE v_unpaid_orders NUMBER;
BEGIN
      SELECT SUM(nvl(unit_price,0)*nvl(quantity,0))
      INTO v_unpaid_orders
      FROM orders o, order_items i
      WHERE o.customer_id = :customers.customer_id
      AND o.order_id = i.order_id
      -- Unpaid credit orders have status between 4 and 9
      AND (o.order_status > 3 AND o.order_status < 10);
      IF v_unpaid_orders > :customers.credit_limit THEN
            MESSAGE('This customer''s current orders exceed the
            new credit limit');
      END IF;
END;
```
2. Click Run Form to run your form and test the functionality.
   Hint: Most customers who have outstanding credit orders exceed the credit limits, so you should receive the warning for most customers. (If you wish to see a list of customers and their outstanding credit orders, run the `CreditOrders.sql` script in SQL*Plus.) Customer 120 has outstanding credit orders of less than $500, so you shouldn't receive a warning when changing this customer's credit limit.
   No formal solution.
3. Begin to implement a JavaBean for the ColorPicker bean area on the `CONTROL` block that will enable a user to choose a color from a color picker.
   Create a button on the CV_CUSTOMER canvas to enable the user to change the canvas color using the ColorPicker bean.
   Set the following properties on the button:

Label: `Canvas Color`	Mouse Navigate: No
Keyboard Navigable: No	Background color: gray

Practice 16 Solutions (continued)

The button should call a procedure named PickColor, with the imported text from the the `pr16_3.txt` file.

The bean will not function at this point, but you will write the code to instantiate it in Practice 20.

Create a procedure under the Program Units node of the Object Navigator. Name the procedure `PickColor`. With the PL/SQL Editor open, select File > Import Text. The code in `pr16_3.txt` is:

```
PROCEDURE PickColor(pvcTarget in VARCHAR2) IS
  vcOldColor VARCHAR2(12 char);
  vcNewColor VARCHAR2(12 char);
  hCanvas CANVAS := FIND_CANVAS('CV_CUSTOMER');
  hColorPicker ITEM := FIND_ITEM('CONTROL.COLORPICKER');
  vnPos1 NUMBER;
  vnPos2 NUMBER;
BEGIN
  -- First get the current Color for this target
  vcOldColor := get_canvas_property(hCanvas,background_color);
  vnPos1 := instr(vcOldColor,'g',1);
  vnPos2 := instr(vcOldColor,'b',vnPos1 + 1);
  vcOldColor := substr(vcOldColor,2,vnPos1 - 2)||
    ' '||substr(vcOldColor,vnPos1+1,vnPos2 - vnPos1 -1)||
    ' '||substr(vcOldColor,vnPos2 + 1,length(vcOldColor)-vnPos2);
  -- now display the picker with that color as an initial value
  vcNewColor :=
  FBean.Invoke_char(hColorPicker,1,'showColorPicker','"Select
    color for '||pvcTarget||'","'||vcOldColor||'"');
  -- finally if vcColor is not null reset the Canvas property
  if (vcNewColor is not null) then
    vnPos1 := instr(vcNewColor,' ',1);
    vnPos2 := instr(vcNewColor,' ',vnPos1 + 1);
    vcNewColor := 'r'||substr(vcNewColor,1,vnPos1 - 1)||
      'g'||substr(vcNewColor,vnPos1+1,vnPos2 - vnPos1 -1)||
      'b'||substr(vcNewColor,vnPos2 + 1,length(vcNewColor)-
      vnPos2);
    set_canvas_property(hCanvas,background_color,vcNewColor);
  end if;
END;
```

Create a button in the CONTROL block called `Color_Button` on the CV_CUSTOMER canvas, setting its properties as shown above. Define a When-Button-Pressed trigger for the button:

```
      PickColor('Canvas');
```

4. Save and compile the form. You will not be able to test the Color button yet, because the bean does not function until you instantiate it in Practice 20.
   No formal solution.

Practice 16 Solutions (continued)

5. In the ORDG*XX* form CONTROL block, create a new button called Image_Button and position it on the toolbar. Set the Label property to Image Off. Set the navigation and color properties like the other toolbar buttons.
   Display the Layout Editor. Make sure the Toolbar canvas and the Control block are selected.
   Select the Button tool.
   Create a button and place it on the toolbar.
   Set the Name to `Image_Button`.
   Set the Keyboard Navigable property to No.
   Set the Mouse Navigate property to No.
   Set the Label property to `Image Off`.
   Set the Background Color property to `white`.
6. Import the `pr16_6.txt` file into a trigger that fires when the Image_Button is clicked. The file contains code that determines the current value of the visible property of the Product Image item. If the current value is True, the visible property toggles to False for both the Product Image item and the Image Description item. Finally, the label changes on the Image_Button to reflect its next toggle state. However, if the visible property is currently False, the visible property toggles to True for both the Product Image item and the Image Description item.
   Create a When-Button-Pressed trigger on CONTROL.Image_Button. With the the PL/SQL Editor open, select File > Import Text.
   In the Import window, click the poplist in the "Files of type:" field to change the File type to All Files (*.*).
   Select `pr16_6.txt` and click Open.
   When-Button-Pressed on Control.Image_Button:

```
IF GET_ITEM_PROPERTY('CONTROL.product_image',VISIBLE)='TRUE' THEN
  SET_ITEM_PROPERTY('CONTROL.product_image', VISIBLE,
   PROPERTY_FALSE);
  SET_ITEM_PROPERTY('ORDER_ITEMS.image_description',
   VISIBLE,PROPERTY_FALSE);
  SET_ITEM_PROPERTY('CONTROL.image_button',LABEL,'Image On');
ELSE
  SET_ITEM_PROPERTY('CONTROL.product_image', VISIBLE,
   PROPERTY_TRUE);
  SET_ITEM_PROPERTY('ORDER_ITEMS.image_description',
   VISIBLE,PROPERTY_TRUE);
  SET_ITEM_PROPERTY('CONTROL.image_button',LABEL,'Image Off');
END IF;
```

7. Save and compile the form. Click Run Form to run your form. **Note:** The image will not display in the image item at this point; you will add code to populate the image item in Practice 20.
   No formal solution.

Practice 17 Solutions

1. Create an alert in CUSTG*XX* called Credit_Limit_Alert with one OK button. The message should read “This customer's current orders exceed the new credit limit”.
   Create an alert.
   Set Name to `Credit_Limit_Alert`.
   Set Title to `Credit Limit`.
   Set Alert Style to Caution.
   Set Button1 Label to `OK`.
   Set Message to “`This customer's current orders exceed the new credit limit`”
   Remove the labels for the other buttons.
2. Alter the When-Radio-Changed trigger on Credit_Limit to show the Credit_Limit_Alert instead of the message when a customer’s credit limit is exceeded.
   When-Radio-Changed on ORDERS.Credit_Limit (arrows denote changed or added lines):

```
    DECLARE
---->  n NUMBER;
       v_unpaid_orders NUMBER;
    BEGIN
       SELECT SUM(nvl(unit_price,0)*nvl(quantity,0))
       INTO v_unpaid_orders
       FROM orders o, order_items i
       WHERE o.customer_id = :customers.customer_id
       AND o.order_id = i.order_id
       -- Unpaid credit orders have status between 4 and 9
       AND (o.order_status > 3 AND o.order_status < 10);
       IF v_unpaid_orders > :customers.credit_limit THEN
---->     n := SHOW_ALERT('credit_limit_alert');
       END IF;
    END;
```

3. Save and compile the form. Click Run Form to run your form and test the changes.
4. Create a generic alert in ORDG*XX* called Question_Alert that allows Yes and No replies.
   Create an alert.
   Set Name to `QUESTION_ALERT`.
   Set Title to `Question`.
   Set Alert Style to Stop.
   Set Button1 Label to `Yes`.
   Set Button2 Label to `No`.

Practice 17 Solutions (continued)

5. Alter the When-Button-Pressed trigger on `CONTROL`. Exit_Button to use the Question_Alert to ask the operator to confirm that the form should terminate. (You can import the text from `pr17_5.txt`.)

   When-Button-Pressed on CONTROL.Exit_Button:

```
SET_ALERT_PROPERTY('Question_Alert',
  ALERT_MESSAGE_TEXT,
  'Do you really want to leave the form?');
IF SHOW_ALERT('Question_Alert') = ALERT_BUTTON1 THEN
     EXIT_FORM;
END IF;
```

6. Save and compile the form. Click Run Form to run your form and test the changes.
   No formal solution.

Practice 18 Solutions

1. In the ORDG*XX* form, write a trigger that populates the Customer_Name and the Sales_Rep_Name for every row fetched by a query on the `ORDERS` block. You can import the text from `pr18_1.txt`.

   Post-Query on ORDERS block:

   ```
   BEGIN
     IF :orders.customer_id IS NOT NULL THEN
        SELECT cust_first_name || ' ' || cust_last_name
        INTO :orders.customer_name
        FROM customers
        WHERE customer_id = :orders.customer_id;
     END IF;
     IF :orders.sales_rep_id IS NOT NULL THEN
        SELECT first_name || ' ' || last_name
             INTO :orders.sales_rep_name
             FROM employees
             WHERE employee_id = :orders.sales_rep_id;
     END IF;
   END;
   ```

2. Write a trigger that populates the Description for every row fetched by a query on the `ORDER_ITEMS` block. You can import the text from `pr18_2.txt`.

   Post-Query on `ORDER_ITEMS` block:

   ```
   BEGIN
     IF :order_items.product_id IS NOT NULL THEN
        SELECT translate(product_name using char_cs)
        INTO :order_items.description
        FROM products
        WHERE product_id = :order_items.product_id;
     END IF;
   END;
   ```

3. Change the When-Button-Pressed trigger on the Stock_Button in the `CONTROL` block so that users will be able to execute a second query on the `INVENTORIES` block that is not restricted to the current Product_ID in the `ORDER_ITEMS` block. You can import the text from `pr18_3.txt`.

   When-Button-Pressed on Stock_Button:

   ```
   SET_BLOCK_PROPERTY('INVENTORIES',ONETIME_WHERE,
      'product_id='||:ORDER_ITEMS.product_id);
   GO_BLOCK('INVENTORIES');
   EXECUTE_QUERY;
   ```

Practice 18 Solutions (continued)

4. Ensure that the Exit_Button has no effect in Enter-Query mode.
   Set Fire in Enter-Query Mode property to No for the When-Button-Pressed trigger.
5. Click Run Form to run your form and test the changes.
   No formal solution.
6. Open the CUSTG*XX* form module. Adjust the default query interface. Add a check box called `CONTROL`. Case_Sensitive to the form so that the user can specify whether or not a query for a customer name should be case sensitive. Place the check box on the Name page of the TAB_CUSTOMER canvas. You can import the `pr18_6.txt` file into the
   When-Checkbox-Changed trigger. Set the initial value property to Y, and the Checked/Unchecked properties to Y and N.
   Set the Mouse Navigate property to No.
   Open the Layout Editor and check that the canvas is TAB_CUSTOMER and the block is `CONTROL`. Click the Name tab.
   Using the Checkbox tool, place a check box on the canvas.
   Open the Property Palette for the new check box. Set the Name to `CASE_SENSITIVE`, the Initial Value to `Y`, Value when Checked to `Y`, Value when Unchecked to `N`, and Mouse Navigate to No. Set the Label to: `Perform case sensitive query on name?`
   In the Layout Editor, resize the check box so that the label fully displays.
   When-Checkbox-Changed trigger on the `CONTROL`. Case_Sensitive item (checkbox):

```
IF NVL(:CONTROL.case_sensitive, 'Y') = 'Y' THEN
  SET_ITEM_PROPERTY('CUSTOMERS.cust_first_name',
    CASE_INSENSITIVE_QUERY, PROPERTY_FALSE);
  SET_ITEM_PROPERTY('CUSTOMERS.cust_last_name',
    CASE_INSENSITIVE_QUERY, PROPERTY_FALSE);
ELSE
  SET_ITEM_PROPERTY('CUSTOMERS.cust_first_name',
    CASE_INSENSITIVE_QUERY, PROPERTY_TRUE);
  SET_ITEM_PROPERTY('CUSTOMERS.cust_last_name',
    CASE_INSENSITIVE_QUERY, PROPERTY_TRUE);
END IF;
```

Practice 18 Solutions (continued)

7. Add a check box called `CONTROL`. Exact_Match to the form so that the user can specify whether or not a query condition for a customer name should exactly match the table value. (If a nonexact match is allowed, the search value can be part of the table value.) Set the label to: Exact match on query?
   Set the initial value property to Y, and the Checked/Unchecked properties to Y and N. Set the Mouse Navigate property to No. You can import the `pr18_7.txt` file into the Pre-Query Trigger.
   Create the check box as in 18-6 and set its properties.
   Pre-Query trigger on the CUSTOMERS block:

```
IF NVL( :CONTROL.exact_match, 'Y' ) = 'N' THEN
     IF :customers.cust_first_name IS NOT NULL THEN
          :CUSTOMERS.cust_first_name := '%' ||
          :CUSTOMERS.cust_first_name || '%';
     END IF;
     IF :customers.cust_last_name IS NOT NULL THEN
          :CUSTOMERS.cust_last_name := '%' ||
          :CUSTOMERS.cust_last_name || '%';
     END IF;
END IF;
```

8. Ensure that the When-Radio-Changed trigger for the Credit_Limit item does not fire when in Enter-Query mode.
   Open the Property Palette for the When-Radio-Changed trigger on the Credit_Limit item. Set Fire in Enter-Query Mode property to No
9. Click Run Form to run your form and test the changes.
   No formal solution.

Practice 19 Solutions

1. In the CUSTG*XX* form, cause the Account_Mgr_Lov to be displayed whenever the user enters an Account_Mgr_Id that does not exist in the database.
   Set the Validate from List property to Yes for the Account_Mgr_Id item in the CUSTOMERS block.
2. Save and compile the form. Click Run Form to run your form and test the changes.
   No formal solution.
3. In the ORDG*XX* form, write a validation trigger to check that if the Order_Mode is online, the Order_Status indicates a CREDIT order (values between 4 and 10). You can import the text from `pr19_3.txt`.
   When-Validate-Record on ORDERS block (shows error if an online order is not a credit order):
```
IF :ORDERS.order_mode = 'online' and
  :ORDERS.order_status not between 4 and 10 THEN
     MESSAGE('Online orders must be CREDIT orders');
     RAISE form_trigger_failure;
END IF;
```
4. In the ORDG*XX* form, create a trigger to write the correct values to the Customer_Name, Sales_Rep_Name, and Sales_Rep_Id items whenever validation occurs on Customer_Id. Fail the trigger if the data is not found. You can import text from `pr19_4a.txt` and `pr19_4b.txt`.
   When-Validate-Item on ORDERS.Customer_Id:
```
SELECT cust_first_name || ' ' || cust_last_name
INTO :orders.customer_name
FROM customers
WHERE customer_id = :orders.customer_id;
EXCEPTION
     WHEN NO_DATA_FOUND THEN
       MESSAGE('Invalid customer id');
       RAISE form_trigger_failure;
```
   When-Validate-Item on ORDERS.Sales_Rep_Id:
```
SELECT first_name || ' ' || last_name
  INTO :orders.sales_rep_name
  FROM employees
  WHERE employee_id = :orders.sales_rep_id
  AND job_id = 'SA_REP';
EXCEPTION
     WHEN NO_DATA_FOUND THEN
          MESSAGE('Invalid sales rep id');
          RAISE form_trigger_failure;
```

Practice 19 Solutions (continued)

5. Create another validation trigger on ORDER_ITEMS.Product_Id to derive the name of the product and suggested wholesale price, and write them to the Description item and the Price item. Fail the trigger and display a message if the product is not found. You can import the text from `pr19_5.txt`.

   When-Validate-Item on ORDER_ITEMS.Product_Id:

   ```
   SELECT product_name, list_price
   INTO :order_items.description,
        :order_items.unit_price
   FROM products
   WHERE product_id = :order_items.product_id;
   EXCEPTION
        WHEN NO_DATA_FOUND THEN
             MESSAGE('Invalid product id');
             RAISE form_trigger_failure;
   ```

6. Perform client-side validation on the ORDER_ITEMS.Quantity item using a Pluggable Java Component to filter the keystrokes and allow only numeric values. The full path to the PJC class is `oracle.forms.demos.KeyFilter` (this is case sensitive), to be used as the Implementation Class for the item. You will set the filter for the item in the next practice, so the validation is not yet functional. **Open the Property Palette for the :ORDER_ITEMS.Quantity item. Set the Implementation Class property to: `oracle.forms.demos.KeyFilter`**

7. Save and compile the form. Click Run Form to run your form and test the changes. Do not test the validation on the Quantity item because it will not function until after you set the filter on the item in Practice 20.
   No formal solution.

Practice 20 Solutions

1. When the ORDG*XX* form first opens, set a filter on the ORDER_ITEMS. Quantity Pluggable Java Component, and execute a query . You can import the code for the trigger from `pr20_1.txt`.
   When-New-Form-Instance at form level:
   ```
   SET_CUSTOM_PROPERTY('order_items.quantity',1,
     'FILTER_TYPE','NUMERIC');
   GO_BLOCK('ORDERS');
   EXECUTE_QUERY;
   ```
2 Write a trigger that fires as the cursor arrives in each record of the ORDER_ITEMS block to populate the Product_Image item with a picture of the product, if one exists. First create a procedure called get_image to populate the image, then call that procedure from the appropriate trigger. You can import the code for the procedure from `pr20_2.txt`.
   Get_Image procedure
   ```
   PROCEDURE get_image IS
        filename VARCHAR2(250);
   BEGIN
        filename := to_char(:order_items.product_id)
                    || '.jpg';
        READ_IMAGE_FILE(filename,'jpeg',
                        'control.product_image');
   END;
   ```
   When-New-Record-Instance on ORDER_ITEMS block:
   ```
   get_image;
   ```
3. Define the same trigger type and code on the ORDERS block.
   When-New-Record-Instance on ORDERS block: `get_image;`
4. Is there another trigger where you might also want to place this code?
   When-Validate-Item on ORDER_ITEMS.Product_Id is a candidate for the code.
5. Save and compile the form. Click Run Form to run your form and test the changes.
   No formal solution.

Practice 20 Solutions (continued)

6. Notice that you receive an error if the image file does not exist. Code a trigger to gracefully handle the error by populating the image item with a default image called `blank.jpg`. You can import the code from `pr20_6.txt`.
   On-Error trigger on ORDGXX form:
   ```
   IF ERROR_CODE = 47109 THEN
        READ_IMAGE_FILE('blank.jpg','jpeg',
                        'control.product_image');
   ELSE
        MESSAGE(ERROR_TYPE || '-' || TO_CHAR(ERROR_CODE) ||
               ': ' || ERROR_TEXT);
   END IF;
   ```
7. The image item has a lot of blank space when the image does not take up the entire area. To make it look better, set its Background Color of both the CONTROL.Product_Image item and the CV_ORDER canvas to the same value, such as r0g75b75. Set the Bevel for the Product_Image item to None.
   With both the CONTROL.Product_Image item and the CV_ORDER canvas selected in the Object Navigator, open the Property Palette. Set the Background Color for both objects to r0g75b75. With just the Product_Image item selected, open the Property Palette and set Bevel to None.
8. Click Run Form to run your form again and test the changes.
   No formal solution.
9. In the CUSTG*XX* form, register the ColorPicker bean (making its methods available to Forms) when the form first opens, and also execute a query on the CUSTOMERS block. You can import the code from `pr20_9.txt`.
   When-New-Form-Instance trigger on the CUSTG*XX* form:
   ```
        FBean.Register_Bean('control.colorpicker',1,
             'oracle.forms.demos.beans.ColorPicker');
        GO_BLOCK(‘customers’);
        EXECUTE_QUERY;
   ```
10. Save, compile, and click Run Form to run your form and test the Color button. You should be able to invoke the ColorPicker bean from the Color button, now that the bean has been registered at form startup.
    No formal solution.

Practice 21 Solutions

1. In the ORDG*XX* form, write a transactional trigger on the `ORDERS` block that populates ORDERS.Order_Id with the next value from the ORDERS_SEQ sequence. You can import the code from `pr21_1.txt`.
   Pre-Insert on ORDERS block:
   ```
   SELECT orders_seq.NEXTVAL INTO :orders.order_id
     FROM sys.dual;
   EXCEPTION
        WHEN OTHERS THEN
             MESSAGE('Unable to assign order id');
             RAISE form_trigger_failure;
   ```
2. In the `ORDERS` block, set the Enabled property for the Order_ID item to No. Set the Required property for the Order_ID item to No. To ensure that the data remains visible, set the Background Property to gray.
   In the Property Palette, set the Enabled and Required properties to No for ORDERS.Order_Id. Set Background Color to gray.
3. Save, compile, and run the form to test.
   No formal solution.
4. Create a similar trigger on the `ORDER_ITEMS` block that assigns the Line_Item_Id when a new record is saved. Set the properties for the item as you did on ORDERS.ORDER_ID. You can import the code from `pr21_4.txt`.
   Pre-Insert on ORDER_ITEMS block:
   ```
   SELECT NVL(MAX(line_item_id),0) + 1
   INTO :ORDER_ITEMS.line_item_id
        FROM ORDER_ITEMS
        WHERE :ORDER_ITEMS.order_id = order_id;
   EXCEPTION
   WHEN OTHERS THEN
        MESSAGE('Unable to assign line item id');
        RAISE form_trigger_failure;
   ```
   Set the Required and Enabled properties to No and the Background Color to gray for ORDER_ITEMS.Line_Item_Id.
5. Save and compile the form. Click Run Form to run your form and test.
   No formal solution.

Instructor Note

Solution 21-4 is not the safest way. The better solution is to keep the total number of rows in another table that you can lock, but this solution is too advanced at this stage.

Practice 21 Solutions (continued)

6. Open the CUSTG*XX* form module. Create three global variables called `GLOBAL.INSERT`, `GLOBAL.UPDATE`, and `GLOBAL.DELETE`. These variables indicate respectively the number of inserts, updates, and deletes. You need to write Post-Insert, Post-Update, and Post-Delete triggers to initialize and increment the value of each global variable.

   Post-Insert at form level:

```
      DEFAULT_VALUE('0', 'GLOBAL.insert');
      :GLOBAL.insert := TO_CHAR( TO_NUMBER( :GLOBAL.insert ) + 1 );
            Post-Update at form level:
            DEFAULT_VALUE('0', 'GLOBAL.update');
      :GLOBAL.update := TO_CHAR( TO_NUMBER( :GLOBAL.update ) + 1 );
            Post-Delete at form level:
      DEFAULT_VALUE('0', 'GLOBAL.delete');
      :GLOBAL.delete := TO_CHAR( TO_NUMBER( :GLOBAL.delete ) + 1 );
```

7. Create a procedure called `HANDLE_MESSAGE`. Import the `pr21_7a.txt` file. This procedure receives two arguments. The first one is a message number, and the second is a Boolean error indicator. This procedure uses the three global variables to display a customized commit message and then erases the global variables.

```
PROCEDURE handle_message( message_number IN NUMBER,
message_line IN VARCHAR2 ) IS
      BEGIN
      IF message_number IN ( 40400, 40406, 40407 ) THEN
            DEFAULT_VALUE( '0', 'GLOBAL.insert' );
            DEFAULT_VALUE( '0', 'GLOBAL.update' );
            DEFAULT_VALUE( '0', 'GLOBAL.delete' );
            MESSAGE('Save Ok: ' ||
            :GLOBAL.insert || ' records inserted, ' ||
            :GLOBAL.update || ' records updated, ' ||
            :GLOBAL.delete || ' records deleted !!!' );
            ERASE('GLOBAL.insert');
            ERASE('GLOBAL.update');
            ERASE('GLOBAL.delete');
      ELSE
            MESSAGE( message_line );
      END IF;
END ;
```

Practice 21 Solutions (continued)

Call the procedure when an error occurs. Pass the error code and an error message to be displayed. You can import the code from `pr21_7b.txt`.

Call the procedure when a message occurs. Pass the message code and a message to be displayed. You can import the code from `pr21_7c.txt`.

On-Error at form level:

```
handle_message( error_code, 'ERROR: ' || ERROR_TYPE ||
'-' || TO_CHAR(ERROR_CODE) || ': '|| ERROR_TEXT );
```

On-Message at form level:

```
handle_message( message_code, MESSAGE_TYPE || '-' ||
TO_CHAR(MESSAGE_CODE) || ': ' || MESSAGE_TEXT );
```

8. Write an On-Logon trigger to control the number of connection tries. Use the `LOGON_SCREEN` built-in to simulate the default login screen and `LOGON` to connect to the database. You can import the `pr21_8.txt` file.

On-Logon at form level:

```
DECLARE
     connected BOOLEAN := FALSE;
     tries NUMBER := 3;
     un VARCHAR2(30);
     pw VARCHAR2(30);
     cs VARCHAR2(30);
BEGIN
     SET_APPLICATION_PROPERTY(CURSOR_STYLE, 'DEFAULT');
     WHILE connected = FALSE and tries > 0 LOOP
       LOGON_SCREEN;
       un := GET_APPLICATION_PROPERTY( USERNAME );
       pw := GET_APPLICATION_PROPERTY( PASSWORD );
       cs := GET_APPLICATION_PROPERTY
             ( CONNECT_STRING );
       LOGON( un, pw || '@' || cs, FALSE );
       IF FORM_SUCCESS THEN
          connected := TRUE ;
       END IF;
       tries := tries - 1;
     END LOOP;
     IF NOT CONNECTED THEN
          MESSAGE('Too many tries!');
          RAISE FORM_TRIGGER_FAILURE;
     END IF;
END;
```

9. Click Run Form to run your form and test the changes.

No formal solution.

Practice 22 Solutions

1. In the ORDG*XX* form, alter the code called by the triggers that populate the Product_Image item when the image item is displayed.

   Add a test in the code to check Product_Image. Perform the trigger actions only if the image is currently displayed. Use the `GET_ITEM_PROPERTY` built-in function. The code is contained in `pr22_1.txt`.

   The `get_image` procedure is called by When-New-Record-Instance on ORDERS and ORDER_ITEMS blocks. Modify the procedure under the Program Units node as follows (note the change in the call to `READ_IMAGE_FILE` denoted by italicized plain text):

```
PROCEDURE get_image IS
  product_image_id ITEM :=
    FIND_ITEM('control.product_image');
  image_button_id ITEM:=
    FIND_ITEM('CONTROL.image_button');
  filename VARCHAR2(250);
BEGIN
  IF GET_ITEM_PROPERTY(product_image_id,VISIBLE)='TRUE'
  THEN
    filename := to_char(:order_items.product_id) ||'.jpg';
    READ_IMAGE_FILE(filename,'jpeg',product_image_id);
  END IF;
END;
```

Practice 22 Solutions (continued)

2. Alter the When-Button-Pressed trigger on the Image_Button so that object IDs are used.

   Use a FIND_*object* function to obtain the IDs of each item referenced by the trigger. Declare variables for these IDs, and use them in each item reference in the trigger. The code is contained in `pr22_2.txt`.

   When-Button-Pressed on CONTROL.image_button:

```
DECLARE
  product_image_id ITEM :=
     FIND_ITEM('CONTROL.product_image');
  image_desc_id ITEM :=
     FIND_ITEM('ORDER_ITEMS.image_description');
  image_button_id ITEM :=  FIND_ITEM('CONTROL.image_button');
BEGIN
  IF GET_ITEM_PROPERTY(product_image_id, VISIBLE)='TRUE'
  THEN
    SET_ITEM_PROPERTY(product_image_id, VISIBLE,
                      PROPERTY_FALSE);
    SET_ITEM_PROPERTY(image_desc_id,VISIBLE,
                      PROPERTY_FALSE);
    SET_ITEM_PROPERTY(image_button_id,LABEL,'Image On');
  ELSE
    SET_ITEM_PROPERTY(product_image_id, VISIBLE,
                      PROPERTY_TRUE);
    SET_ITEM_PROPERTY(image_desc_id,VISIBLE,PROPERTY_TRUE);
    SET_ITEM_PROPERTY(image_button_id,LABEL,'Image Off');
  END IF;
END;
```

3. Create a button called Blocks_Button in the `CONTROL` block and place it on the Toolbar canvas. Label the button Show Blocks. Set its navigation and color properties the same as the other toolbar buttons.
   The code for the button should print a message showing what block the user is currently in. It should keep track of the block and item where the cursor was located when the trigger was invoked (`:SYSTEM.CURSOR_BLOCK` and `:SYSTEM.CURSOR_ITEM`). It should then loop through the remaining navigable blocks of the form and print a message giving the names (`:SYSTEM.CURRENT_BLOCK`) of all the navigable blocks in the form. Finally, it should navigate back to the block and item where the cursor was located when the trigger began to fire. Be sure to set the Mouse Navigate property of the button to No. You may import the code for the trigger from `pr22_3.txt`.
   Create a button on the TOOLBAR canvas and set its properties:

Name: Blocks_Button	**Label: Show Blocks**
Mouse Navigate: No	**Keyboard Navigable: No**
Height: 16	**Background Color: white**

   When-Button-Pressed on Blocks_Button:
```
DECLARE
  vc_startblk VARCHAR2(30) := :SYSTEM.cursor_block;
  vc_startitm VARCHAR2(30) := :SYSTEM.cursor_item;
  vc_otherblks VARCHAR2(80) := NULL;
BEGIN
  MESSAGE('You are in the ' || vc_startblk || ' block.');
  NEXT_BLOCK;
  WHILE :SYSTEM.current_block != vc_startblk LOOP
    vc_otherblks := vc_otherblks || ' ' ||
                    :system.current_block;
    NEXT_BLOCK;
  END LOOP;
  message('Other block(s) in the form:‘ || vc_otherblks);
  GO_BLOCK(vc_startblk);
  GO_ITEM(vc_startitm);
END;
```

4. Save, compile, and run the form to test these features.
   No formal solution.

5. The trigger code above is generic, so it will work with any form. Open the CUSTG*XX* form and define a similar Blocks_Button, labeled Show Blocks, in the `CONTROL` block, and place it just under the Color button on the `CV_CUSTOMER` canvas. Drag the When-Button-Pressed trigger you created for the Blocks_Button of the ORDG*XX* form to the Blocks_Button of the CUSTG*XX* form. Run the CUSTG*XX* form to test the button.
   No formal solution. The code should print messages about the blocks in the CUSTOMERS form, with no code changes.

Practice 23 Solutions

1. In the ORDG*XX* form, create an object group, called Stock_Objects, consisting of the `INVENTORIES` block, `CV_INVENTORY` canvas, and `WIN_INVENTORY` window.
   Select the Object Groups node and click the Create icon. Rename the group to `STOCK_OBJECTS`.
   Drag the `INVENTORIES` block, `CV_INVENTORY` canvas, and `WIN_INVENTORY` window under the Object Group Children entry.
2. Save the form.
   No formal solution.
3. Create a new form module and copy the Stock_Objects object group into it.
   Select the Forms node and click the Create icon.
   Drag Stock_Objects from the ORDERS form to your new module under the Object Groups node. Select the Copy option.
4. In the new form module, create a property class called ClassA. Include the following properties and settings:

Font Name:	Arial
Format Mask:	99,999
Font Size:	8
Justification:	Right
Delete Allowed:	No
Background Color:	DarkRed
Foreground Color:	Gray

   Select the Property Classes node and click the Create icon.
   In the Property Palette for the property class, set the Name to `CLASSA`.
   Add the properties listed by clicking the Add Property icon and selecting from the list displayed.
   Set the properties to the values listed above.
5. Apply ClassA to `CV_INVENTORY` and the Quantity_on_Hand item.
   In the Property Palette for each of the objects, select the Subclass Information property, and click More. In the Subclass Information dialog box, select the Property Class radio option and set the Property Class Name list item to ClassA.
6. Save the form module as `STOCK`*XX*.fmb, compile, and run the form and note the error.
   You should receive the following error:
   FRM-30047: Cannot resolve item reference ORDER_ITEMS.PRODUCT_ID.

Practice 23 Solutions (continued)

7. Correct the error. Save, compile, and run the form again.
   Make Copy Value from Item a variant property. In the Property Palette for PRODUCT_ID, delete ORDER_ITEMS.PRODUCT_ID from the Copy Value from Item Property.
   The form should run without error and show the objects and properties from the Object Group and the Property Class.
8. Create an object library and name it `summit_olb`.
   Create two tabs in the object library called Personal and Corporate.
   Add the `CONTROL` block, the Toolbar, and the Question_Alert from the Orders form to the Personal tab of the object library.
   Save the object library as `summit.olb`.
   Select the Object Libraries node in the Object Navigator, and click Create. Rename this object library `SUMMIT_OLB`.
   Select the Library Tabs node in the Object Navigator.
   Click Create twice to create two tabs. Set the Name and Label properties for the first tab as `Personal` and for the second tab as `Corporate`.
   Open the object library by double-clicking its icon in the Object Navigator.
   From the ORDG*XX* form, drag the `CONTROL` block, the Toolbar, and the Question_Alert to the Personal tab of the object library.
   Save the `Summit` object library.
9. Create a new form, and create a data block based on the `DEPARTMENTS` table, including all columns. Use the Form layout style.
   Drag the Toolbar canvas, `CONTROL` block, and Question_Alert from the object library into the new form and select to **subclass** the objects. For proper behavior, the `DEPARTMENTS` block must precede the `CONTROL` block in the Object Navigator.
   Some items are not applicable to this form. Set the Canvas property for the following items to NULL: Image_Button, Stock_Button, Show_Help_Button, Product_Lov_Button, Hide_Help_Button, Product_Image, Total.
   The code of some of the triggers does not apply to this form. Set the code for the When-Button-Pressed triggers for the above buttons to: `NULL;`
   For the Total item, set the Calculation Mode and Summary Function properties to None, and set the Summarized Block property to Null.
   Use Toolbar as the Horizontal Toolbar canvas for this form.
   Set the Window property to WINDOW1 for the Toolbar canvas.
   Set the Horizontal Toolbar Canvas property to TOOLBAR for the window.
   Follow the practice steps; for a solution see the `DEPTwk23.fmb` file in the Solutions directory.
10. Save this form as DEPTG*XX*, compile, and run the form to test it.
    No formal solution.
    Notice the color of the Exit button is white; you will change this shortly in the Object Library to see how it affects subclassed objects.

Practice 23 Solutions (continued)

11. Try to delete items on the Null canvas. What happens and why?
    You cannot delete the objects because the toolbar and contents are subclassed from another object. If you had copied the objects, you would have been able to delete items, but any changes made to the objects in the Object Library would not be reflected in the form.
12. Change the Exit button of the Object Library's CONTROL block to have a gray background. Run the Departments form again to see that the Exit button is now gray.
    Create a new form module and drag the CONTROL block into it from the Object Library. Use the Copy option.
    In the form, change the Background Color of the Exit button to gray.
    Drag the CONTROL block back into the Object Library. Answer Yes to the Alert: "An object with this name already exists. Replace it with the new object?"
    You can delete the new form module, because you will not be using it. You created it only to use in editing the Object Library.
    Run the Departments form again. You should see the Exit button is now gray, rather than white as it was before.
13. Create two sample buttons, one for wide buttons and one for medium buttons, by means of width.
    Create a sample date field. Set the width and the format mask to your preferred standard.
    Drag these items to the Corporate tab of your object library.
    Mark these items as SmartClasses.
    Create a new form and a new data block in the form. Apply these SmartClasses in your form. Place the Toolbar canvas in the new form.
    No formal solution. See `Employeeswk23.fmb` in the `Solutions` directory for an example.
14. In the Orders form, note the similarity of the code in the Post-Query trigger of the Orders block and in the When-Validate-Item triggers of the Orders.Customer_Id and Orders.Sales_Rep_Id items. Move the similar code to PL/SQL program units and call the program units from the triggers, then run the form to test the changes..
    In the PL/SQL Editor, open the When-Validate-Item trigger for the Orders.Customer_Id item. Copy the code.
    In the Object Navigator, select the Program Units node and click Create. Enter the procedure name `GET_CUSTOMER_NAME` and click OK. Paste the copied code into the program unit and click Compile. The code should be as follows:

```
PROCEDURE GET_CUSTOMER_NAME IS
BEGIN
  SELECT cust_first_name || ' ' || cust_last_name
    INTO :orders.customer_name
    FROM customers
  WHERE customer_id = :orders.customer_id;
```

```
EXCEPTION
     WHEN NO_DATA_FOUND THEN
     MESSAGE('Invalid customer id');
     RAISE form_trigger_failure;
END;
```

In the PL/SQL Editor, open the When-Validate-Item trigger for the Orders.Sales_Rep_Id item. Copy the code.

In the Object Navigator, select the Program Units node and click Create. Enter the procedure name GET_SALES_REP_NAME and click OK. Paste the copied code into the program unit and click Compile. The code should be as follows:

```
PROCEDURE GET_SALES_REP_NAME IS
BEGIN
SELECT first_name || ' ' || last_name
  INTO :orders.sales_rep_name
  FROM employees
 WHERE employee_id = :orders.sales_rep_id
   AND job_id = 'SA_REP';
 EXCEPTION
  WHEN NO_DATA_FOUND THEN
     MESSAGE('Invalid sales rep id');
     RAISE form_trigger_failure;
END;
```

In the When-Validate-Item trigger for Orders.Customer_Id item, replace the existing code with:

```
GET_CUSTOMER_NAME;
```

In the When-Validate-Item trigger for Orders.Sales_Rep_Id item, replace the existing code with:

```
GET_SALES_REP_NAME;
```

In the Post-Query trigger of the Orders block, replace the existing code with:

```
GET_CUSTOMER_NAME;
GET_SALES_REP_NAME;
```

Run the form to test that the Customer Name and Sales Rep Id items are populated correctly when the form first displays and when you make changes to the Customer Id or the Sales Rep Id values.

Practice 24

1. In the ORDG*XX* form, integrate the WebUtil objects and library.
   Open the `Ordgxx.fmb` form. Also open the `webutil.olb` object library from the `e:\webutil_102\forms` directory. See the previous lesson for the steps to attach a PL/SQL library to a form.
   In the Object Navigator, double-click the icon for the WEBUTIL object library to open it. Select the WEBUTIL object group from the object library and drag it to the Object Groups node of the Orders form.
   Choose to Subclass, rather than Copy, the objects.
   In the Object Navigator, drag the WEBUTIL block to the last position under the Blocks node.
   Attach the WebUtil library (`.pll` file) from the `d:\webutil_102\forms` directory, choosing to remove the hard-coded path.
2. Save the form.
   No formal solution.
3. Change the Exit button in the Control block. Rename it: New_Image_Btn. Relabel it: New Image. Delete the current code for the button and write code to enable the user to choose a new JPEG image to display in the Product_Image item.
   Hint: You will need to use `CLIENT_GET_FILENAME` and `CLIENT_IMAGE.READ_IMAGE_FILE`. You may import the code from `pr24_3.txt`.
   Open the Property Palette for the Control.Exit_Button button. Change its name to `NEW_IMAGE_BTN` and its Label to New Image.
   Open the When-Button-Pressed trigger for the button and replace its code with the following code imported from `pr24_3.txt`:

```
DECLARE
   v_file VARCHAR2(250):=
   client_get_file_name('','','JPEG Files|*.jpg|',
      'Select a product image',open_file,TRUE);
   it_image_id ITEM :=
   FIND_ITEM('control.product_image');
BEGIN
   client_image.read_image_file(v_file,'',it_image_id);
END;
```

4. Set the Forms Builder run-time preferences to use a WebUtil configuration that has been set up for you, `?config=webutil`, then run the form to test it. Try to load one of the `.jpg` images in the lab directory.
   Note: Because the image item is not a base table item, the new image is not saved when the form is exited.
   In Forms Builder, select Edit > Preferences. Click the Runtime tab. Click Reset to Default. Append to the Application Server URL: `?config=webutil`

Click OK. Click Run.
The first time you run a form with WebUtil configured, you must respond to the JInitiator Security Warning. Choose "Grant this session" if you want to see the warning again the next time you run a form, or choose "Grant always" to eliminate the warning the next time.
(If you receive an error about the Forms session being aborted, be sure to drag the WEBUTIL block to the last position under the Blocks node in the Object Navigator and then run the form again.)
When running the form, click New Image. In the File dialog, navigate to the lab directory and select one of the `.jpg` files in that directory. Click Open. The new image should be displayed. (If not, you may have chosen the same image that was displayed previously, so click New Image again and choose a different image.)

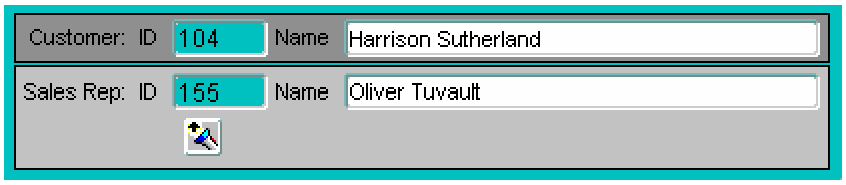
4. If you have time, experiment with some of the other client-server parity APIs by adding additional code to the New_Image_Btn button. For example, you could:
   a. Display a message on the status line that contains the value of the `ORACLE_HOME` environment variable. You can import the code from `pr24_4a.txt`.
   To the end of the trigger, add the code:
   ```
     DECLARE
       v_message VARCHAR2(30);
     BEGIN
       CLIENT_TOOL_ENV.GETVAR('ORACLE_HOME',v_message);
       MESSAGE('ORACLE_HOME='||v_message);
     END;
   ```
   b. Create a file called `hello.txt` in the `c:\temp` directory that contains the text “Hello World” and invoke Notepad to display this file. You can import the code from `pr24_4b.txt`.
   ```
    DECLARE
     v_dir VARCHAR2(250) := 'c:\temp';
     ft_tempfile CLIENT_TEXT_IO.FILE_TYPE;
   begin
     ft_tempfile := CLIENT_TEXT_IO.FOPEN(v_dir ||
       '\hello.txt','w');
     CLIENT_TEXT_IO.PUT_LINE(ft_tempfile,'Hello
       World');
     CLIENT_TEXT_IO.FCLOSE(ft_tempfile);
    CLIENT_HOST('cmd /c notepad c:\temp\hello.txt');
   END;
   ```

   If desired, you can reset the preferences to delete the configuration parameter: Select Edit > Preferences. Click the Runtime tab. Click Reset to Default. However, this step is not necessary, because the `webutil` configuration parameter will not affect the remaining practices.

Practice 25 Solutions

1. In the ORDG*XX* form, create a Pre-Form trigger to ensure that a global variable called Customer_Id exists.
   Pre-Form at form level:
   ```
   DEFAULT_VALUE('','GLOBAL.customer_id');
   ```
2. Add a trigger to ensure that queries on the ORDERS block are restricted by the value of GLOBAL.Customer_Id
   Pre-Query on ORDERS block:
   ```
   :ORDERS.customer_id := :GLOBAL.customer_id;
   ```
3. Save, compile, and run the form to test that it works as a stand-alone.
   No formal solution.
4. In the CUSTG*XX* form, create a CONTROL block button called Orders_Button and set appropriate properties. Place in on the `CV_CUSTOMER` canvas below the Customer_Id item.
   Create a button.
   Set the Name to ORDERS_BUTTON.
   Set Label to Orders.
   Set Keyboard Navigable and Mouse Navigate to No.
   Set Height to 16.
5. Define a trigger for CONTROL.Orders_Button that initializes GLOBAL.Customer_Id with the current customer's ID, and then opens the ORDG*XX* form, passing control to it.
   When-Button-Pressed on CONTROL.Orders_Button:
   ```
   :GLOBAL.customer_id := :CUSTOMERS.customer_id;
   OPEN_FORM('ordgxx');
   ```
6. Save and compile each form. Run the CUSTG*XX* form and test the button to open the Orders form.
   No formal solution.
7. Change the window location of the ORDG*XX* form, if required.
   No formal solution.

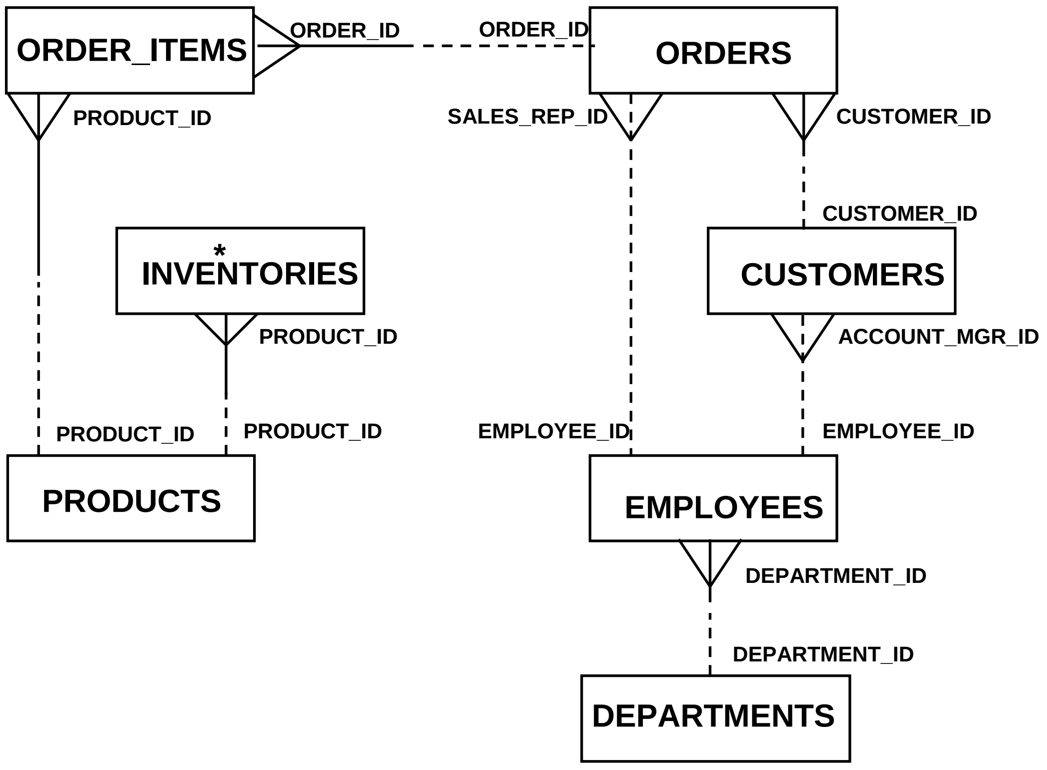
Practice 25 Solutions (continued)

8. Alter the Orders_Button trigger in CUSTG*XX* so that it does not open more than one instance of the Orders form, but uses GO_FORM to pass control to ORDG*XX* if the form is already running. Use the FIND_FORM built-in for this purpose.
   When-Button-Pressed on Orders_Button:
   ```
   :GLOBAL.customer_id := :CUSTOMERS.customer_id;
   IF ID_NULL(FIND_FORM('ORDERS'))THEN
             OPEN_FORM('ORDGXX');
        ELSE
             GO_FORM('ORDERS');
        END IF;
   ```
   Remember that you need to use the module name in the `FIND_FORM` and `GO_FORM` built-ins, and the filename in the `OPEN_FORM` built-in.
9. If you navigate to a second customer record and click Orders, the Orders form still displays the records for the previous customer. Write a trigger to reexecute the query in the ORDERS form in this situation.
   When-Form-Navigate trigger on ORDERS form:
   ```
   IF :GLOBAL.customer_id IS NOT NULL
   AND :ORDERS.customer_id != :GLOBAL.customer_id THEN
        EXECUTE_QUERY;
   END IF;
   ```
10. Write a When-Create-Record trigger on the ORDERS block that uses the value of GLOBAL.Customer_Id as the default value for ORDERS.Customer_Id.
    When-Create-Record on ORDERS block:
    ```
    :ORDERS.customer_id := :GLOBAL.customer_id;
    ```
11. Add code to the CUSTG*XX* form so that GLOBAL.Customer_Id is updated when the current Customer_Id changes.
    When-Validate-Item on CUSTOMERS.Customer_Id:
    ```
    :GLOBAL.customer_id := :CUSTOMERS.customer_id;
    ```
12. Save and compile the ORDG*XX* form. Save, compile, and run the CUSTG*XX* form to test the functionality.
    No formal solution.
13. **If you have time**, you can modify the appearance of the ORD*XX* form to make it easier to read, similar to what you see in `ORDERS.fmb`.
    
    Customer: ID 104 Name Harrison Sutherland
    Sales Rep: ID 155 Name Oliver Tuvault

    Modify prompts and add boilerplate text and rectangles. With the rectangles selected, choose Layout > Send to Back.

B Table Descriptions

Summit Office Supply Database Diagram



*Unique occurrences are identified by `PRODUCT_ID` and `WAREHOUSE_ID`.

Note: What follows is not a complete list of schema objects, but only those relevant to the Summit Office Supply application.

CUSTOMERS Description

```
SQL> desc customers
 Name                Null?    Type
 ------------------ -------- -------------
 CUSTOMER_ID         NOT NULL NUMBER(6)
 CUST_FIRST_NAME     NOT NULL VARCHAR2(20)
 CUST_LAST_NAME      NOT NULL VARCHAR2(20)
 CUST_ADDRESS                 CUST_ADDRESS_TYP
 PHONE_NUMBERS                VARCHAR2(30)
 NLS_LANGUAGE                 VARCHAR2(3)
 NLS_TERRITORY                VARCHAR2(30)
 CREDIT_LIMIT                 NUMBER(9,2)
 CUST_EMAIL                   VARCHAR2(30)
 ACCOUNT_MGR_ID               NUMBER(6)

SQL> desc cust_address_typ
 Name                Null?    Type
 ------------------ -------- ------------
 STREET_ADDRESS               VARCHAR2(40)
 POSTAL_CODE                  VARCHAR2(10)
 CITY                         VARCHAR2(30)
 STATE_PROVINCE               VARCHAR2(10)
 COUNTRY_ID                   CHAR(2)
```

Related object creation statements

```
CREATE TYPE cust_address_typ
  AS OBJECT
    ( street_address     VARCHAR2(40)
    , postal_code        VARCHAR2(10)
    , city               VARCHAR2(30)
    , state_province     VARCHAR2(10)
    , country_id         CHAR(2)
    );

CREATE TABLE customers
    ( customer_id        NUMBER(6)
    , cust_first_name    VARCHAR2(20) CONSTRAINT
cust_fname_nn NOT NULL
    , cust_last_name     VARCHAR2(20) CONSTRAINT
cust_lname_nn NOT NULL
    , cust_address       cust_address_typ
    , phone_numbers      varchar2(30)
    , nls_language       VARCHAR2(3)
```

CUSTOMERS Description (continued)

```
    , nls_territory      VARCHAR2(30)
    , credit_limit       NUMBER(9,2)
    , cust_email         VARCHAR2(30)
    , account_mgr_id     NUMBER(6)
    , CONSTRAINT         customer_credit_limit_max
                         CHECK (credit_limit <= 5000)
    , CONSTRAINT         customer_id_min
                         CHECK (customer_id > 0)) ;

CREATE UNIQUE INDEX customers_pk
   ON customers (customer_id) ;

ALTER TABLE customers
ADD ( CONSTRAINT customers_pk
      PRIMARY KEY (customer_id)) ;

ALTER TABLE customers
ADD ( CONSTRAINT customers_account_manager_fk
    FOREIGN KEY (account_mgr_id)
                 REFERENCES employees(employee_id)
      ON DELETE SET NULL) ;

CREATE SEQUENCE customers_seq
START WITH 982
INCREMENT BY 1
NOCACHE
NOCYCLE;
```

Sample record (1 out of 319 customers):

```
CUSTOMER_ID CUST_FIRST_NAME      CUST_LAST_NAME
----------- -------------------- --------------------
CUST_ADDRESS(STREET_ADDRESS, POSTAL_CODE, CITY, STATE_PROVINCE,
   COUNTRY_ID)
----------------------------------------------------------------
   ----------------
PHONE_NUMBERS                 NLS NLS_TERRITORY
   CREDIT_LIMIT
----------------------------- --- -----------------------------
   - ------------
CUST_EMAIL                    ACCOUNT_MGR_ID
----------------------------- --------------
        101 Constantin           Welles
CUST_ADDRESS_TYP('514 W Superior St', '46901', 'Kokomo', 'IN',
   'US')
+1 317 123 4104               us  AMERICA
   100
Constantin.Welles@ANHINGA.COM
```

DEPARTMENTS Description

```
 SQL> desc departments
Name                Null?    Type
------------------- -------- ------------
DEPARTMENT_ID       NOT NULL NUMBER(4)
DEPARTMENT_NAME     NOT NULL VARCHAR2(30)
MANAGER_ID                   NUMBER(6)
LOCATION_ID                  NUMBER(4)
```

Related object creation statements

```
CREATE TABLE departments
    ( department_id    NUMBER(4)
    , department_name  VARCHAR2(30)
    CONSTRAINT  dept_name_nn  NOT NULL
    , manager_id       NUMBER(6)
    , location_id      NUMBER(4)
    ) ;

CREATE UNIQUE INDEX dept_id_pk
ON departments (department_id) ;

ALTER TABLE departments
ADD ( CONSTRAINT dept_id_pk
                PRIMARY KEY (department_id)
    , CONSTRAINT dept_loc_fk
                FOREIGN KEY (location_id)
            REFERENCES locations (location_id)
     ) ;
```

Note: The Locations table is not documented here because it is not used in the Summit Office Supply application.

```
Rem      Useful for any subsequent addition of rows to
         departments table
Rem      Starts with 280


CREATE SEQUENCE departments_seq
 START WITH     280
 INCREMENT BY   10
 MAXVALUE       9990
 NOCACHE
 NOCYCLE;
```

DEPARTMENTS Description (continued)

```
SQL> select * from departments;
DEPARTMENT_ID DEPARTMENT_NAME   MANAGER_ID LOCATION_ID
------------- ---------------- ---------- -----------
           10 Administration          200        1700
           20 Marketing               201        1800
           30 Purchasing              114        1700
           40 Human Resources         203        2400
           50 Shipping                121        1500
           60 IT                      103        1400
           70 Public Relations        204        2700
           80 Sales                   145        2500
           90 Executive               100        1700
          100 Finance                 108        1700
          110 Accounting              205        1700
          120 Treasury                           1700
          130 Corporate Tax                      1700
          140 Control And Credit                 1700
          150 Shareholder Services               1700
          160 Benefits                           1700
          170 Manufacturing                      1700
          180 Construction                       1700
          190 Contracting                        1700
          200 Operations                         1700
          210 IT Support                         1700
          220 NOC                                1700
          230 IT Helpdesk                        1700
          240 Government Sales                   1700
          250 Retail Sales                       1700
          260 Recruiting                         1700
          270 Payroll                            1700
27 rows selected.
```

EMPLOYEES Description

```
SQL> desc employees
 Name                Null?    Type
 ----------------- -------- -----------
 EMPLOYEE_ID         NOT NULL NUMBER(6)
 FIRST_NAME          VARCHAR2(20)
 LAST_NAME           NOT NULL VARCHAR2(25)
 EMAIL               NOT NULL VARCHAR2(25)
 PHONE_NUMBER                 VARCHAR2(20)
 HIRE_DATE           NOT NULL DATE
 JOB_ID              NOT NULL VARCHAR2(10)
 SALARY                       NUMBER(8,2)
 COMMISSION_PCT               NUMBER(2,2)
 MANAGER_ID                   NUMBER(6)
 DEPARTMENT_ID                NUMBER(4)
```

Related object creation statements

```
CREATE TABLE employees
    ( employee_id    NUMBER(6)
    , first_name     VARCHAR2(20)
    , last_name      VARCHAR2(25)
     CONSTRAINT     emp_last_name_nn  NOT NULL
    , email          VARCHAR2(25)
    CONSTRAINT     emp_email_nn  NOT NULL
    , phone_number   VARCHAR2(20)
    , hire_date      DATE
    CONSTRAINT     emp_hire_date_nn  NOT NULL
    , job_id         VARCHAR2(10)
    CONSTRAINT     emp_job_nn  NOT NULL
    , salary         NUMBER(8,2)
    , commission_pct NUMBER(2,2)
    , manager_id     NUMBER(6)
    , department_id  NUMBER(4)
    , CONSTRAINT     emp_salary_min
                     CHECK (salary > 0)
    , CONSTRAINT     emp_email_uk
                     UNIQUE (email)
    ) ;
```

EMPLOYEES Description (continued)

```
CREATE UNIQUE INDEX emp_emp_id_pk
ON employees (employee_id) ;
ALTER TABLE employees
ADD ( CONSTRAINT     emp_emp_id_pk
                     PRIMARY KEY (employee_id)
    , CONSTRAINT     emp_dept_fk
                     FOREIGN KEY (department_id)
                      REFERENCES departments
    , CONSTRAINT     emp_job_fk
                     FOREIGN KEY (job_id)
                      REFERENCES jobs (job_id)
    , CONSTRAINT     emp_manager_fk
                     FOREIGN KEY (manager_id)
                      REFERENCES employees
    ) ;
ALTER TABLE departments
ADD ( CONSTRAINT dept_mgr_fk
                FOREIGN KEY (manager_id)
                 REFERENCES employees (employee_id)
    ) ;
Rem Useful for any subsequent addition of rows to
    employees table
Rem Starts with 207
CREATE SEQUENCE employees_seq
 START WITH     207
 INCREMENT BY   1
 NOCACHE
 NOCYCLE;
```

Sample records (2 out of 107 employees):

```
EMPLOYEE_ID FIRST_NAME LAST_NAME EMAIL    PHONE_NUMBER HIRE_DATE
----------- ---------- --------- -------- ------------ ----------
  JOB_ID   SALARY COMMISSION_PCT MANAGER_ID DEPARTMENT_ID
  -------- ------ -------------- ---------- -------------
100         Steven     King      SKING    515.123.4567 17-JUN-87
 AD_PRES          24000                            90
101         Neena      Kochhar   NKOCHHAR 515.123.4568 21-SEP-89
 AD_VP       17000                   100       90
```

INVENTORIES Description and Data

```
SQL> desc inventories
 Name                                      Null?    Type
 ----------------------------------------- -------- ---------
 PRODUCT_ID                                NOT NULL NUMBER(6)
 WAREHOUSE_ID                              NOT NULL NUMBER(3)
 QUANTITY_ON_HAND                          NOT NULL NUMBER(8)
```

Related object creation statements:

```
CREATE TABLE inventories
  ( product_id         NUMBER(6)
  , warehouse_id       NUMBER(3) CONSTRAINT
inventory_warehouse_id_nn NOT NULL
  , quantity_on_hand   NUMBER(8)
CONSTRAINT inventory_qoh_nn NOT NULL
  , CONSTRAINT inventory_pk PRIMARY KEY (product_id,
warehouse_id)
  ) ;

ALTER TABLE inventories
ADD ( CONSTRAINT inventories_warehouses_fk
      FOREIGN KEY (warehouse_id)
      REFERENCES warehouses (warehouse_id)
      ENABLE NOVALIDATE
    ) ;

ALTER TABLE inventories
ADD ( CONSTRAINT inventories_product_id_fk
      FOREIGN KEY (product_id)
      REFERENCES product_information (product_id)
    ) ;
```

Sample records (11 out of 1112 inventory items):

```
PRODUCT_ID WAREHOUSE_ID QUANTITY_ON_HAND
---------- ------------ ----------------
      1733            1              106
      1734            1              106
      1737            1              106
      1738            1              107
      1745            1              108
      1748            1              108
      2278            1              125
      2316            1              131
      2319            1              117
      2322            1              118
      2323            1              118
```

PRODUCTS Description and Data

```
SQL> desc products
 Name                       Null?    Type
 ---------------------- -------- ---------------
 PRODUCT_ID             NOT NULL NUMBER(6)
 LANGUAGE_ID                     VARCHAR2(3)
 PRODUCT_NAME                    NVARCHAR2(250)
 CATEGORY_ID                     NUMBER(2)
 PRODUCT_DESCRIPTION             NVARCHAR2(4000)
 WEIGHT_CLASS                    NUMBER(1)
 WARRANTY_PERIOD                 NUMBER(5)
 SUPPLIER_ID                     NUMBER(6)
 PRODUCT_STATUS                  VARCHAR2(20)
 LIST_PRICE                      NUMBER(8,2)
 MIN_PRICE                       NUMBER(8,2)
 CATALOG_URL                     VARCHAR2(50)
```

Related object creation statements:

```
CREATE OR REPLACE VIEW products
AS
SELECT i.product_id
,      d.language_id
,      CASE WHEN d.language_id IS NOT NULL
            THEN d.translated_name
            ELSE TRANSLATE(i.product_name USING NCHAR_CS)
       END    AS product_name
,      i.category_id
,      CASE WHEN d.language_id IS NOT NULL
            THEN d.translated_description
            ELSE TRANSLATE(i.product_description USING NCHAR_CS)
       END    AS product_description
,      i.weight_class
,      i.warranty_period
,      i.supplier_id
,      i.product_status
,      i.list_price
,      i.min_price
,      i.catalog_url
FROM   product_information  i
,      product_descriptions d
WHERE  d.product_id  (+) = i.product_id
AND    d.language_id (+) = sys_context('USERENV','LANG');
```

PRODUCTS Description and Data (continued)

```
CREATE TABLE product_information
    ( product_id          NUMBER(6)
    , product_name        VARCHAR2(50)
    , product_description VARCHAR2(2000)
    , category_id         NUMBER(2)
    , weight_class        NUMBER(1)
    , warranty_period     NUMBER(5)
    , supplier_id         NUMBER(6)
    , product_status      VARCHAR2(20)
    , list_price          NUMBER(8,2)
    , min_price           NUMBER(8,2)
    , catalog_url         VARCHAR2(50)
    , CONSTRAINT          product_status_lov
                          CHECK (product_status in ('orderable',
                         'planned',development','obsolete'))) ;

ALTER TABLE product_information
ADD ( CONSTRAINT product_information_pk PRIMARY KEY
   (product_id));

CREATE TABLE product_descriptions
    ( product_id             NUMBER(6)
    , language_id            VARCHAR2(3)
    , translated_name        NVARCHAR2(50)
CONSTRAINT translated_name_nn NOT NULL
    , translated_description NVARCHAR2(2000)
CONSTRAINT translated_desc_nn NOT NULL);

CREATE UNIQUE INDEX prd_desc_pk
ON product_descriptions(product_id,language_id) ;

ALTER TABLE product_descriptions
ADD ( CONSTRAINT product_descriptions_pk
   PRIMARY KEY (product_id, language_id));
```

Sample records (4 out of 288 products; only 3 columns shown):

```
PRODUCT_ID PRODUCT_NAME                   LIST_PRICE
---------- ------------------------------ ----------
      1726 LCD Monitor 11/PM                     259
      2359 LCD Monitor 9/PM                      249
      3060 Monitor 17/HR                         299
      2243 Monitor 17/HR/F                       350
```

ORDER_ITEMS Description

```
SQL> desc order_items;
 Name                    Null?    Type
 -------------------- -------- -----------
 ORDER_ID             NOT NULL NUMBER(12)
 LINE_ITEM_ID         NOT NULL NUMBER(3)
 PRODUCT_ID           NOT NULL NUMBER(6)
 UNIT_PRICE                    NUMBER(8,2)
 QUANTITY                      NUMBER(8)
```

Related object creation statements:

```
CREATE TABLE order_items
    ( order_id           NUMBER(12)
    , line_item_id       NUMBER(3)  NOT NULL
    , product_id         NUMBER(6)  NOT NULL
    , unit_price         NUMBER(8,2)
    , quantity           NUMBER(8)
    ) ;

CREATE UNIQUE INDEX order_items_pk
ON order_items (order_id, line_item_id) ;

CREATE UNIQUE INDEX order_items_uk
ON order_items (order_id, product_id) ;

ALTER TABLE order_items
ADD ( CONSTRAINT order_items_pk PRIMARY KEY (order_id,
   line_item_id)
    );

ALTER TABLE order_items
ADD ( CONSTRAINT order_items_order_id_fk
      FOREIGN KEY (order_id)
      REFERENCES orders(order_id)
      ON DELETE CASCADE enable novalidate) ;

ALTER TABLE order_items
ADD ( CONSTRAINT order_items_product_id_fk
      FOREIGN KEY (product_id)
      REFERENCES product_information(product_id)) ;
```

ORDER_ITEMS Description (continued)

```
CREATE OR REPLACE TRIGGER insert_ord_line
  BEFORE INSERT ON order_items
  FOR EACH ROW
  DECLARE
    new_line number;
  BEGIN
    SELECT (NVL(MAX(line_item_id),0)+1) INTO new_line
      FROM order_items
      WHERE order_id = :new.order_id;
    :new.line_item_id := new_line;
  END;
```

Sample records (11 out of 665 order items):

```
 ORDER_ID LINE_ITEM_ID PRODUCT_ID UNIT_PRICE   QUANTITY
---------- ------------ ---------- ---------- ----------
      2355            1       2289         46        200
      2356            1       2264      199.1         38
      2357            1       2211        3.3        140
      2358            1       1781      226.6          9
      2359            1       2337      270.6          1
      2360            1       2058         23         29
      2361            1       2289         46        180
      2362            1       2289         48        200
      2363            1       2264      199.1          9
      2364            1       1910         14          6
      2365            1       2289         48         92
```

ORDERS Description and Data

```
SQL> desc orders
 Name                   Null?    Type
 ------------------ -------- -----------
 ORDER_ID           NOT NULL NUMBER(12)
 ORDER_DATE         NOT NULL DATE
 ORDER_MODE                  VARCHAR2(8)
 CUSTOMER_ID        NOT NULL NUMBER(6)
 ORDER_STATUS                NUMBER(2)
 ORDER_TOTAL                 NUMBER(8,2)
 SALES_REP_ID                NUMBER(6)
 PROMOTION_ID                NUMBER(6)
```

Related object creation statements:

```
CREATE TABLE orders
    ( order_id          NUMBER(12)
    , order_date        DATE
                        CONSTRAINT order_date_nn NOT NULL
    , order_mode        VARCHAR2(8)
    , customer_id       NUMBER(6)
                        CONSTRAINT order_customer_id_nn NOT NULL
    , order_status      NUMBER(2)
    , order_total       NUMBER(8,2)
    , sales_rep_id      NUMBER(6)
    , promotion_id      NUMBER(6)
    , CONSTRAINT        order_mode_lov
                        CHECK (order_mode in
   ('direct','online'))
    , constraint        order_total_min
                        check (order_total >= 0)) ;

CREATE UNIQUE INDEX order_pk
ON orders (order_id) ;

ALTER TABLE orders
ADD ( CONSTRAINT order_pk
      PRIMARY KEY (order_id));

ALTER TABLE orders
ADD ( CONSTRAINT orders_sales_rep_fk
      FOREIGN KEY (sales_rep_id)
      REFERENCES employees(employee_id)
      ON DELETE SET NULL) ;
```

ORDERS Description and Data (continued)

```
ALTER TABLE orders
ADD ( CONSTRAINT orders_customer_id_fk
      FOREIGN KEY (customer_id)
      REFERENCES customers(customer_id)
      ON DELETE SET NULL) ;
```

Sample records (12 out of 105 orders):

```
 ORDER_ID ORDER_DAT ORDER_MO CUSTOMER_ID ORDER_STATUS ORDER_TOTAL
---------- --------- -------- ----------- ------------ ----------
SALES_REP_ID PROMOTION_ID
------------ ------------
      2458 16-AUG-99 direct           101            0    78279.6
         153
      2397 19-NOV-99 direct           102            1    42283.2
         154
      2454 02-OCT-99 direct           103            1     6653.4
         154
      2354 14-JUL-00 direct           104            0      46257
         155
      2358 08-JAN-00 direct           105            2       7826
         155
      2381 14-MAY-00 direct           106            3    23034.6
         156
      2440 31-AUG-99 direct           107            3    70576.9
         156
      2357 08-JAN-98 direct           108            5    59872.4
         158
      2394 10-FEB-00 direct           109            5      21863
         158
      2435 02-SEP-99 direct           144            6      62303
         159
      2455 20-SEP-99 direct           145            7    14087.5
         160
      2379 16-MAY-99 direct           146            8    17848.2
         161
```

C

Introduction to Query Builder

Query Builder Features

- Easy-to-use data access tool
- Point-and-click graphical user interface
- Distributed data access
- Powerful query building

Lesson Aim

This lesson teaches you more about using Query Builder, the tool used in the LOV wizard to build the query on which a record group is based.

What Is Query Builder?

Easy-to-Use Data Access Tool

Query Builder is an easy-to-use data access tool. It provides a logical and intuitive means to access information stored in networked, distributed databases for analysis and reporting.

Point-and-Click Graphical User Interface

Query Builder enables you to become productive quickly because its graphical user interface works like your other applications. A toolbar enables you to perform common operations quickly.

Distributed Data Access

Query Builder represents all database objects (tables, views, and so on) as graphical datasources, which look and work exactly the same way regardless of which database or account the data came from.

Lesson Aim (continued)

What Is Query Builder? (continued)

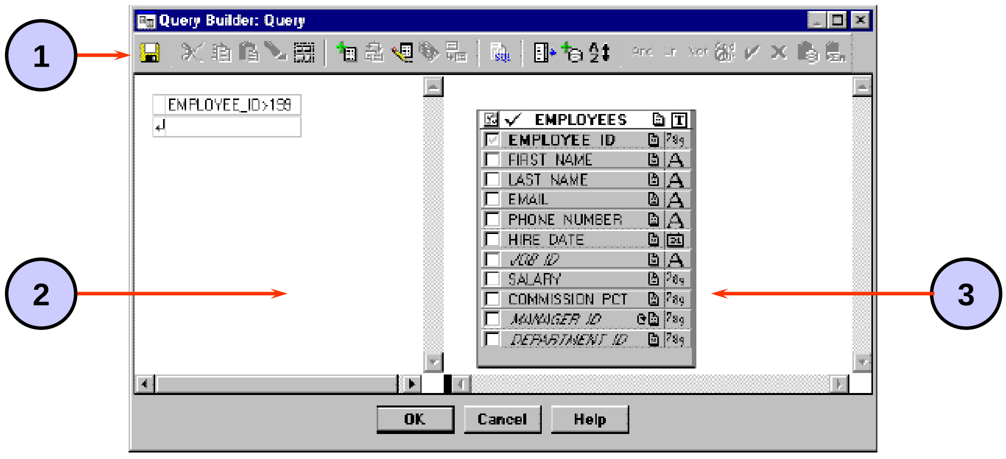
- Performing distributed queries on complex, enterprisewide databases is no more difficult than querying a single database.
- Locating database objects is easy because Query Builder uses a single hierarchical directory that lists all accessible data in your account, in other accounts, and in other databases.

Powerful Query Building

Query Builder is designed for professionals who do not have a computer programming or database background. However, because of its powerful query features and its support of Structured Query Language (SQL) statements, experienced database users and programmers will find that Query Builder serves many of their needs as well.

- Graphical representation of tables and their relationships enables you to see the structure of your data.
- You can build queries by clicking the columns that you want to retrieve from the database. The browser generates the necessary SQL statements behind the scenes.
- You can specify exactly which rows to retrieve from the database by using conditions, which consist of any valid SQL expression that evaluates to true or false.
- You can combine and nest conditions graphically using logical operators. You can disable conditions temporarily for *what-if* analysis.

Query Builder Window



EMPLOYEE_ID>199

EMPLOYEES
EMPLOYEE_ID
FIRST_NAME
LAST_NAME
EMAIL
PHONE_NUMBER
HIRE_DATE
JOB_ID
SALARY
COMMISSION_PCT
MANAGER_ID
DEPARTMENT_ID

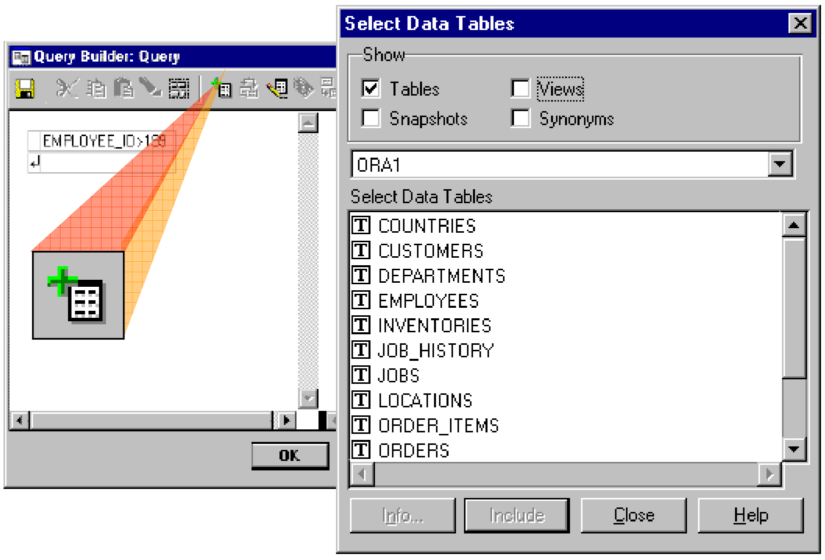
The Query Builder Window

The Query Builder graphical user interface consists of one window, the Query window, where you build your queries.

The Query Builder window is comprised of:

1. The Toolbar that enables you to issue commands with a click of the mouse. You can create conditions, add new datasources, or define new columns.
2. The Conditions panel where you specify conditions to refine your queries
3. The Datasource panel where you display and select tables and columns for a query

Building a New Query



Building a New Query

To build a query, you must select the tables you want to include and the columns you want to retrieve.

How to Include a Table

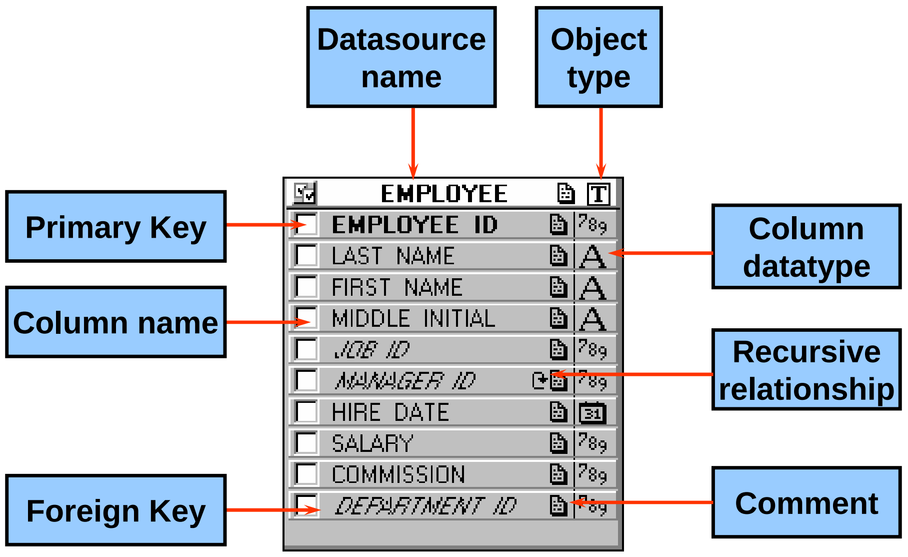
1. Choose Select Data Tables from the toolbar.
   The Select Data Tables dialog box appears.
   Note: When you open Query Builder to create a new query, the Select Data Tables dialog box is open by default.
2. Select the table name and then click Include, or simply double-click the desired table name.
   The selected table appears in the Query window.
3. Click Close to close the dialog box.

You can at any time include additional tables in the query by following these steps.

How to Delete a Table

1. Select the data source in the Query window.
2. Select Clear from the toolbar or press [Delete].

Datasource Components



Datasource Components

Datasource name

An object that has been included in a query is referred to as a *datasource*. It is displayed as a rectangular graphic in the Datasource panel of the Query window. The top part of the rectangle contains the table name and an icon representing the **object type:**

Object Type	Description
Table	Stores data in the database
View	Acts like a table when you execute a query, but is really a pointer to either a subset of a table, a combination of tables, or a join of two or more tables.
Synonym	Another name for an object. Sometimes table names can be rather cryptic, such as emp_em_con_tbl. You can create a synonym that simply calls this table Contracts.
Alias	Query Builder uses an alias name for a table when the table is used more than once in a query, mostly with self-joins. You can also rename a table.

Datasource Components (continued)

Columns

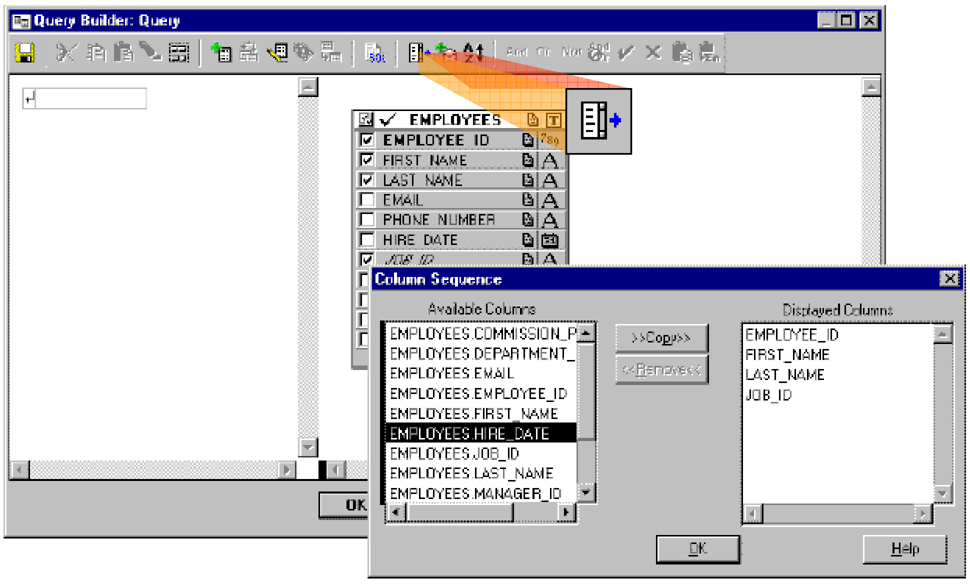
The body of the rectangle contains column names listed vertically. To the right of the column name is an icon representing the datatype.
Columns also provide additional information.

- **Primary keys** are displayed in bold.
- **Foreign keys** are displayed in italics.
- **Recursive relationships** are indicated by a self-relationship icon.

Comments

If the table or column is commented, an icon appears to indicate the presence of a comment. You can double-click the icon to read the comment.

Refining a Query



Refining a Query

Adding Columns to a Query

You can add a column to a query either by selecting the check box to the left of the column name, or by double-clicking the column name. To include all columns from any single table, double-click the table heading.

Removing Columns from a Query

You can remove columns from a query either by clearing the check box to the right of the column name, or by double-clicking the column name. To remove all columns from any single table, double-click the table heading.

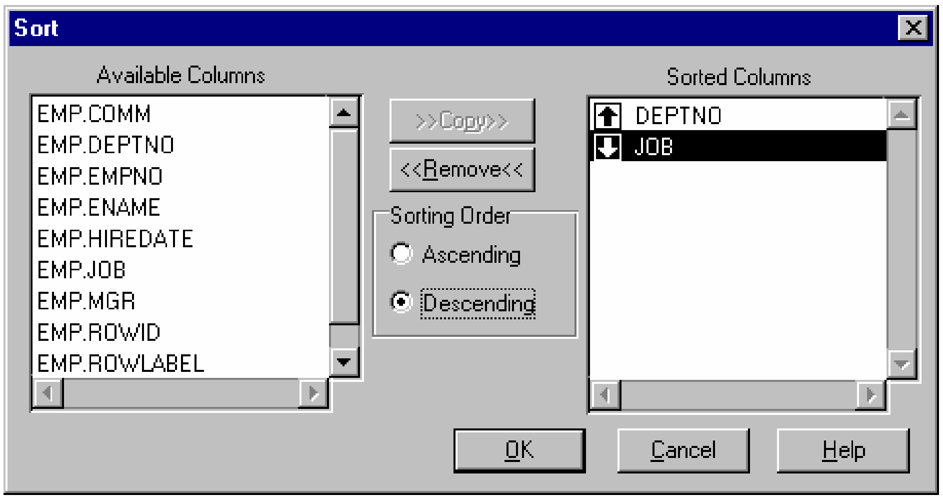
Changing Column Position in a Query

By default, Query Builder places columns in the order in which you select them. You can resequence them by selecting the Column Sequence tool.

The Column Sequence dialog box appears. Column names are shown in the Displayed Columns list in order of their appearance in the query. Drag any column to a new position.

Note: You can also use the Column Sequence dialog box to add columns to or remove them from a query.

Sorting Data


Sort

Available Columns

EMP.COMM
EMP.DEPTNO
EMP.EMPNO
EMP.ENAME
EMP.HIREDATE
EMP.JOB
EMP.MGR
EMP.ROWID
EMP.ROWLABEL

Sorted Columns

DEPTNO
JOB

Sorting Order
Ascending
[✓ Descending]

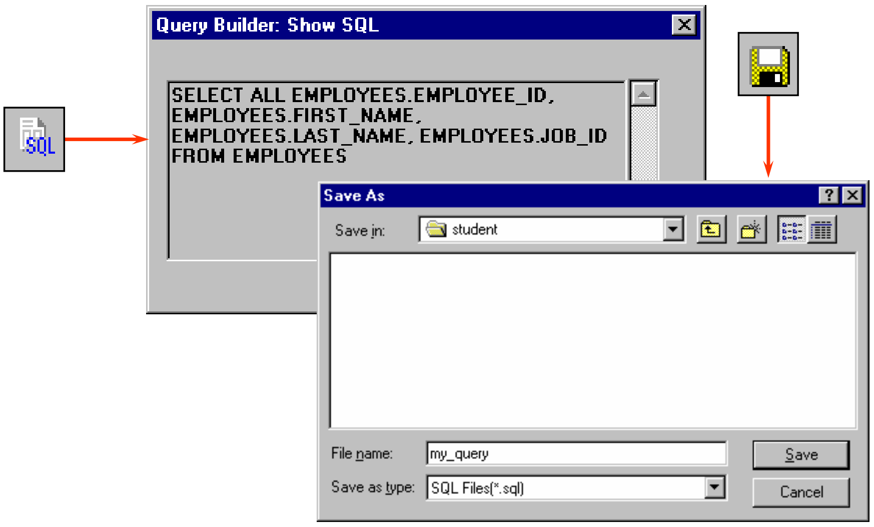
Sorting Data

By default, a query returns the data in no specific order. To sort the data, you must add an `ORDER BY` clause to the query.

How to Add an `ORDER BY` Clause to a Query

1. Select the Sort tool. The Sort dialog box appears.
2. Select the column you want to sort from the Available Columns list.
3. Select Copy.
   Query Builder adds the column to the Sorted Columns list and places an up arrow in the list box to indicate the default sort ascending order.
   Note: You can sort by more than one column. Query Builder sorts according to the order in which columns appear in the Sorted Columns list.
4. You can change the sorting order by selecting the column name in the Sorted Columns list and selecting the desired option button.
5. Select OK to close the dialog box.

Viewing and Saving Queries



```
SELECT ALL EMPLOYEES.EMPLOYEE_ID,
EMPLOYEES.FIRST_NAME,
EMPLOYEES.LAST_NAME, EMPLOYEES.JOB_ID
FROM EMPLOYEES
```

Viewing and Saving Queries

Viewing a Query

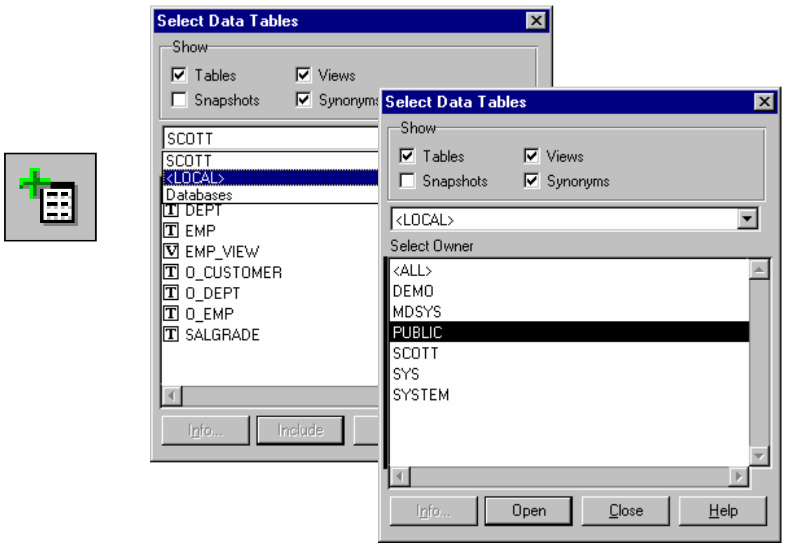
Select the Show SQL tool to view the query text that Query Builder will create.

How to Save a Query

You can save your query as a SQL statement to the file system.

1. Select the Save tool from the toolbar.
   The Save As dialog box appears.
2. Enter a filename.
   Note: If you do not enter a file extension, Query Builder automatically appends the `.SQL` file extension.
3. Select a destination.
4. Click OK.

Including Additional Tables



DEPT
EMP
EMP_VIEW
O_CUSTOMER
O_DEPT
O_EMP
SALGRADE

<ALL>
DEMO
MDSYS
PUBLIC
SCOTT
SYS
SYSTEM

Including Additional Tables

Often, all of the data needed to create a desired report cannot be found in a single table. With Query Builder, you can include multiple tables in a single query.

Remember that you can see the names of all the tables in your schema by choosing the Select Data Tables tool. This brings up the Select Data Tables dialog box. You can also use this dialog box to show only certain types of datasources as well as datasources in other schemas and databases.

How to Find Tables in Other Schemas

If you do not find the table you are looking for in the Select Data Tables dialog box in your own schema, you can search other schemas.

1. Open the pop-up menu to display the name of the current database and the option databases.
2. Select the current database name.
   The list box displays a list of schemas that contain tables you can access.
3. Select the name of the schema in the list and then click Open, or simply double-click the schema name. This opens the schema and displays a list of objects in the schema.
4. Follow the normal procedure for including objects in the Query window.

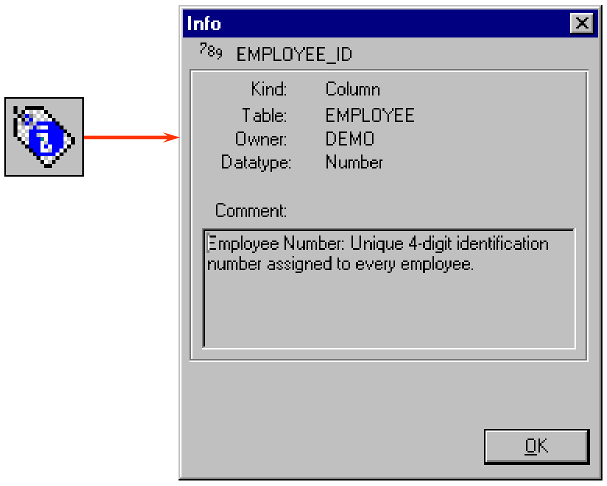
Including Additional Tables (continued)

How to Find Tables in Other Databases

If you do not find the table you are looking for in the Select Data Tables dialog box in your own schema, you can search other databases.

1. Open the pop-up menu to display the name of the current database and the option databases.
2. Select databases.
   The list box displays a list of databases you can access.
3. Select the name of the database in the list and then click Open, or simply double-click the account name.
4. Follow the normal procedure for including tables in the Query window.

Viewing Comments



Info

789 EMPLOYEE_ID

Kind:	Column
Table:	EMPLOYEE
Owner:	DEMO
Datatype:	Number

Comment:

Employee Number: Unique 4-digit identification number assigned to every employee.

How to View Comments

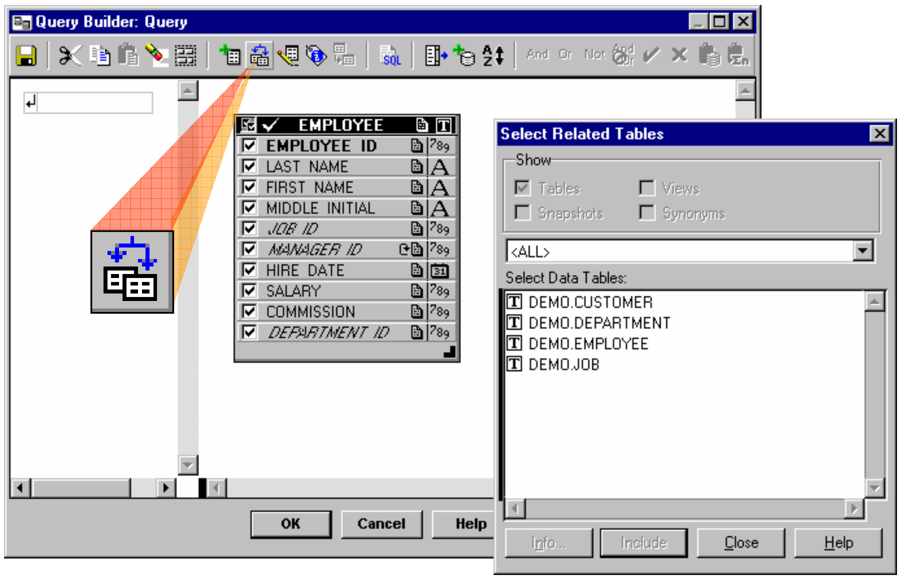
Sometimes, details about the kind of data stored in the database are not reflected by the table or column names. This can often make it difficult to decide which objects to include in your query. Query Builder features the Get Info dialog box to provide this kind of information for tables and columns.

To open the dialog box, follow these steps:

1. In the Datasource panel, select the table or column name.
2. Select the Get Info tool.

Alternatively, you can double-click the Comment icon in each table or column in the Datasources panel.

Including Related Tables



Relationships

To combine data from multiple tables into one query, Query Builder enables you to search for relationships between tables and to create user-defined relationships if they do not exist. Additionally, you can activate or deactivate relationships to suit your needs.

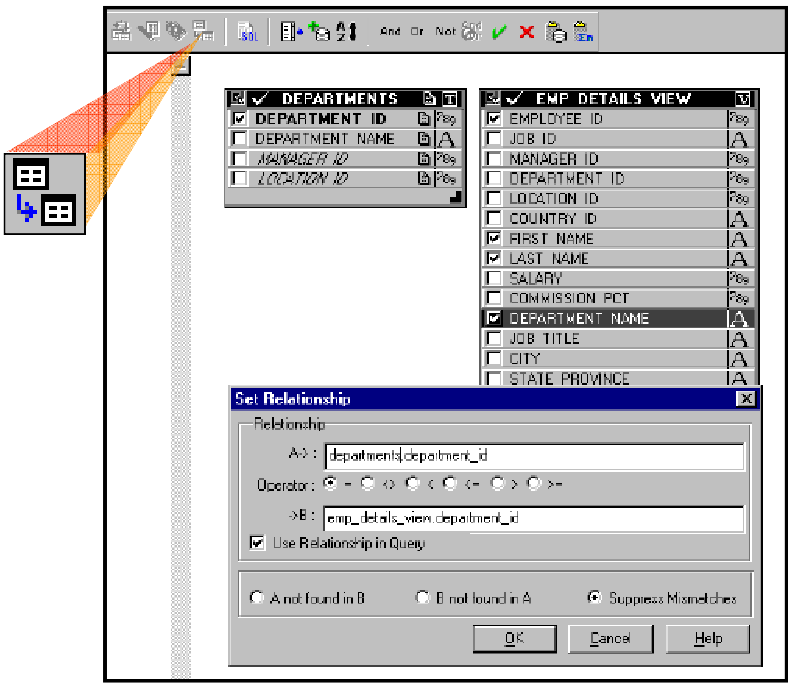
How to Find and Include Related Tables

1. Select the table in the Datasource panel.
2. Choose the Select Related Tables tool.
   A list of tables that have relationships defined with the selected table appears.
3. Select the table name and click Include, or simply double-click the table name.
   The selected table appears in the Datasource panel. Relationships between the tables are identified by relationship lines, drawn from the primary keys in one table to the foreign keys in another.
4. Click Close to close the dialog box.

Once you have included the table, you can retrieve its columns.

Note: When you select a foreign key column before selecting related tables, only the table to which the foreign key refers appears in the Select Related Tables dialog box.

Creating a User-Defined Relationship



Relationship

A-> : departments.department_id

Operator : [✓ =] <> < <= > >=

->B : emp_details_view.department_id

[✓] Use Relationship in Query

A not found in B   B not found in A   [✓ Suppress Mismatches]

Creating a User-Defined Relationship

When you create a user-defined relationship, Query Builder draws a relationship line connecting the related columns.

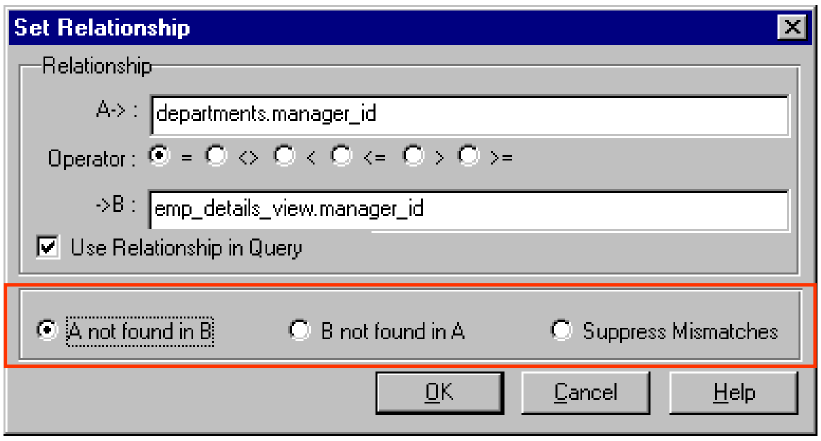
How to Create a Relationship

1. Select the Set Table Relationship tool.
   The Set Relationship dialog box appears.
2. Enter the foreign key (A >) and primary key (B >) column names.
3. Type the complete table and column names (separate the table name from its column name with a period).
4. Click OK to close the dialog box.

How to Create a Relationship (Optional Method)

1. Select the column that you want to relate (foreign key).
2. Press and hold the mouse button, and drag the cursor to the related column in the second table (the primary key).
   You are drawing a relationship line as you do so.
3. Once the target column is selected, release the mouse button to anchor the relationship line.

Unmatched Rows


Set Relationship

Relationship

A-> : departments.manager_id

Operator : [✓ =] <> < <= > >=

->B : emp_details_view.manager_id

[✓] Use Relationship in Query

[✓ A not found in B] B not found in A Suppress Mismatches

Retrieving Unmatched Rows

Query Builder enables you to choose whether to retrieve any unmatched rows when you are using a relationship in a query. An unmatched row occurs when the relationship connects tables where there are values on one side that have no corresponding values on the other side.

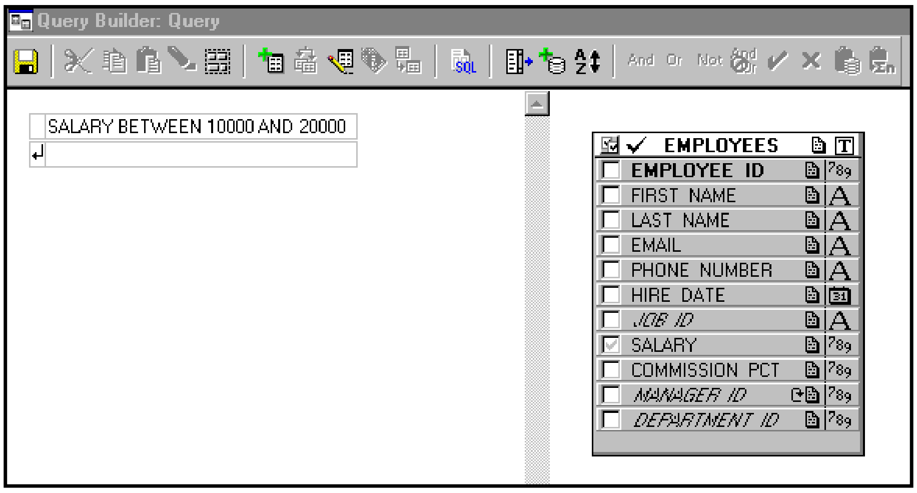
There are three types of relationships that you can choose from:

- Display records from table A not found in table B
- Display records from table B not found in table A
- Suppress any mismatched records (default)

How to Create an Unmatched Relationship

1. Click on the relationship line that connects the tables.
   Both the column names and the relationship line should be selected.
2. Select the Set Table Relationship tool from the toolbar.
   The Set Relationship dialog box appears.
3. Choose one of the three relationship option buttons.
4. Click OK.
   An unmatched relationship icon is placed on the relationship line next to the column that returns unmatched rows.

Conditions



SALARY BETWEEN 10000 AND 20000

Selecting Rows with Conditions

The Conditions Panel

The Query window contains two independently scrollable panels, the Conditions panel and the Datasource panel. The Datasource panel is where you include tables and columns. The Conditions panel is where you apply conditions. You enter conditions into the Condition field of the panel.

How to Add Conditions to a Query

1. Activate the Conditions field.
2. Enter the text that describes the condition in one of the following ways:
   - Type the conditions directly into the Condition field
   - Click in the columns in the Datasource panel and enter the rest of the condition
   - Select columns and functions, using the appropriate tools.

Note: Character and date values must be enclosed in single quotes.

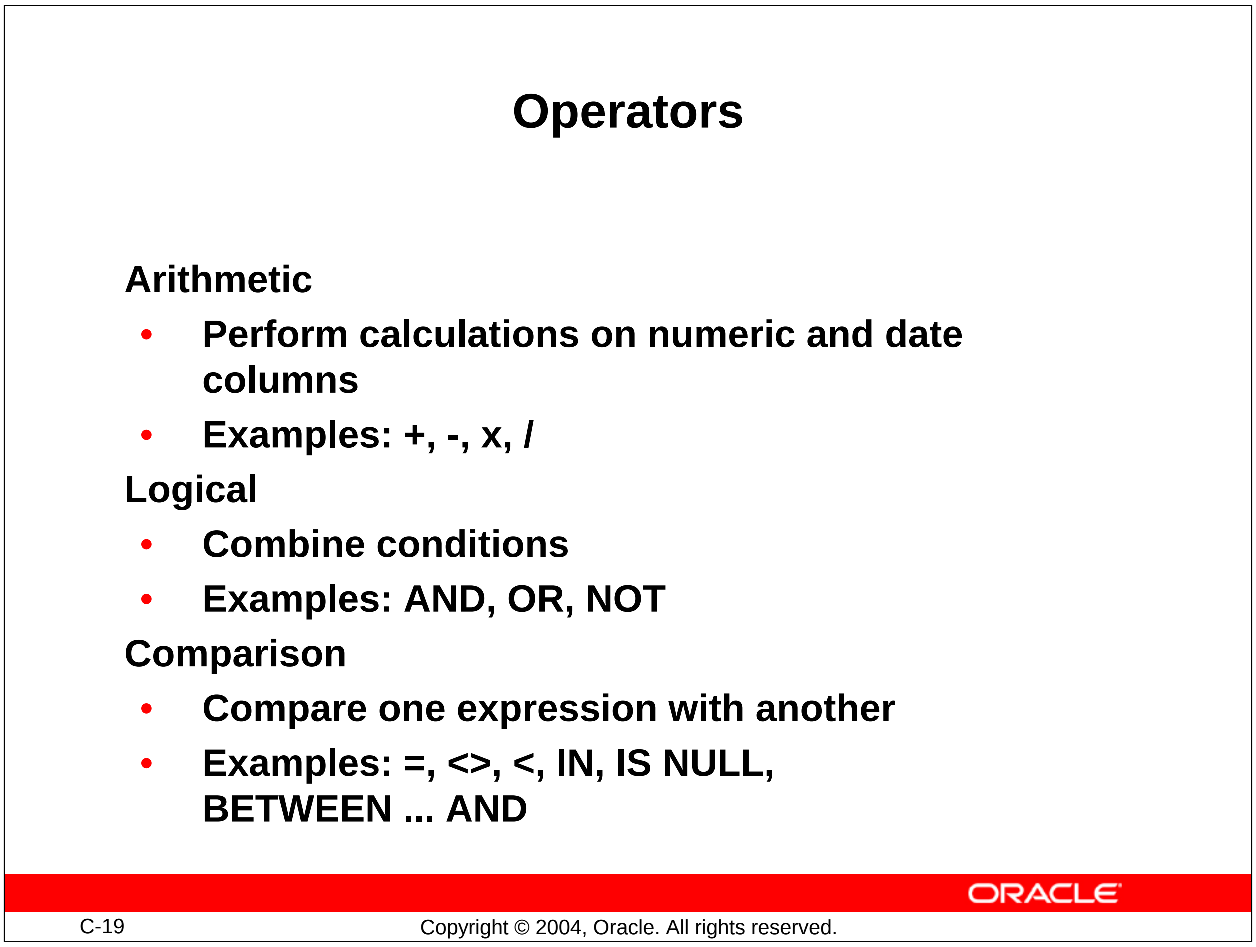
Selecting Rows with Conditions (continued)

3. Closing and Validating the Condition: Query Builder automatically validates the condition when you close the Condition field. You can close the Condition field in the following ways:
   - Press [Return].
   - Click in the Conditions panel outside the Condition field.
   - Select Accept from the toolbar.

Note: If a column is used in a condition but it is not displayed in the results window, a gray check mark appears to the left of the column name in the datasource in the Query Window.

The Toolbar

The logical operators and the Accept and Cancel tools in the toolbar are active whenever the Condition field is active. You can insert an operator from the toolbar into the condition by clicking it.



Operators

Arithmetic

- Perform calculations on numeric and date columns
- Examples: +, -, x, /

Logical

- Combine conditions
- Examples: AND, OR, NOT

Comparison

- Compare one expression with another
- Examples: =, <>, <, IN, IS NULL, BETWEEN ... AND

Operators

An operator is a character or reserved word that is used to perform some operation in Query Builder, such as the + operator, which performs addition. Query Builder uses several types of operators.

Arithmetic Operators

Arithmetic (+, -, x, /) operators are used to perform calculations on numeric and date columns. In handling calculations, Query Builder first evaluates any multiplication or division, and then evaluates any addition or subtraction.

Logical Operators

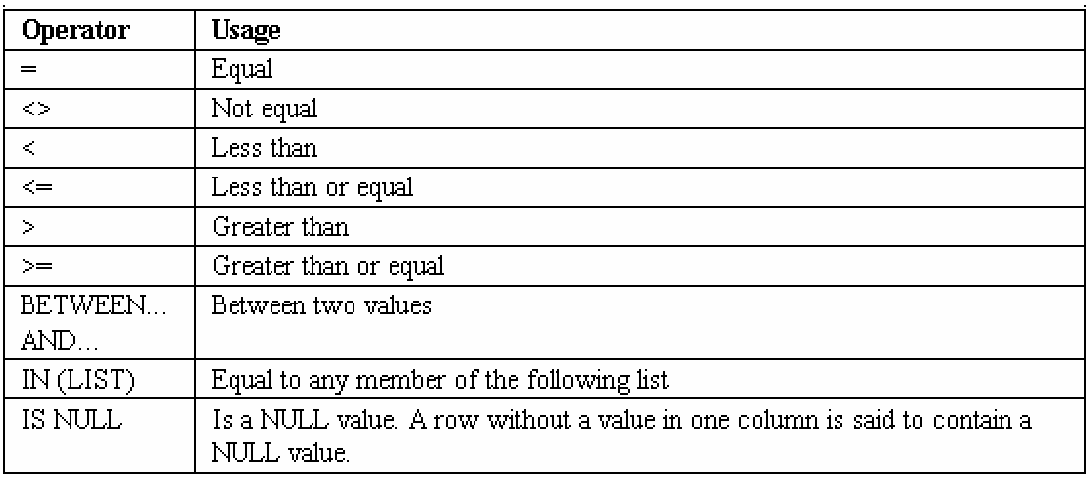
Logical operators are used to combine conditions. They include:

- **AND:** Causes the browser to retrieve only data that meets all conditions
- **OR:** Causes the browser to retrieve all data that meets at least one of the conditions
- **NOT:** Used to make a negative condition, such as NOT NULL

Comparison Operators

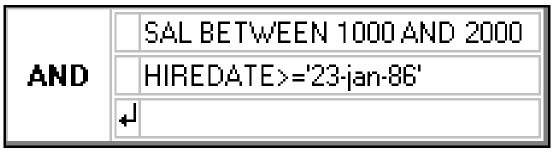
Comparison operators are used to compare one expression with another, such that the result will either be true or false. The browser returns all data for which the result evaluates true.

Operators (continued)

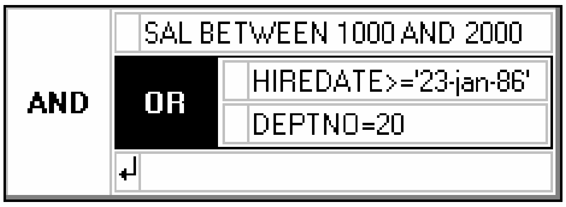


Operator	Usage
=	Equal
<>	Not equal
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
BETWEEN… AND…	Between two values
IN (LIST)	Equal to any member of the following list
IS NULL	Is a NULL value. A row without a value in one column is said to contain a NULL value.

Multiple Conditions



SAL BETWEEN 1000 AND 2000
HIREDATE>='23-jan-86'



SAL BETWEEN 1000 AND 2000
HIREDATE>='23-jan-86'
DEPTNO=20

Entering Multiple Conditions

There is no limit to the number of conditions you can include in a browser query. Multiple conditions are always combined using logical operators. You can add conditions to any query either before or after execution.

How to Add Conditions

The Conditions panel always displays a blank Condition field at the bottom of the list of conditions. Use this field to enter multiple conditions.

1. Click in the empty Condition field to activate it.
2. Enter the new condition.
   Each time you add a condition, a new blank Condition field is created.
   Note: Pressing [Shift]-[Return] following the entry of each condition is the fastest way to create multiple conditions, because it moves the cursor and prompt down one line so that you can enter another condition.
3. Press [Return] to close and validate the condition.

How to Change Logical Operators

By default, Query Builder combines multiple conditions with the AND operator. To change the logical operator:

Entering Multiple Conditions (continued)

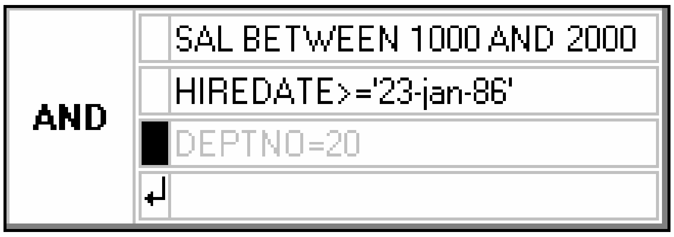
1. Select the logical operator in the Conditions panel.
2. Click a new operator on the toolbar or select one from the Data menu.

How to Create Nested Conditions

Query Builder enables you to combine logical operators to produce complex queries made up of multiple levels of conditions. These are referred to as nested conditions. To nest two or more conditions:

1. Build each condition to be included in the nest.
2. While you press and hold [Shift], click in the box to the left of each condition to be nested.
3. Select the logical operator to combine the conditions—either AND or OR.
   Query Builder draws a box around the highlighted conditions in the Conditions panel and combines them with the operator that you specified.

Deactivating a Condition


AND

SAL BETWEEN 1000 AND 2000

HIREDATE>='23-jan-86'

DEPTNO=20

Changing Conditions

If you change your mind about including one or more conditions in your query, you can delete, deactivate, or edit any of them in the Conditions panel.

How to Delete a Condition

1. Click inside the Condition field to activate it.
2. Select Clear from the toolbar or select Delete.
   The condition is removed from the query.

How to Deactivate a Condition

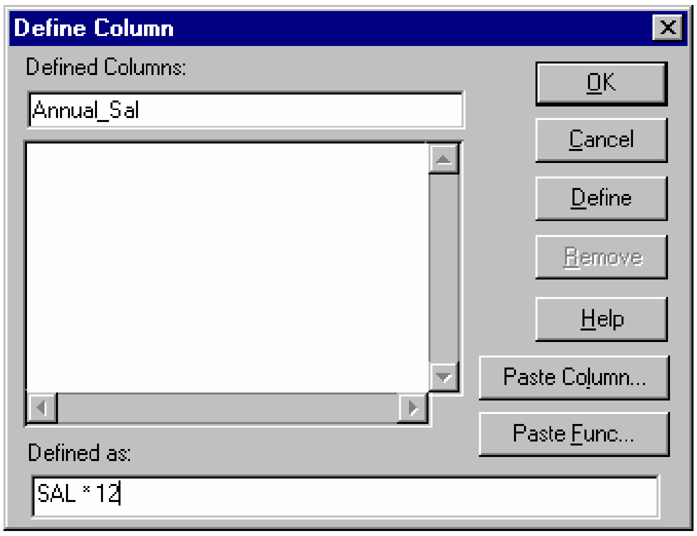
Query Builder enables you to temporarily deactivate a condition so that you can test different scenarios without having to re-create the conditions each time.

Double-clicking the box to the left of the condition or its operator acts as a toggle switch, alternately turning the condition on and off. Deactivated conditions remain in the Condition field but appear dimmed.

Additionally, you can turn conditions on and off by using the Data menu:

1. Select the box to the left of the condition or its operator.
   Note: Hold down the [Shift] key to select multiple conditions.
2. Double-click in the box to deactivate the condition.

Defining Columns Using an Expression


Define Column

Defined Columns: Annual_Sal

Defined as: SAL * 12

Defining Columns Using an Expression

Besides retrieving columns of data stored in a table, Query Builder enables you to define new columns that are derived or calculated from the values in another column.

How to Define a Column

When you define a column, it exists only in your query, not in the database.

1. Select the table where you want to define a new column.
2. Select the Define Column tool.
   The Defined Column dialog box appears, which displays a list of all columns currently defined in the query (if any).
3. Click in the Defined Column field to activate it, and then enter the name for your new column.
4. Move your cursor to the Defined As field and enter the formula or expression that defines your column.
5. Select Define.
   The new column is added to the list of defined columns in the dialog box and appears in the Datasource panel.
6. Click OK to close the dialog box.

Defining Columns Using an Expression (continued)

How to Enter Expressions

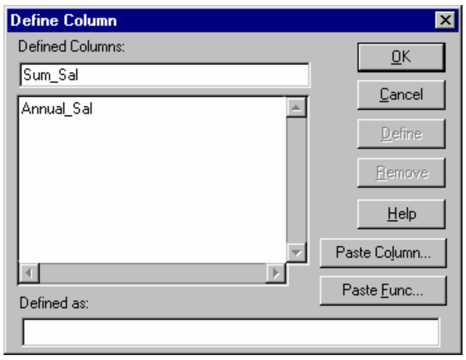
You can enter information in the Defined As field in the following ways:

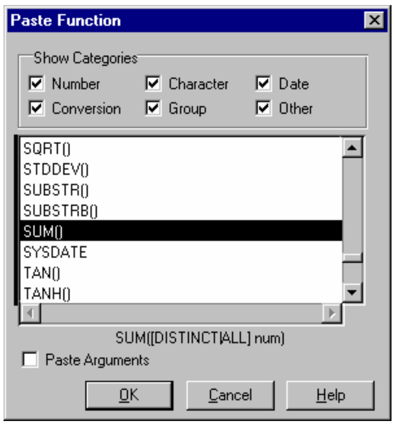
- Type the expression in directly.
- Select Paste Column.
  The Paste Column dialog box appears.
  - Select the column from the displayed list.
  - Click OK to paste the column into your expression and return to the Define Column dialog box.

How to Display and Hide Defined Columns

You display or hide defined columns from a query in exactly the same manner as you do ordinary columns.

Defining Columns Using a Function





ORACLE

Defining Columns Using a Function

The browser also enables you to define columns using a variety of built-in functions provided by the Oracle Server. You can type a function directly in the Defined As field of the Define Columns dialog box, or click Paste Function to select from a list.

What Is a Function?

A function is similar to an operator in that it performs a calculation and returns a result. Functions consist of a function name followed by parentheses, in which you indicate the arguments.
An argument is an expression that supplies information for the function to use. Functions usually include at least one argument, most commonly the name of the column on which the operation will be performed.

Single-Row Functions

- Return one value for every data row operated on
- Examples: INITCAP(), SUBSTR(), TRUNC()

Aggregate Functions

- Return a single row based on the input of multiple rows of data
- Examples: AVG(), COUNT(), SUM()

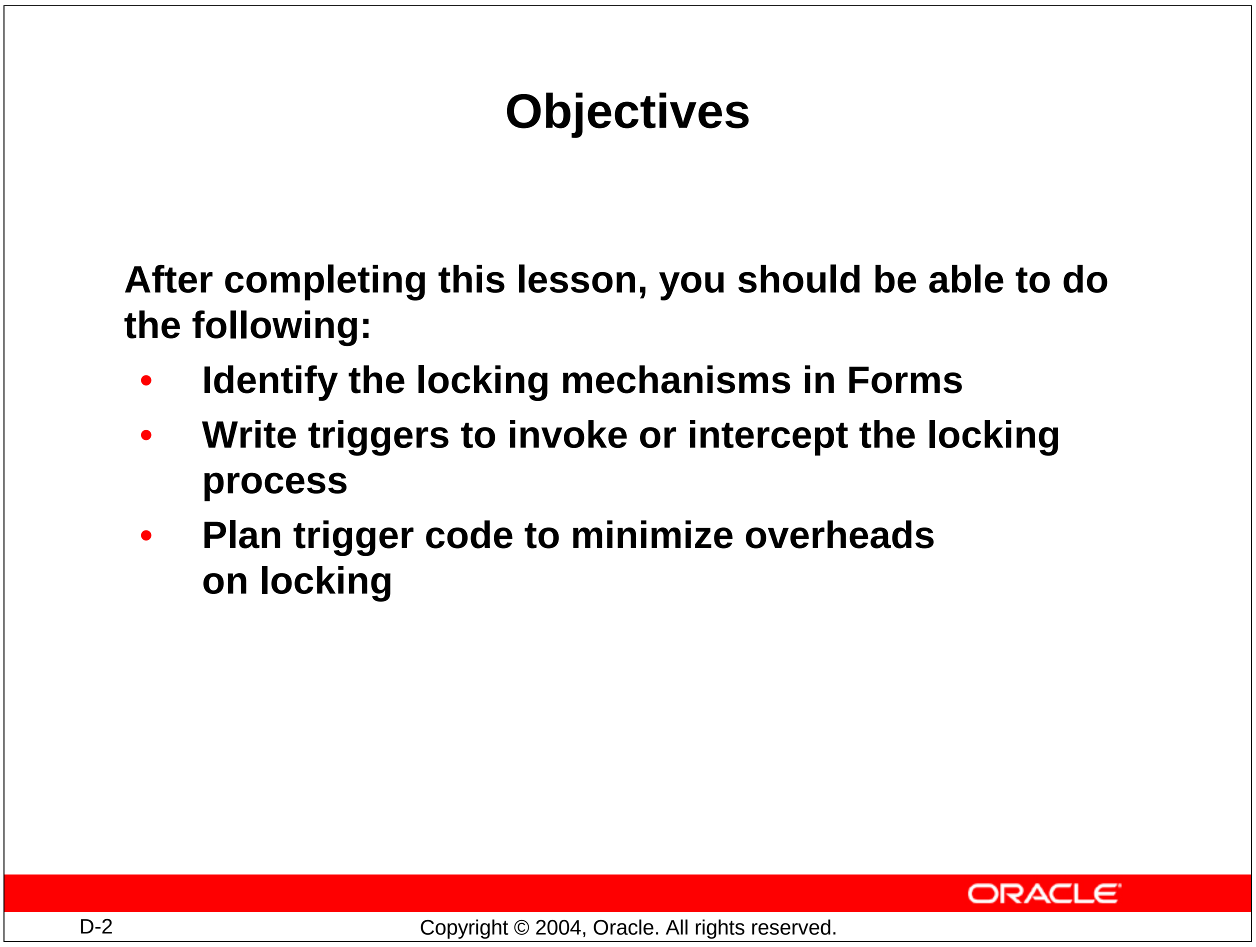
Defining Columns Using a Function (continued)

How to Select a Function

1. Click in the Defined As field to activate it.
2. Select Paste Function.
   The Paste Function dialog box appears.
3. Select or deselect the Show Categories check boxes to view the desired list of functions.
4. Select the function from the displayed list.
5. Select the Paste Arguments check box (optional).
   Note: If this check box is selected, a description or datatype name of the arguments appropriate for the function is pasted into your expression. You then replace the description with the actual arguments.
6. Click OK to paste the function into your expression and return to the Define Columns dialog box.

Locking in Forms

Schedule:	Timing	Topic
	17 minutes	Lecture
	17 minutes	Total



Objectives

After completing this lesson, you should be able to do the following:

- **Identify the locking mechanisms in Forms**
- **Write triggers to invoke or intercept the locking process**
- **Plan trigger code to minimize overheads on locking**

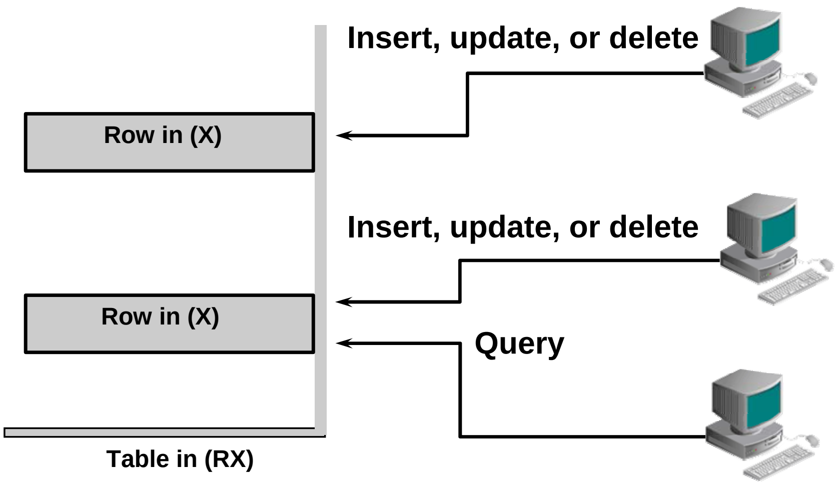
ORACLE

D-2

Overview

Locking is an important consideration in multiuser applications that access the database. This lesson shows you how Forms handles locking and how you can design forms with these mechanisms in mind.

Locking



Locking

In database applications, locking maintains the consistency and integrity of the data, where several users are potentially accessing the same tables and rows. Forms applications are involved in this locking process when they access database information.

Oracle Locking

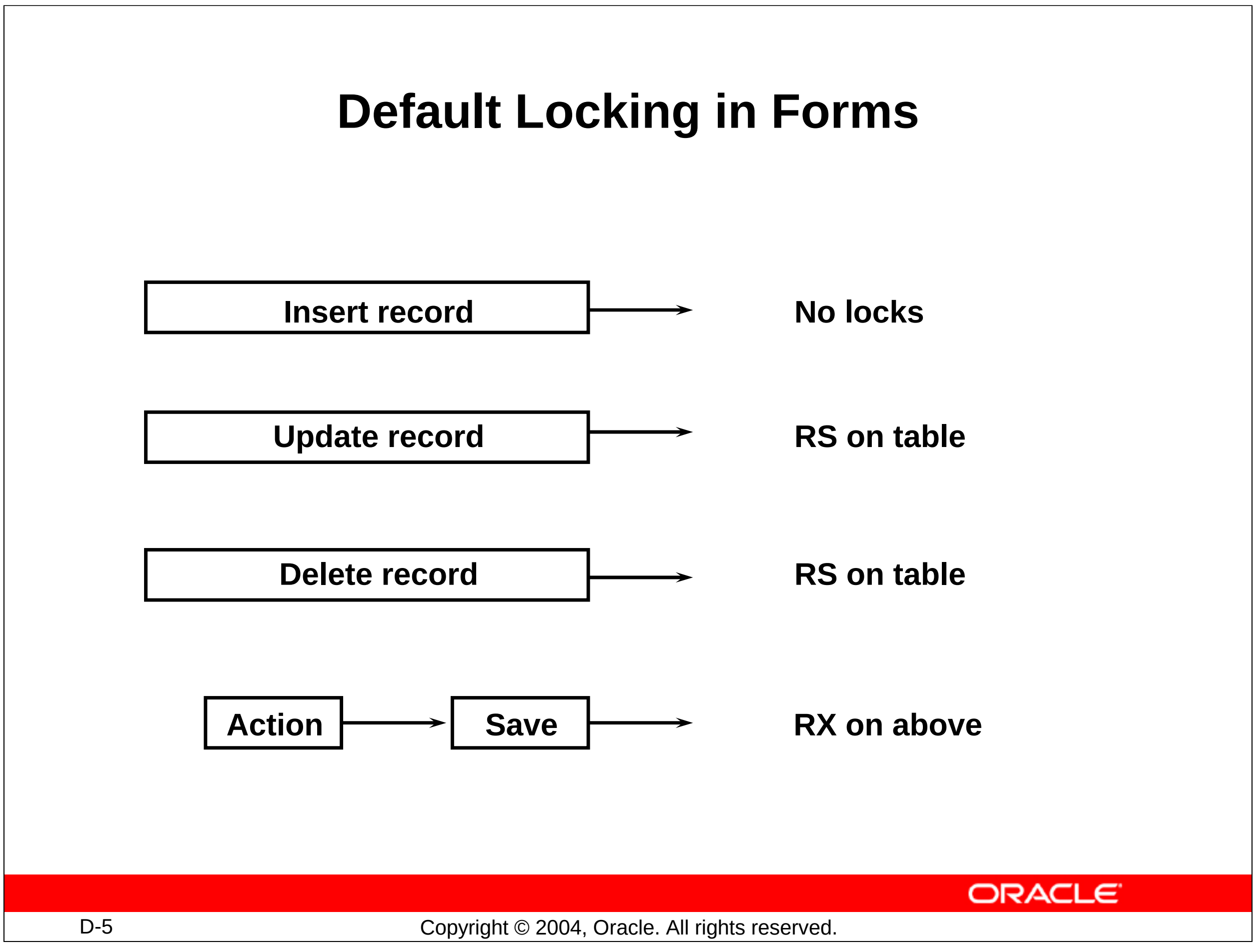
Forms applications that connect to an Oracle database are subject to the standard locking mechanisms employed by the server. Here is a reminder of the main points:

- Oracle uses row-level locking to protect data that is being inserted, updated, or deleted.
- Queries do not prevent other database sessions from performing data manipulation. This also applies to the reverse situation.
- Locks are released at the end of a database transaction (following a rollback or commit).

Locking (continued)

A session issues locks to prevent other sessions from performing certain actions on a row or table. The main Oracle locks that affect Forms applications are the following:

Lock Type	**Description**
Exclusive (X) row lock	Allows other sessions to only read the affected rows
Row Share (RS) table lock	Prevents the above lock (X) from being applied to the entire table; usually occurs because of `SELECT … FOR UPDATE`
Row Exclusive (RX) table lock	Allows write operations on a table from several sessions simultaneously, but prevents (X) lock on entire table; usually occurs because of a DML operation



Default Locking in Forms

D-5

Default Locking in Forms

Forms initiates locking automatically when the operator inserts, updates, or deletes records in a base table block. These locks are released when a save is complete.

Forms causes the following locks on Oracle base tables and rows, when the operator performs actions:

Operator Action	**Locks**
Insert a record	No locks
Update database items in a record*	Row Share (RS) on base table Exclusive (X) on corresponding row
Save	Row Exclusive (RX) on base tables during the posting process (Locks are released when actions are completed successfully.)

*Update of nondatabase items with the Lock Record property set to Yes also causes this.

The exclusive locks are applied to reserve rows that correspond to records that the operator is deleting or updating, so that other users cannot perform conflicting actions on these rows until the locking form has completed (Saved) its transaction.

Concurrent Updates and Deletes

- **When users compete for the same record, normal locking protection applies.**
- **Forms tells the operator if another user has already locked the record.**

Concurrent Updates and Deletes

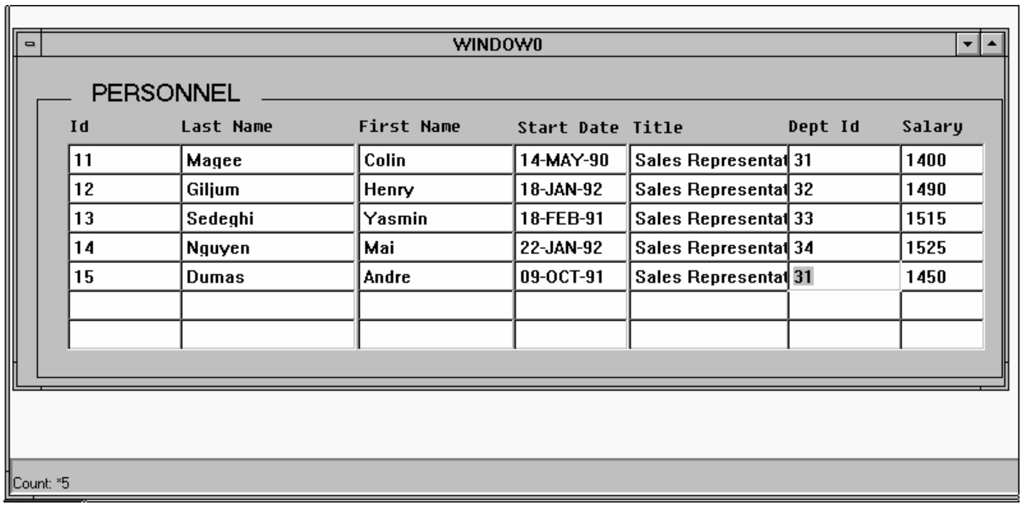
What Happens When Users Compete for the Same Row?

Users who are only querying data are not affected here. If two users are attempting to update or delete the same record, then integrity is protected by the locking that occurs automatically at row level.

Forms also keeps each user informed of these events through messages and alerts when:

- A row is already locked by another user. The user has the option of waiting or trying again later.
- Another user has committed changes since a record was queried. The user must requery before the user’s own change can be applied.

User A: Step 1



WINDOW0

PERSONNEL

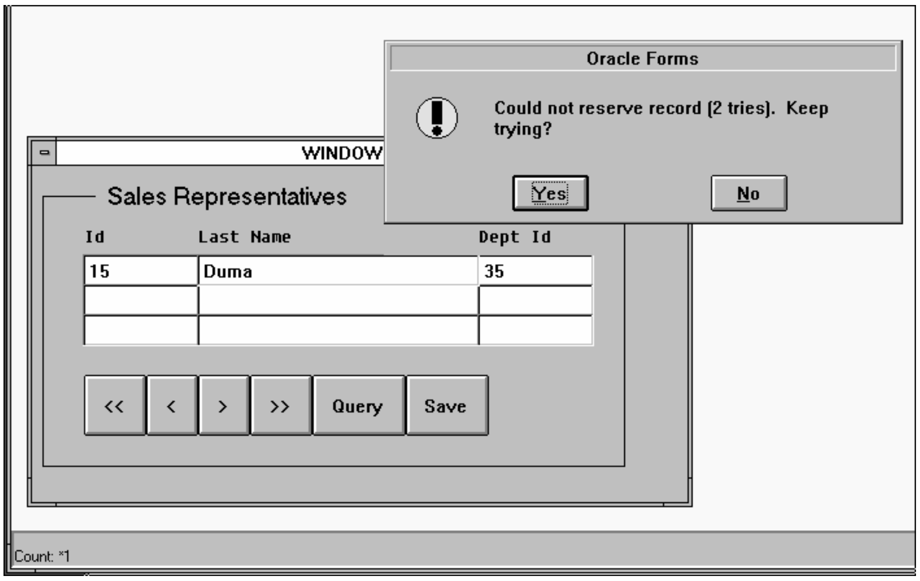
Id	Last Name	First Name	Start Date	Title	Dept Id	Salary
11	Magee	Colin	14-MAY-90	Sales Representat	31	1400
12	Giljum	Henry	18-JAN-92	Sales Representat	32	1490
13	Sedeghi	Yasmin	18-FEB-91	Sales Representat	33	1515
14	Nguyen	Mai	22-JAN-92	Sales Representat	34	1525
15	Dumas	Andre	09-OCT-91	Sales Representat	31	1450

Count: *5

Example: Two Users Accessing the Same Record

1. User A is running the Personnel application and queries the sales representatives. The record for employee 15, Dumas, is updated so that his department is changed to 31.
   User A does not save the change at this point in time. The row that corresponds to the changed record is now locked (exclusively).

User B: Step 2



Could not reserve record (2 tries). Keep trying?

Id	Last Name	Dept Id
15	Duma	35

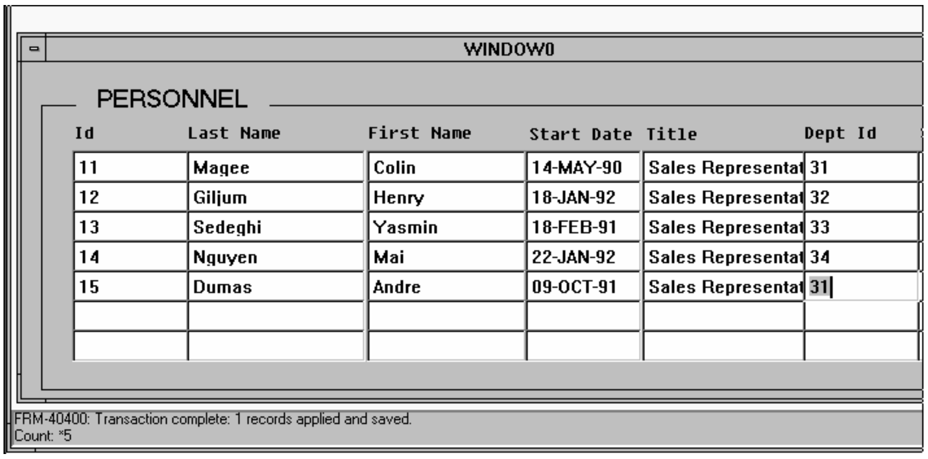
Example: Two Users Accessing the Same Record (continued)

2. User B is running the same application and has started a form that accesses the sales representatives. Employee Dumas still appears in department 15 in this form, because User A has not yet saved the change to the database.
   User B attempts to update the record by changing the sales representative's name to Agasi. Since this action requests a lock on the row, and this row is already locked by User A, Forms issues an alert saying the attempt to reserve the record failed. This user can request additional attempts or reply No and try later.
   User B replies No. This results in the fatal error message 40501, which confirms that the original update action has failed.

Instructor Note

Show a similar example, by starting two separate Forms run-time sessions. Apply the actions as given in steps 1-4.

User A: Step 3



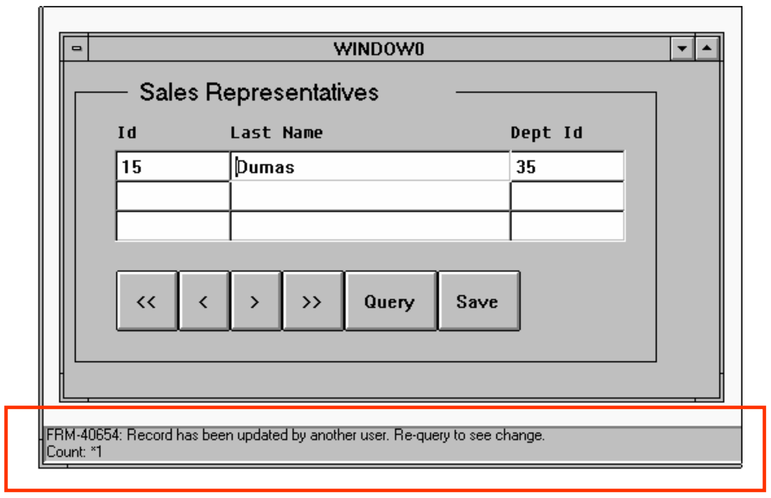
Id	Last Name	First Name	Start Date	Title	Dept Id
11	Magee	Colin	14-MAY-90	Sales Representat	31
12	Giljum	Henry	18-JAN-92	Sales Representat	32
13	Sedeghi	Yasmin	18-FEB-91	Sales Representat	33
14	Nguyen	Mai	22-JAN-92	Sales Representat	34
15	Dumas	Andre	09-OCT-91	Sales Representat	31

```
FRM-40400: Transaction complete: 1 records applied and saved.
Count: *5
```

Example: Two Users Accessing the Same Record (continued)

3. Back in the Personnel form, User A now saves the change to Dumas' department, which applies the change to the database row and then releases the lock at the end of the transaction.

User B: Step 4



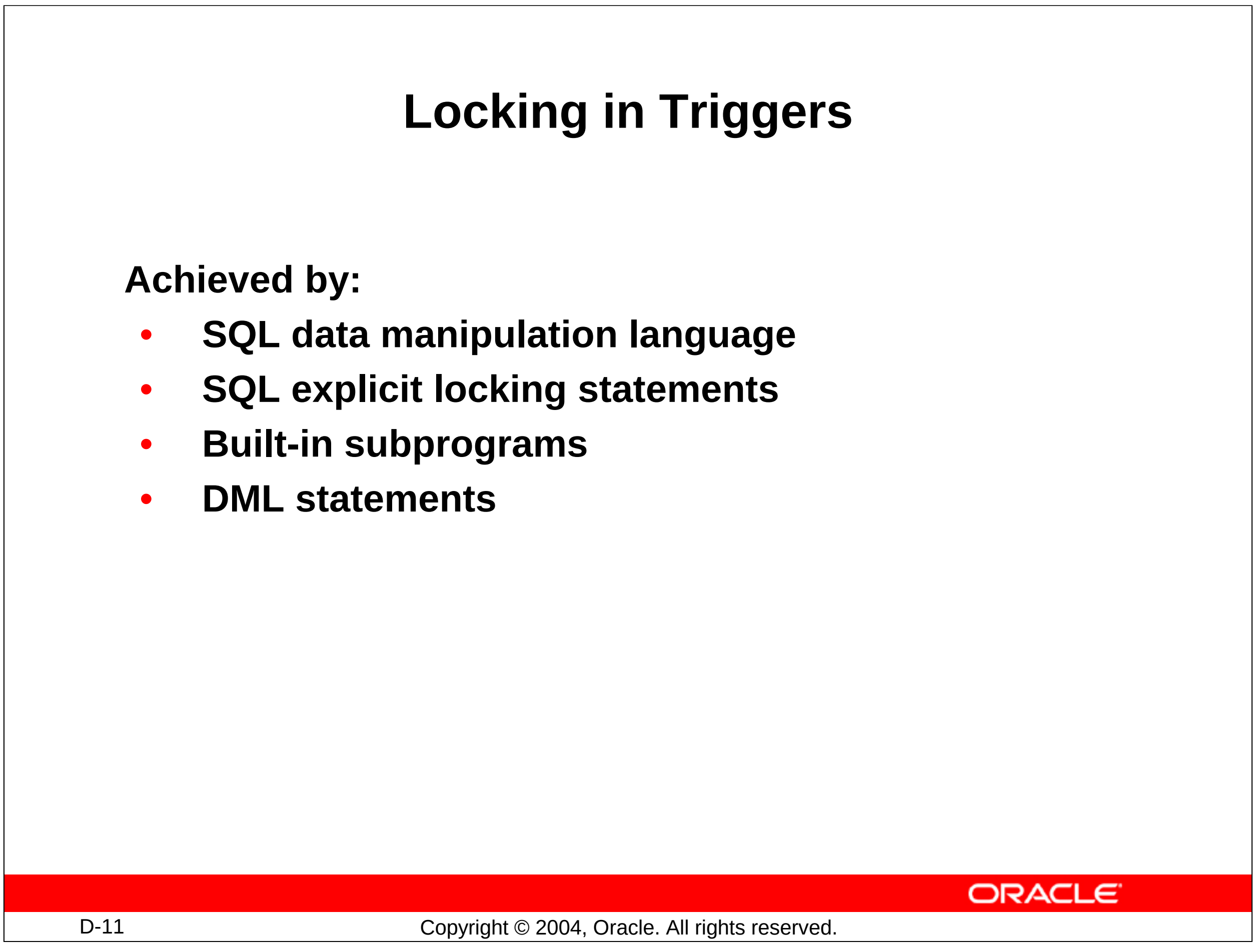
Id	Last Name	Dept Id
15	Dumas	35

FRM-40654: Record has been updated by another user. Re-query to see change.
Count: *1

D-10

Example: Two Users Accessing the Same Record (continued)

4. In the Sales Representatives form, User B can now alter the record. However, because the database row itself has now changed since it was queried in this form, Forms tells User B that it must be requeried before a change can be made.
   This situation is shown in the lower slide, opposite. Once User B requeries, the record can then be changed.



Locking in Triggers

Achieved by:

- SQL data manipulation language
- SQL explicit locking statements
- Built-in subprograms
- DML statements

Locking in Triggers

In addition to the default locking described earlier, database locks can occur in Forms applications because of actions that you include in triggers. These can be:

- **SQL data manipulation language (DML):** Sometimes you may need to perform `INSERT`, `UPDATE`, and `DELETE` statements, which add to those that Forms does during the saving process (posting and committing). These trigger SQL commands cause implicit locks on the tables and rows that they affect.
- **SQL locking statements:** You can explicitly issue locks from a trigger through the `SELECT . . . FOR UPDATE` statement. The `LOCK TABLE` statement is also allowed, though rarely necessary.
- **Built-in subprograms:** Certain built-ins allow you to explicitly lock rows that correspond to the current record (`LOCK_RECORD`) or to records fetched on a query (`ENTER_QUERY and EXECUTE_QUERY`).

To keep locking duration to a minimum, DML statements should be used only in transactional triggers. These triggers fire during the process of applying and saving the user's changes, just before the end of a transaction when locks are released.

Locking in Triggers (continued)

Locking by DML Statements

If you include DML statements in transactional triggers, their execution causes:

- Row exclusive lock on the affected table
- Exclusive lock on the affected rows

Because locks are not released until the end of the transaction, when all changes have been applied, it is advantageous to code DML statements as efficiently as possible, so that their actions are completed quickly.

Locking with Built-Ins

- **`ENTER_QUERY (FOR_UPDATE)`**
- **`EXECUTE_QUERY (FOR_UPDATE)`**

D-13

Locking with Built-Ins

Forms maintains a hidden item called rowid in each base table block. This item stores the ROWID value for the corresponding row of each record. Updates or deletes in triggers that apply to such rows can identify them most efficiently using this value, as in the example below:

```
UPDATE orders
SET     order_date = SYSDATE
WHERE   ROWID = :orders.rowid;
```

Locking with Built-in Subprograms

The following built-ins allow locking:

- **`EXECUTE_QUERY (FOR_UPDATE)` and `ENTER_QUERY (FOR_UPDATE)`:** When called with the `FOR_UPDATE` option, these built-ins exclusively lock the rows fetched for their query. Care should be taken when reserving rows in this way, because large queries cause locking on many rows.
- **`LOCK_RECORD`:** This built-in locks the row that corresponds to the current record in the form. This is typically used in an On-Lock trigger, where it has the same effect as Forms's default locking.

On-Lock Trigger

Example

```
IF USER = 'MANAGER' THEN
  LOCK_RECORD;
ELSE
  MESSAGE('You are not authorized to change
   records here');
  RAISE form_trigger_failure;
END IF;
```

On-Lock Trigger

This block-level trigger replaces the default locking that Forms normally carries out, typically when the user updates or deletes a record in a base table block. The trigger fires before the change to the record is displayed. On failure, the input focus is set on the current item.

Use this trigger to:

- Bypass locking on a single-user system, hence speeding processing
- Conditionally lock the record or fail the trigger (Failing the trigger effectively fails the user's action.)
- Handle locking when directly accessing non-Oracle data sources

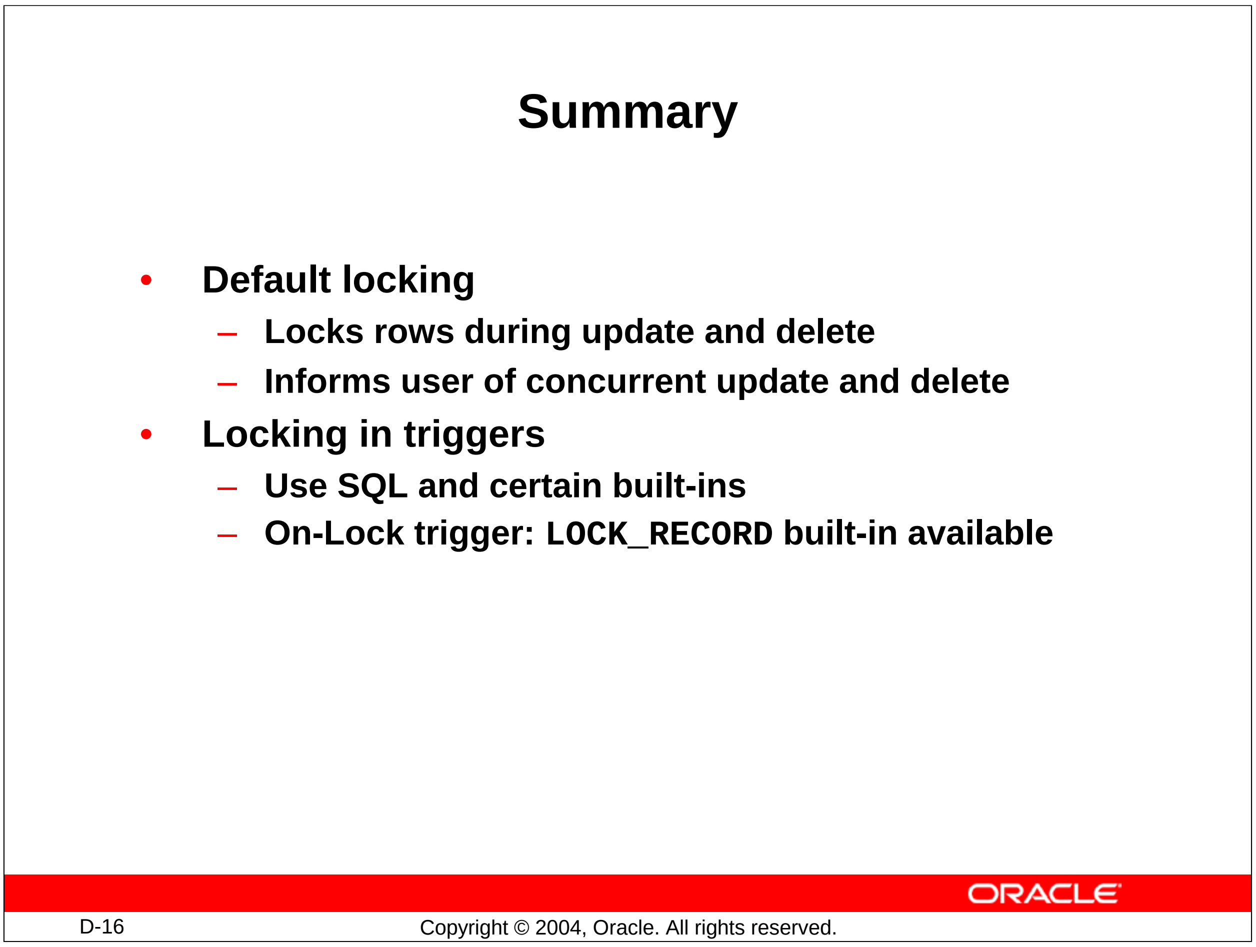
If this trigger succeeds, but its action does not lock the record, then the row remains unlocked after the user's update or delete operation. Use the LOCK_RECORD built-in within the trigger if locking is not to be bypassed.

On-Lock Trigger (continued)

Example

The following On-Lock trigger on the block Stock only permits the user MANAGER to lock records for update or delete.

```
IF USER = 'MANAGER' THEN
     LOCK_RECORD;
          IF NOT FORM_SUCCESS THEN
            RAISE form_trigger_failure;
          END IF;
ELSE
     MESSAGE('You are not authorized to change records
     here');
     RAISE  form_trigger_failure;
END IF;
```



Summary

- Default locking
  - Locks rows during update and delete
  - Informs user of concurrent update and delete
- Locking in triggers
  - Use SQL and certain built-ins
  - On-Lock trigger: `LOCK_RECORD` built-in available

D-16

Summary

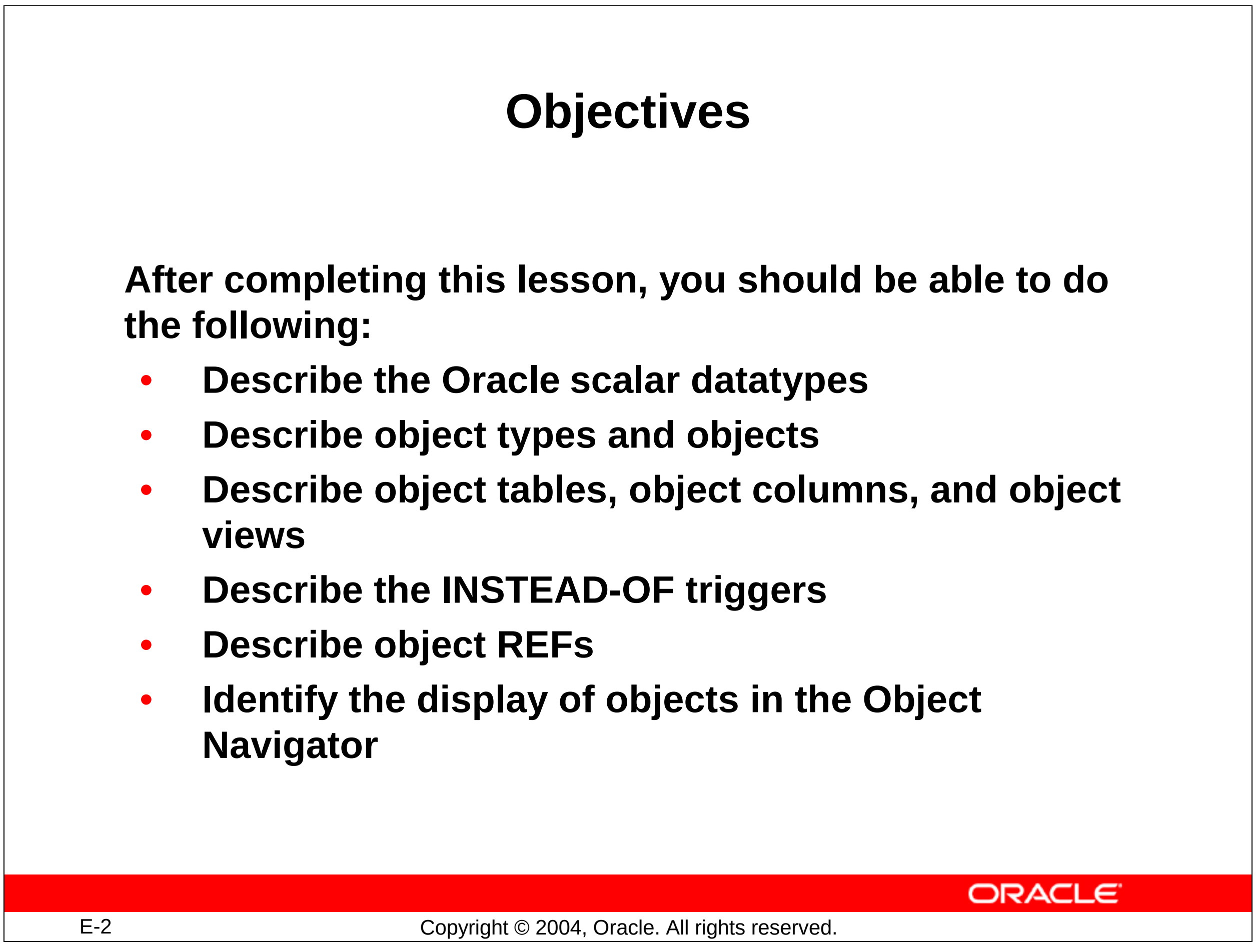
Default Locking

- Locks rows during update and delete
- Informs user about concurrent update and delete

Locking in Triggers

- Use SQL and certain built-ins
- On-Lock trigger: `LOCK_RECORD` built-in available

E Oracle Object Features



Objectives

After completing this lesson, you should be able to do the following:

- **Describe the Oracle scalar datatypes**
- **Describe object types and objects**
- **Describe object tables, object columns, and object views**
- **Describe the INSTEAD-OF triggers**
- **Describe object REFs**
- **Identify the display of objects in the Object Navigator**

Overview

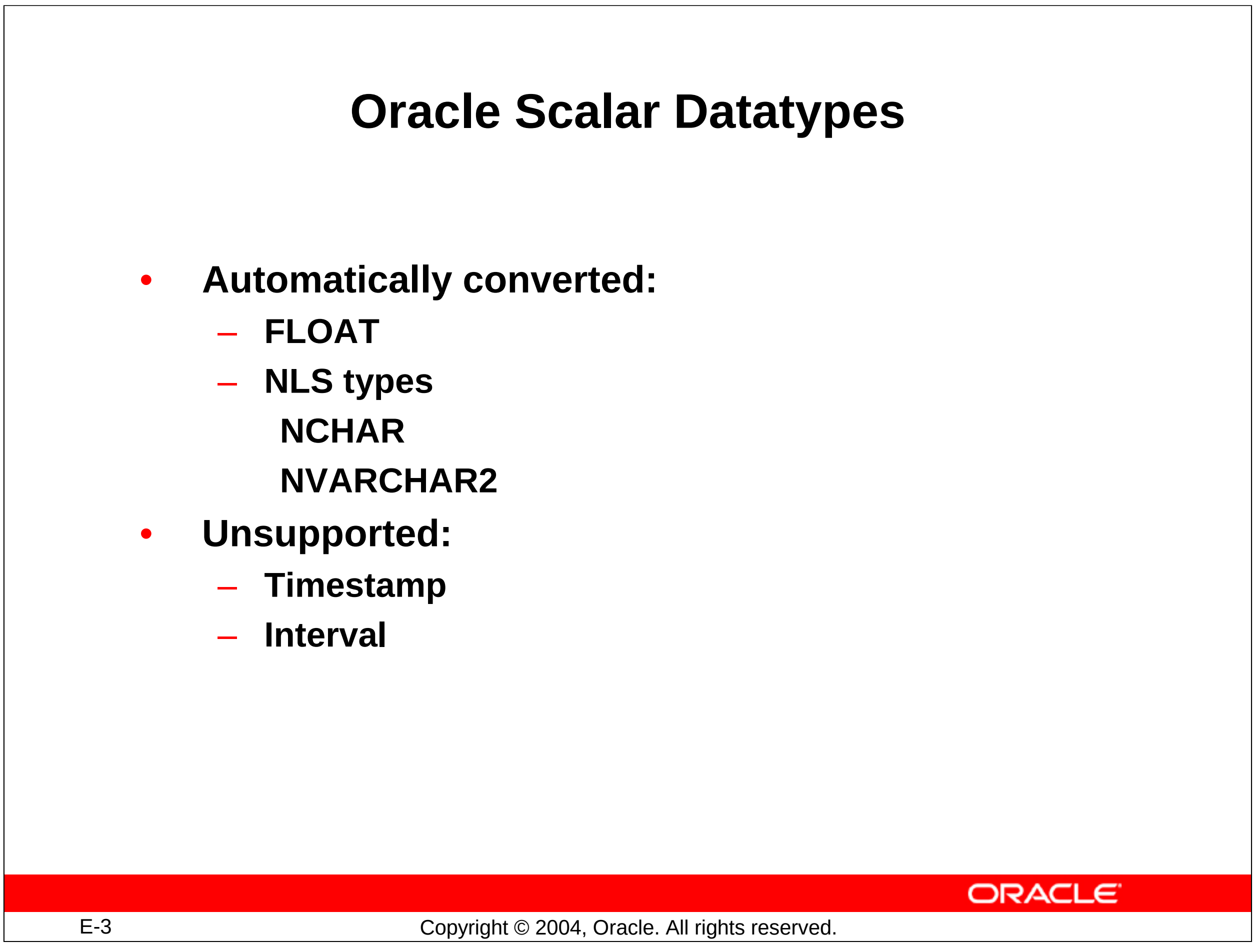
Introduction

In this lesson, you will review certain object features of Oracle. This lesson also explains how objects are displayed in the Object Navigator.

Objectives

After completing this lesson, you should be able to do the following:

- Describe the Oracle scalar datatypes
- Describe object types and objects
- Describe object tables, object columns, and object views
- Describe the INSTEAD-OF triggers
- Describe object REFs
- Identify the display of objects in the Object Navigator



Oracle Scalar Datatypes

- Automatically converted:
  - FLOAT
  - NLS types

    NCHAR

    NVARCHAR2
- Unsupported:
  - Timestamp
  - Interval

E-3

Oracle Datatypes

In Oracle Forms Builder, these datatypes are automatically converted to existing Forms item datatypes. Three new scalar datatypes were introduced with Oracle8*i*:

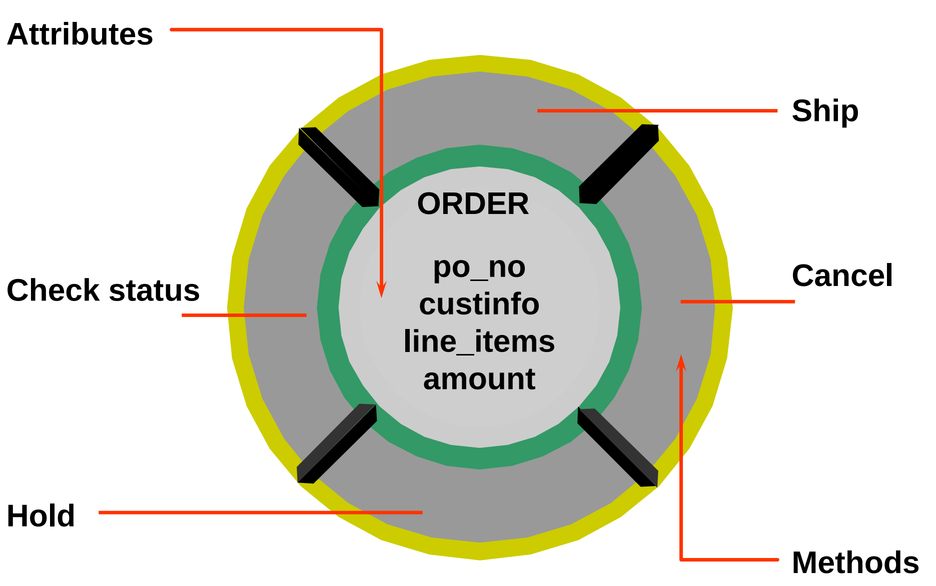
- `NCHAR` stores fixed-length (blank-padded if necessary) NLS (National Language Support) character data.
- `NVARCHAR2` stores variable-length NLS character data.
- `FLOAT` is a subtype of `NUMBER`. However, you cannot specify a scale for `FLOAT` variables. You can only specify a binary precision, which is the total number of binary digits.

NLS (National Language Support) Types in Oracle: In the Oracle database, data stored in columns of `NCHAR` or `NVARCHAR2` datatypes are exclusively stored in Unicode, regardless of the database character set. This enables users to store Unicode in a database that may not use Unicode as the database character set. The Oracle database supports two Unicode encodings for the Unicode datatypes: `AL16UTF16` and `UTF8` .

With NLS, number and date formats adapt automatically to the language conventions specified for a user session. Users around the world can interact with the Oracle server in their native languages. NLS is discussed in *Oracle Database Globalization Support Guide*.

Unsupported: Timestamp and interval datatypes (new with Oracle9*i*).

Object Types



Object Types

An object type is a user-defined composite datatype. Oracle requires enough knowledge of a user-defined datatype to interact with it. Thus, in some sense an object type is similar to a record type, and in some sense it is similar to a package.

An object type must have one or more attributes and can contain methods.

Attributes: An object type is similar to a record type in that it is declared to be composed of one or more subparts that are of predefined datatypes. Record types call these subparts *fields*, but object types call them *attributes*. Attributes define the object structure.

```
CREATE TYPE address_type AS OBJECT
     (address   VARCHAR2(30),
      city      VARCHAR2(15),
      state          CHAR(2),
      zip       CHAR(5));
CREATE TYPE phone_type AS OBJECT
     (country   NUMBER(2),
      area      NUMBER(4),
      phone          NUMBER(9));
```

Object Types (continued)

Just as the fields of a record type can be of other record types, the attributes of an object type can be of other object types. Such an object type is called *nested*.

```
CREATE TYPE address_and_phone_type AS OBJECT
     (address   address_type,
      phone          phone_type);
```

Object types are like record types in another sense: Both of them must be declared as types before the actual object or record can be declared.

Methods

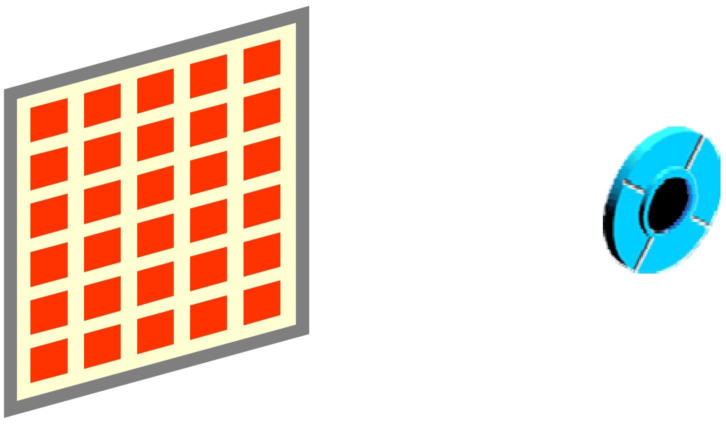
An object type is also similar to a package. Once an object is declared, its attributes are similar to package variables. Like packages, object types can contain procedures and functions. In object types, these subprograms are known as *methods*. A method describes the behavior of an object type.

Like packages, object types can be declared in two parts: a specification and a body. As with package variables, attributes declared in the object type specification are public and those declared in the body are private. As with package subprograms, all methods are defined in the package body, but only those whose specification appears in the object type specification are public methods.

Here is an example of an object type:

```
CREATE TYPE dept_type AS OBJECT
     (dept_id        NUMBER(2),
     dname      VARCHAR2(14),
     loc             VARCHAR2(3),
     MEMBER PROCEDURE set_dept_id (d_id NUMBER),
          PRAGMA RESTRICT_REFERENCES (set_dept_id,
                    RNDS,WNDS,RNPS,WNPS),
          MEMBER FUNCTION get_dept_id RETURN NUMBER,
               PRAGMA RESTRICT_REFERENCES (get_dept_id,
               RNDS,WNDS,RNPS,WNPS));
CREATE TYPE BODY dept_type AS
     MEMBER PROCEDURE set_dept_id (d_id NUMBER)
     IS
     BEGIN
          dept_id := d_id;
     END;
     MEMBER FUNCTION get_dept_id
          RETURN NUMBER
     IS
     BEGIN
          RETURN (dept_id);
     END;
END;
```

Object Tables



Object table based on object type

Creating Oracle Objects

Once you have declared an object type, you can create objects based on the type.

Object Tables

One way to create an object is to create a table whose rows are objects of that object type. Here is an example of an object table declaration:

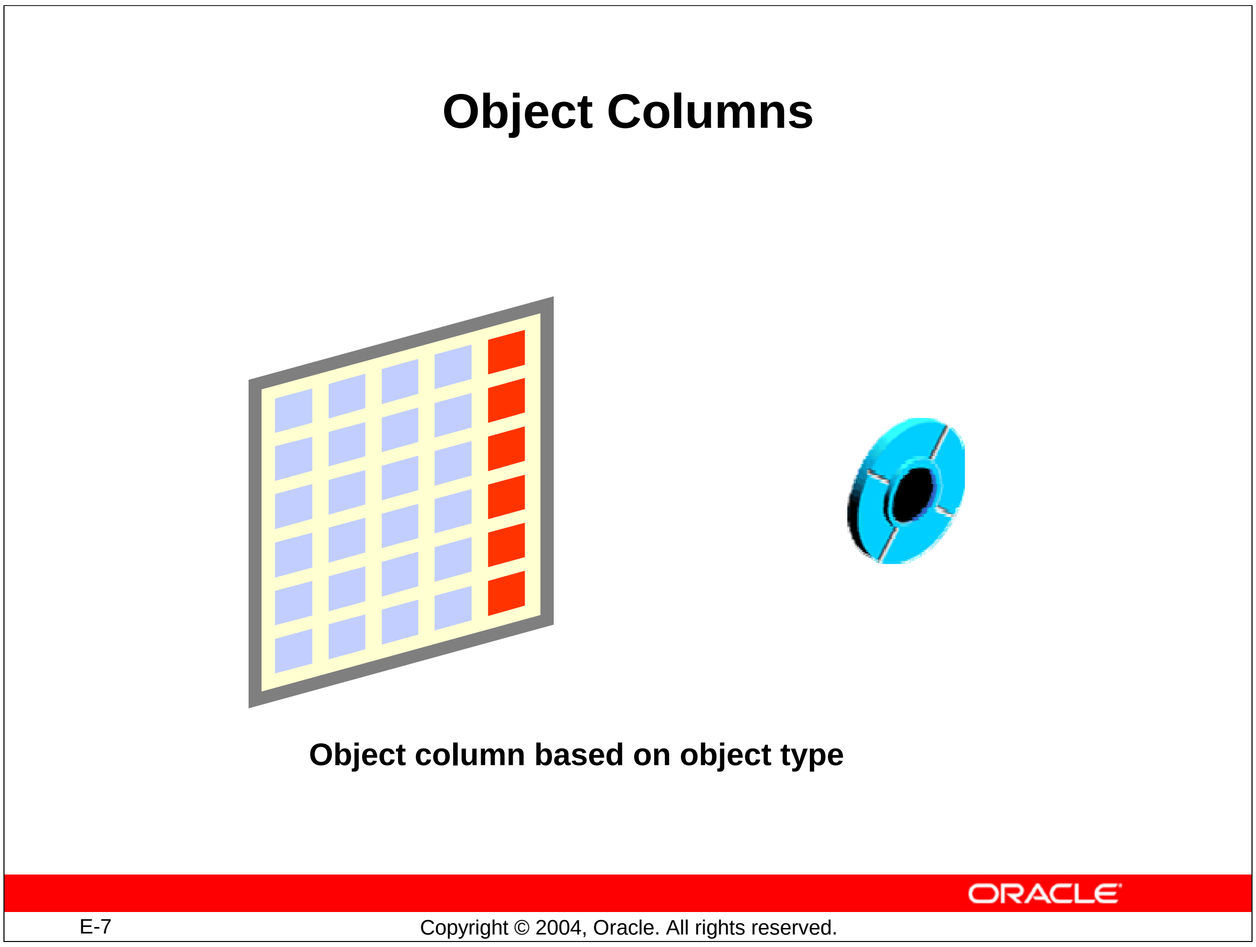
```
CREATE TABLE o_dept OF dept_type;
```

SQL and PL/SQL treat object tables very similarly to relational tables, with the attribute of the object corresponding to the columns of the table. But there are significant differences. The most important difference is that rows in an object table are assigned object IDs (OIDs) and can be referenced using a REF type.

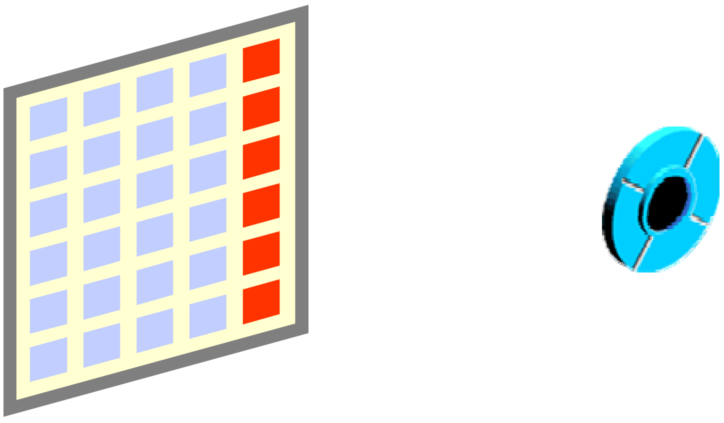
Note: REF types are reviewed later.

Instructor Note

At this point, review the fact that object types are not themselves objects. They are only blueprints for objects. The term object can be confusing because Oracle uses the term to refer to constructs within the database—for example, tables, views, procedures, and so on. This is a completely different usage. Make certain that your students are aware of the overloading of this term.



Object Columns



Object column based on object type

E-7

Object Columns

Another construct that can be based on an object type is an object column in a relational table. Here is an example of a relational table creation statement with an object column:

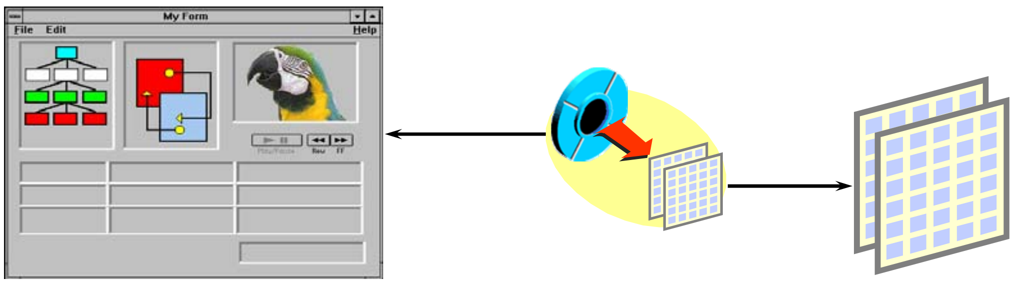
```
CREATE TABLE o_customer (
     custid               NUMBER (6) NOT NULL,
     name            VARCHAR2 (45),
     repid                NUMBER (4) NOT NULL,
     creditlimit          NUMBER (9,2),
     address              address_type,
     phone           phone_type);
```

In the object table, the rows of a table are objects. In a relational table with an object column, the column is an object. The table will usually have standard columns, as well as one or more object columns.

Object columns are not assigned object IDs (OIDs) and therefore cannot be referenced using object REF values.

Note: Object REFs are reviewed later in this section.

Object Views



Object-oriented application **Object view** **Relational table**

Object views based on object types

Object Views

Often, the most difficult part of adopting a new technology is the conversion process itself. For example, a large enterprise might have several applications accessing the same data stored in relational tables. If such an enterprise decided to start using object-relational technology, the enterprise would not convert all of the applications at once. It would convert the applications one at a time.

That presents a problem. The applications that have been converted need the data stored as objects, whereas the applications that have not been converted need the data stored in relational tables.

This dilemma is addressed by object views. Like all views, an object view transforms the way a table appears to a user, without changing the actual structure of the table. Object views make relational tables look like object tables. This allows the developers to postpone converting the data from relational structures to object-relational structures until after all of the applications have been converted. During the conversion process, the object-relational applications can operate against the object view; the relational applications can continue to operate against the relational tables.

Objects accessed through object views are assigned object IDs (OIDs), and can be referenced using object REFs.

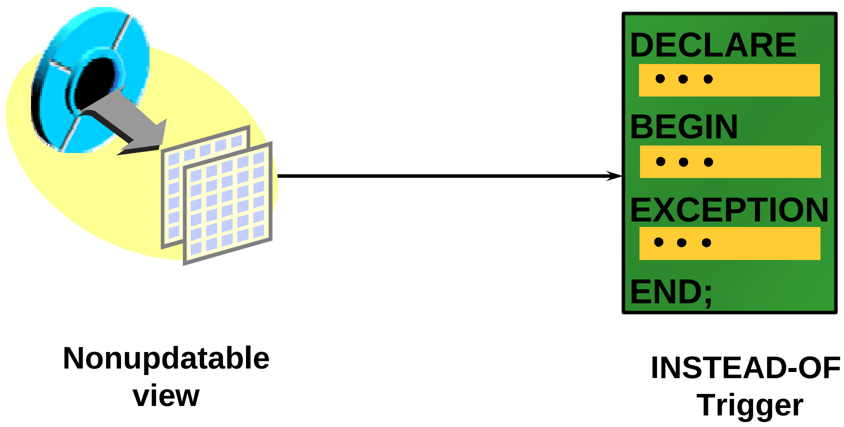
Object Views (continued)

Note: Object REFs are reviewed later in this section.

Here is an example of an object view creation statement:

```
CREATE VIEW emp_view OF emp_type
WITH OBJECT OID (eno)
AS
SELECT          e.empno, e.ename, e.sal, e.job
FROM            emp e;
```

INSTEAD-OF Triggers



Object Views (continued)

INSTEAD-OF Triggers: INSTEAD-OF triggers provide a transparent way of modifying views that cannot be modified directly through SQL DML statements (INSERT, UPDATE, and DELETE).

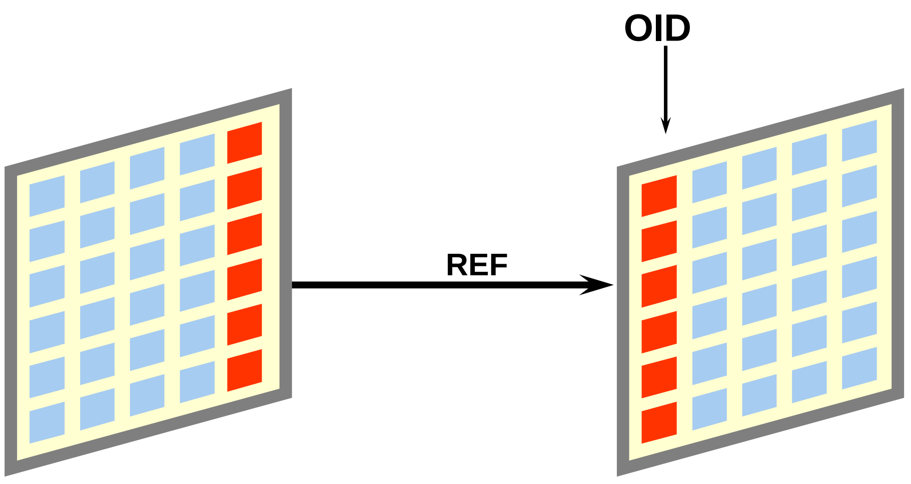
These triggers are called INSTEAD-OF triggers because, unlike other types of triggers, the Oracle server fires the trigger instead of executing the triggering statement. The trigger performs update, insert, or delete operations directly on the underlying tables.

Users write normal INSERT, DELETE, and UPDATE statements against the view, and the INSTEAD-OF trigger works invisibly in the background to make the right actions take place.

INSTEAD-OF triggers are activated for each row.

Note: Although INSTEAD-OF triggers can be used with any view, they are typically needed with object views.

References to Objects



Referencing Objects

Introduction

In relational databases, primary key values are used to uniquely identify records. In object-relational databases, OIDs provide an alternative method.

When a row in an object table or object view is created, it is automatically assigned a unique identifier called an object ID (OID).

Object REFs

With relational tables, you can associate two records by storing the primary key of one record in one of the columns (the foreign key column) of another.

In a similar way, you can associate a row in a relational table to an object by storing the OID of an object in a column of a relational table.

You can associate two objects by storing the OID of one object in an attribute of another. The stored copy of the OID then becomes a pointer, or reference (REF), to the original object. The attribute or column that holds the OID is of datatype REF.

Referencing Objects (continued)

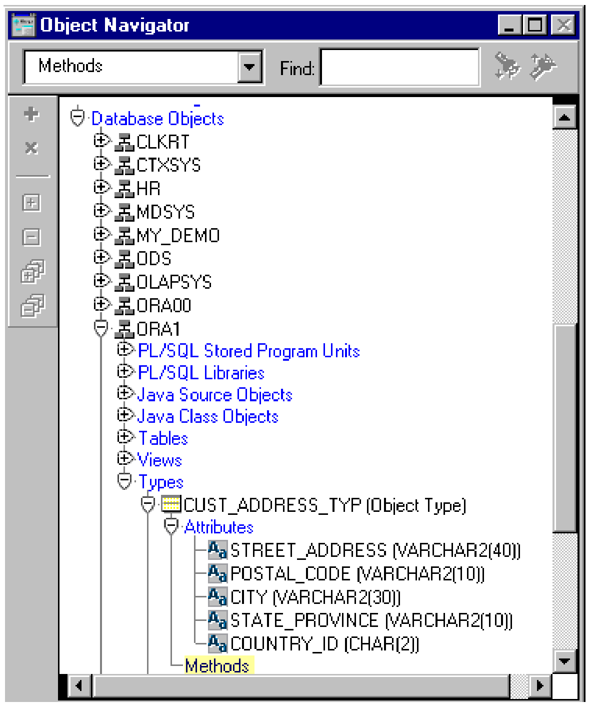
Note: Object columns are not assigned OIDs and cannot be pointed to by a REF.

Here is an example of a table declaration that includes a column with a REF datatype:

```
CREATE TABLE o_emp
     ( empno         NUMBER(4)  NOT NULL,
     ename                VARCHAR2(10),
      job            VARCHAR2(10),
      mgr            NUMBER(4),
      hiredate       DATE,
      sal            NUMBER(7,2),
      comm           NUMBER(7,2),
      dept           REF dept_type SCOPE IS o_dept) ;
```

Note: The REF is scoped here to restrict the reference to a single table, O_DEPT. The object itself is not stored in the table, only the OID value for the object.

Object Types in Object Navigator



Database Objects
CLKRT
CTXSYS
HR
MDSYS
MY_DEMO
ODS
OLAPSYS
ORA00
ORA1
PL/SQL Stored Program Units
PL/SQL Libraries
Java Source Objects
Java Class Objects
Tables
Views
Types
CUST_ADDRESS_TYP (Object Type)
Attributes
STREET_ADDRESS (VARCHAR2(40))
POSTAL_CODE (VARCHAR2(10))
CITY (VARCHAR2(30))
STATE_PROVINCE (VARCHAR2(10))
COUNTRY_ID (CHAR(2))
Methods

Displaying Oracle Objects in the Object Navigator

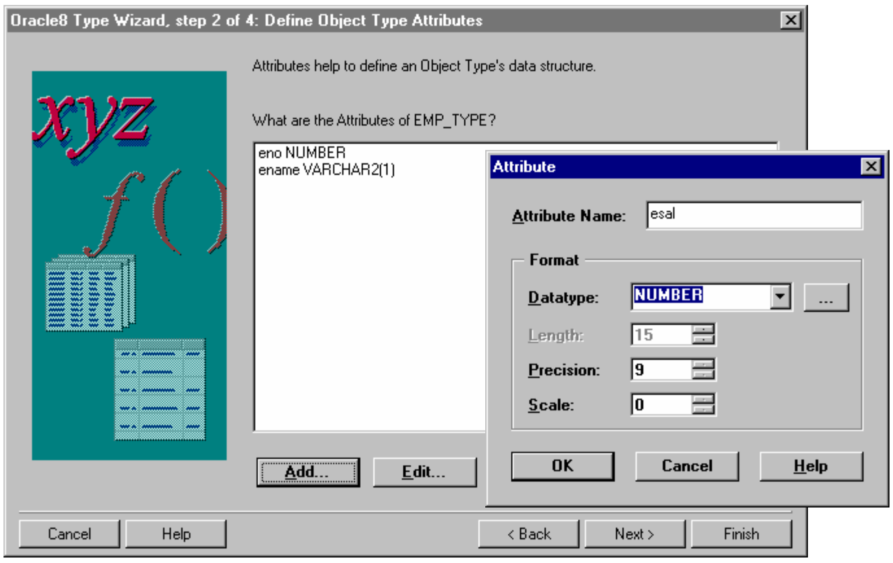
The Object Navigator lists declared types in the “Database Objects” section, along with tables, views, and other Oracle objects.

Object Types

Both the attributes and the methods are listed under each type. Also, the nested types within a type are displayed in an indented sublevel.

This convention is used for nested object and object type displays throughout Oracle Developer.

Object Type Wizard


Oracle8 Type Wizard, step 2 of 4: Define Object Type Attributes

Attributes help to define an Object Type's data structure.

What are the Attributes of EMP_TYPE?

```
eno NUMBER
ename VARCHAR2(1)
```

Attribute

Attribute Name: esal

Format

Datatype: NUMBER

Length: 15

Precision: 9

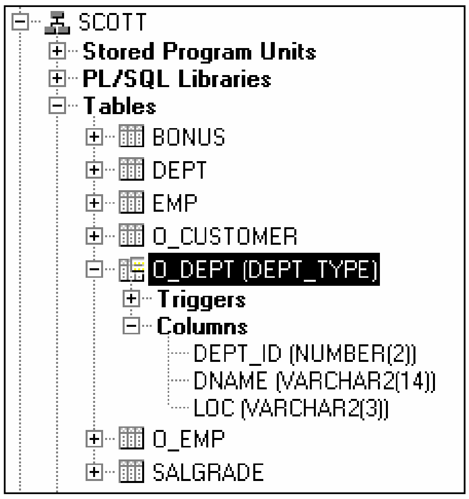
Scale: 0

Oracle8 Type Wizard

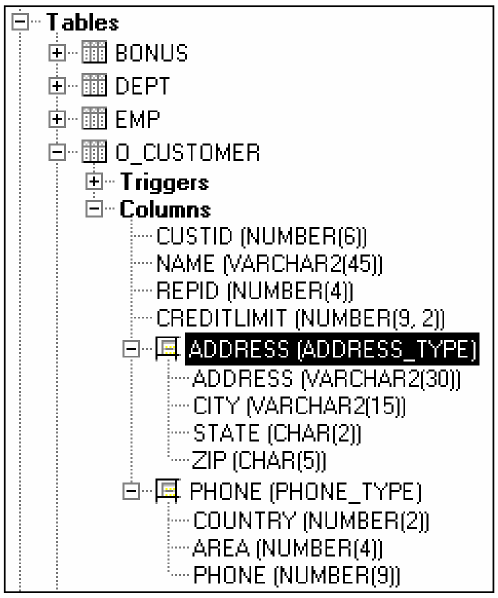
Object types can be created using the Oracle8 Type Wizard. The wizard allows you to define the attributes and methods. You invoke the Oracle8 Type Wizard from Forms Builder by selecting the Types node in a schema under Database Objects, then clicking the Create icon.

Note: The Type Wizard is called Oracle8 Type Wizard, even though it is invoked from Oracle Developer Suite 10*g* Forms Builder connected to an Oracle9*i* database.

Object Tables and Columns in Object Navigator



```
SCOTT
  Stored Program Units
  PL/SQL Libraries
  Tables
    BONUS
    DEPT
    EMP
    O_CUSTOMER
    O_DEPT (DEPT_TYPE)
      Triggers
      Columns
        DEPT_ID (NUMBER(2))
        DNAME (VARCHAR2(14))
        LOC (VARCHAR2(3))
    O_EMP
    SALGRADE
```



```
Tables
  BONUS
  DEPT
  EMP
  O_CUSTOMER
    Triggers
    Columns
      CUSTID (NUMBER(6))
      NAME (VARCHAR2(45))
      REPID (NUMBER(4))
      CREDITLIMIT (NUMBER(9, 2))
      ADDRESS (ADDRESS_TYPE)
        ADDRESS (VARCHAR2(30))
        CITY (VARCHAR2(15))
        STATE (CHAR(2))
        ZIP (CHAR(5))
      PHONE (PHONE_TYPE)
        COUNTRY (NUMBER(2))
        AREA (NUMBER(4))
        PHONE (NUMBER(9))
```

Object Tables and Columns

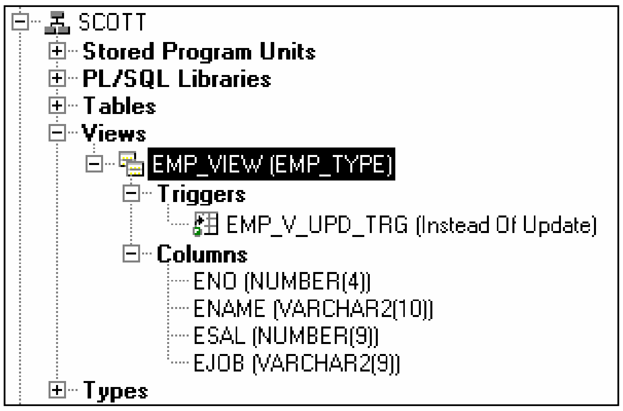
Object Tables

Object tables are displayed like relational tables, with the attributes of the object displayed like columns in a relational table. Also, the object table type name is displayed in parentheses after the name of the object table.

Object Columns

Object columns are displayed with the object type in parentheses after the column name and with the attributes of the type indented underneath the column name.

Object Views in Object Navigator


SCOTT
Stored Program Units
PL/SQL Libraries
Tables
Views
EMP_VIEW (EMP_TYPE)
Triggers
EMP_V_UPD_TRG (Instead Of Update)
Columns
ENO (NUMBER(4))
ENAME (VARCHAR2(10))
ESAL (NUMBER(9))
EJOB (VARCHAR2(9))
Types

E-16

Object Views

Object views are displayed like any other view, except that the object type on which they are based is written in parentheses after the view name.

INSTEAD-OF Trigger Dialog Box

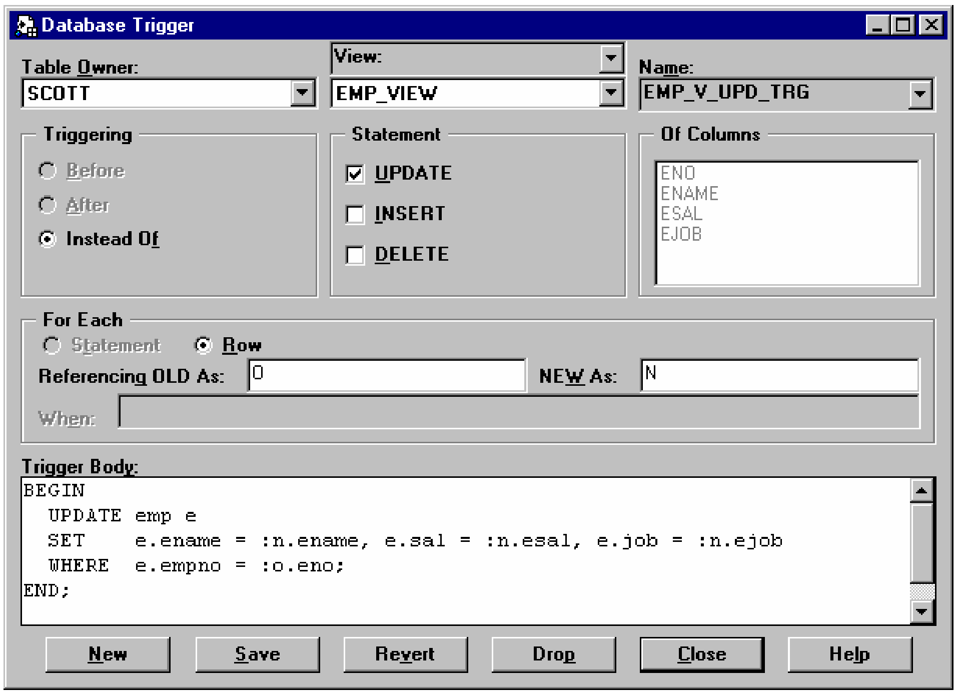


Table Owner: SCOTT
View: EMP_VIEW
Name: EMP_V_UPD_TRG

Triggering: Before / After / [✓ Instead Of]

Statement: [✓ UPDATE] / INSERT / DELETE

Of Columns: ENO, ENAME, ESAL, EJOB

For Each: Statement / [✓ Row]

Referencing OLD As: O NEW As: N

When:

Trigger Body:

```
BEGIN
  UPDATE emp e
  SET    e.ename = :n.ename, e.sal = :n.esal, e.job = :n.ejob
  WHERE  e.empno = :o.eno;
END;
```

Object Views

Object views are displayed like any other view, except that the object type on which they are based is written in parentheses after the view name.

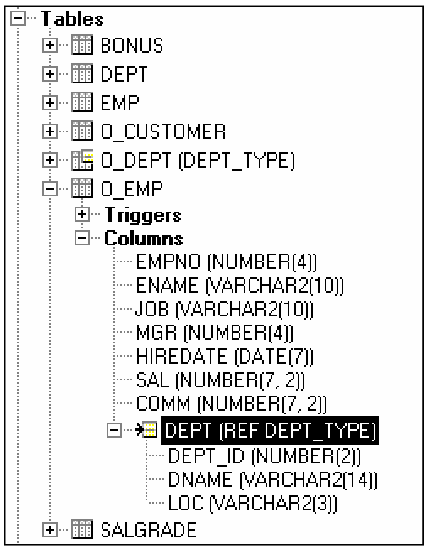
INSTEAD-OF Triggers: INSTEAD-OF database triggers can now be created through the trigger creation dialog box, just like any other database trigger.

INSTEAD-OF INSERT, UPDATE, and DELETE triggers allow you to directly insert, update, and delete against object views. They can also be used with any other type of view that does not allow direct DML.

When a view has an INSTEAD-OF trigger, the code in the trigger is executed in place of the triggering DML code.

Reference: For more information about INSTEAD-OF triggers, see *Oracle Application Developer’s Guide – Fundamentals* or *Oracle Database Concepts*.

Object REFs in Object Navigator



```
Tables
  BONUS
  DEPT
  EMP
  O_CUSTOMER
  O_DEPT (DEPT_TYPE)
  O_EMP
    Triggers
    Columns
      EMPNO (NUMBER(4))
      ENAME (VARCHAR2(10))
      JOB (VARCHAR2(10))
      MGR (NUMBER(4))
      HIREDATE (DATE(7))
      SAL (NUMBER(7, 2))
      COMM (NUMBER(7, 2))
      DEPT (REF DEPT_TYPE)
        DEPT_ID (NUMBER(2))
        DNAME (VARCHAR2(14))
        LOC (VARCHAR2(3))
  SALGRADE
```

E-18

Object REFs

Object types that contain attributes of type REF, and relational tables that have columns of type REF, display the keyword REF before the name of the object type that is being referenced.

The attributes of the referenced object type are displayed indented under the column or attribute.

Summary

- Oracle8 introduced three scalar datatypes.
- Objects and object types allow representation of complex data.
- Three kinds of objects are object tables, object columns, and object views.

E-19

Summary

- INSTEAD-OF triggers allow DML on object views.
- Object REFs store the object identifier of certain types of objects.
- The Object Navigator can display certain types of objects.

Summary

Oracle Datatypes

Oracle8 introduced three scalar datatypes and composite datatypes, such as object types.

Objects

Three kinds of objects are object tables, object columns, and object views.

INSTEAD-OF triggers allow DML on object views.

Object REFs store the object identifier of certain types of objects.

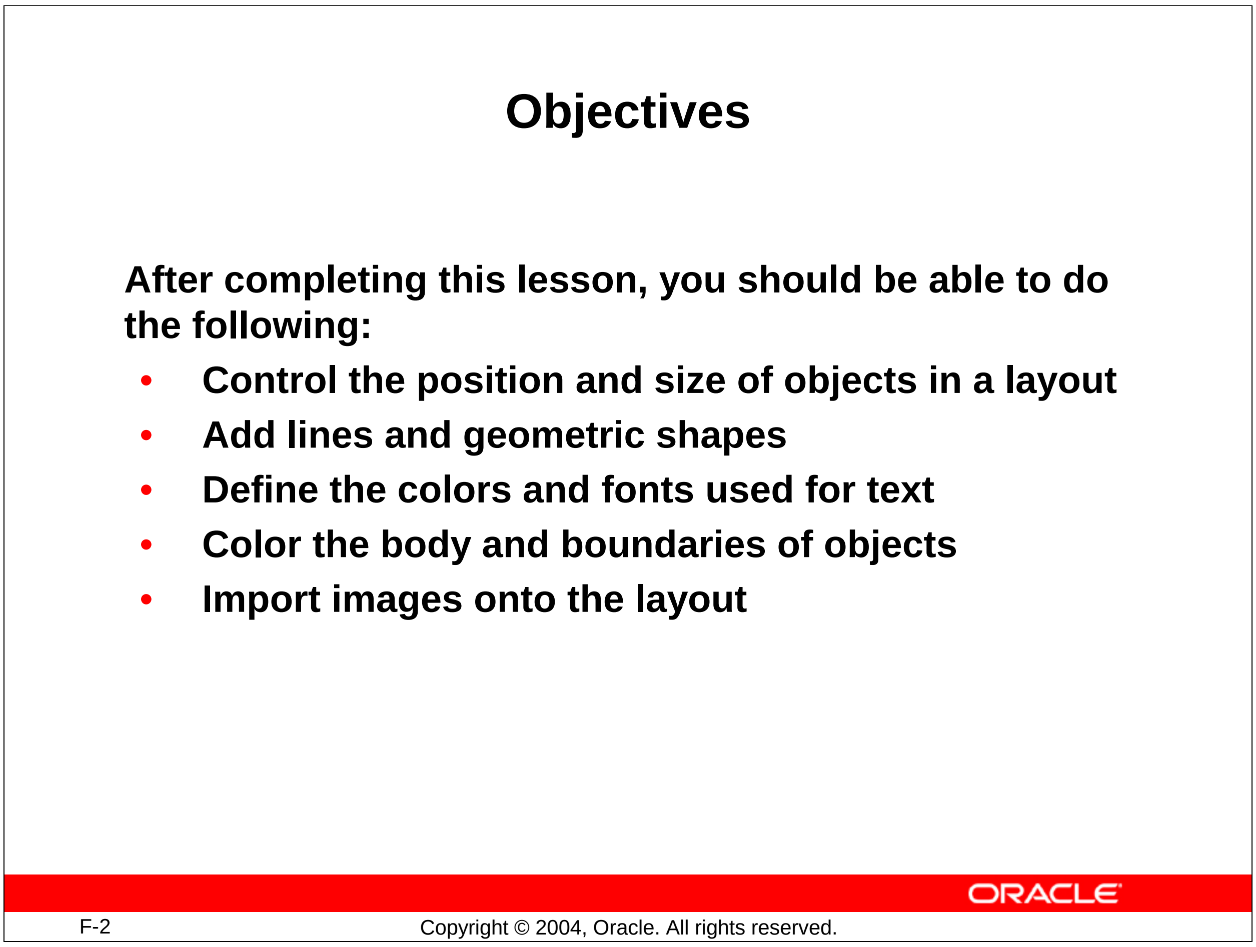
Oracle Objects in the Object Navigator

The Object Navigator can display certain types of objects.

F

Using the Layout Editor

Schedule:	Timing	Topic
	20 minutes	Lecture
	20 minutes	Total



Objectives

After completing this lesson, you should be able to do the following:

- **Control the position and size of objects in a layout**
- **Add lines and geometric shapes**
- **Define the colors and fonts used for text**
- **Color the body and boundaries of objects**
- **Import images onto the layout**

F-2

Overview

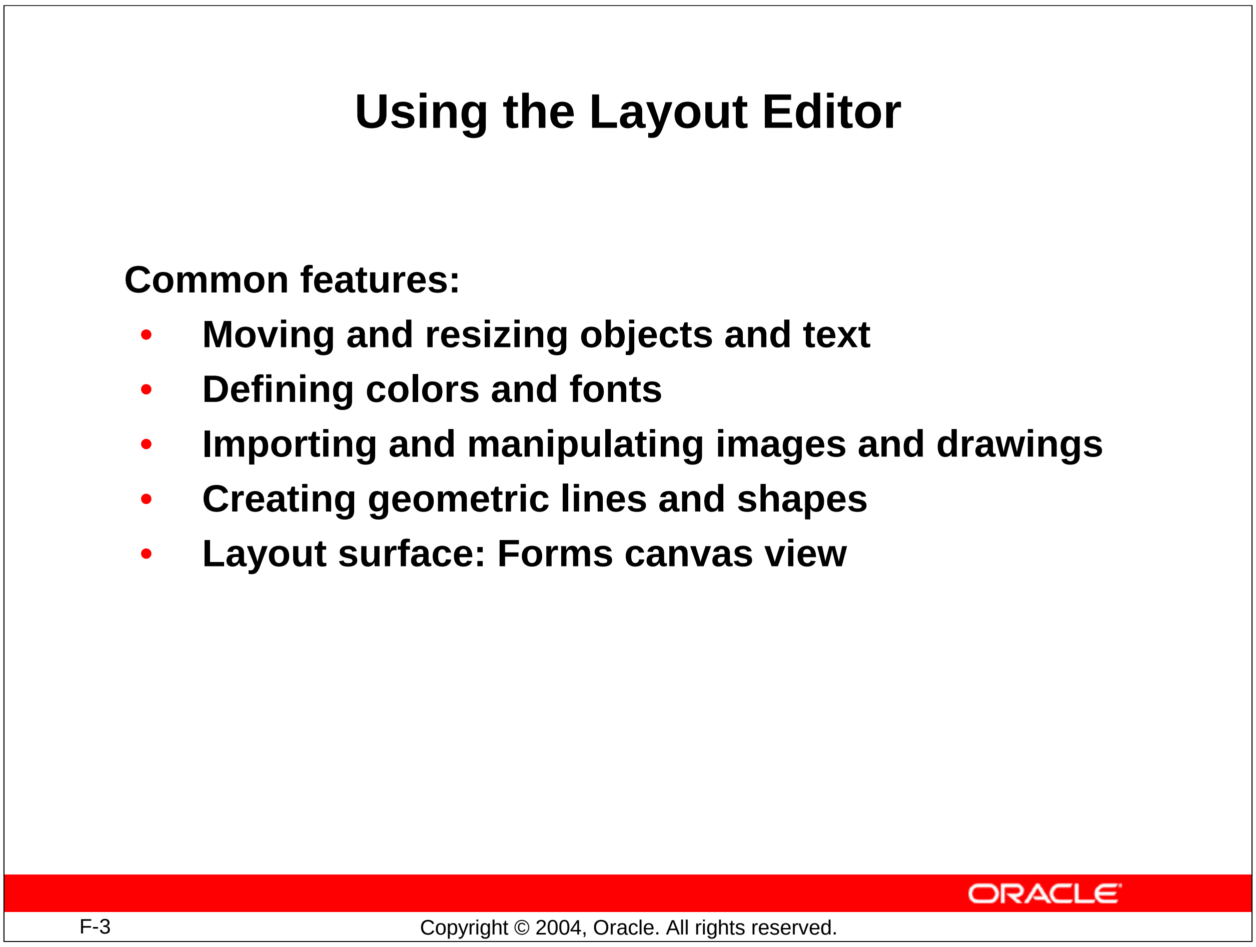
Introduction

In this lesson, you learn the graphical features of the Layout Editor of Oracle Forms Developer. This will help you control the visual arrangement and appearance of objects in your applications.

Objectives

After completing this lesson, you should be able to do the following:

- Control the position and size of objects in a layout
- Add lines and geometric shapes
- Define the colors and fonts used for text
- Color the body and boundaries of objects
- Import images onto the layout



Using the Layout Editor

Common features:

- **Moving and resizing objects and text**
- **Defining colors and fonts**
- **Importing and manipulating images and drawings**
- **Creating geometric lines and shapes**
- **Layout surface: Forms canvas view**

Why Use the Layout Editor?

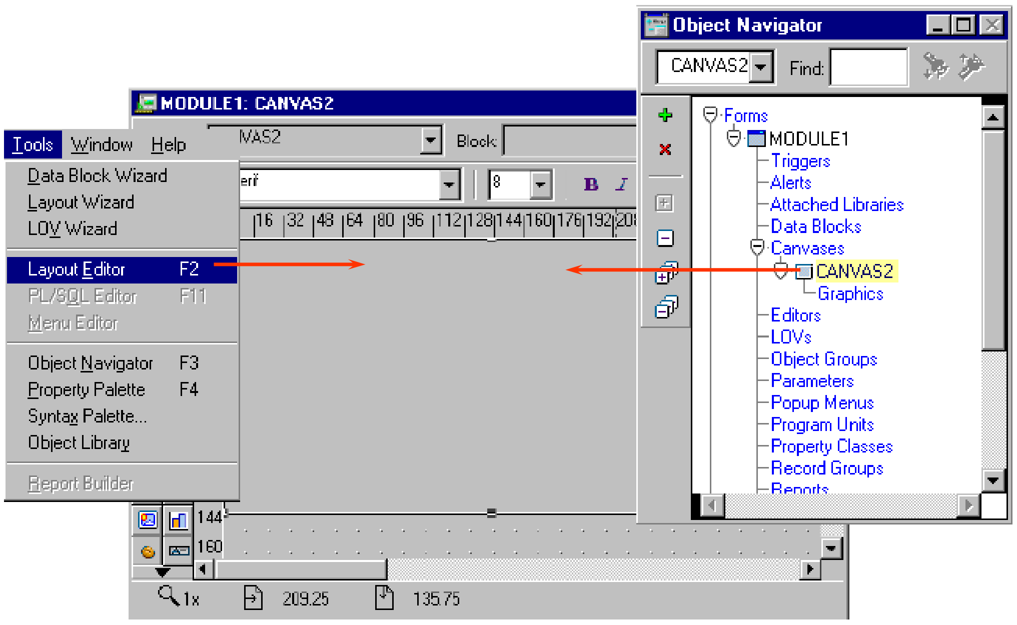
The Layout Editor is a graphical tool for defining the arrangement and appearance of visual objects. The Layout Editor opens windows in Forms Builder to present the surfaces on which you can arrange objects.

The following are functions of the Layout Editor:

- Moving objects to new positions in the layout and aligning them with each other
- Resizing objects and text
- Defining the colors of text and visual objects
- Creating lines, boxes, and other geometric shapes
- Importing and manipulating images on the layout
- Changing the font style and weight of text
- Accessing the properties of objects that you see in the layout

You can use the Layout Editor to control the visual layout on canvas views in Oracle Forms Developer. A canvas is the surface on which you arrange a form's objects. Its view is the portion of that canvas that is initially visible in a window at run time. You can also see stacked or tabbed canvas views in the Layout Editor; their views might overlay others in the same window. You can also display stacked views in the Layout Editor.

Invoking the Layout Editor



How to Access the Layout Editor

You can invoke the Layout Editor from either the Builder menus or the Object Navigator. This applies whether you are doing so for the first time in a session or at a later stage. If you have minimized an existing Layout Editor window, you can also reacquire it in the way that you would any window.

Opening from the Object Navigator

Double-click a canvas view icon in a form hierarchy.

Opening from the Builder Menus

- Make sure that you have a context for a form by selecting the appropriate objects in the Navigator.
- Select Tools > Layout Editor from the Builder menu.

How to Access the Layout Editor (continued)

In Forms, you can open several Layout Editor windows—one for each canvas view. Make sure that you have the canvas that you want.

Closing the Layout Editor

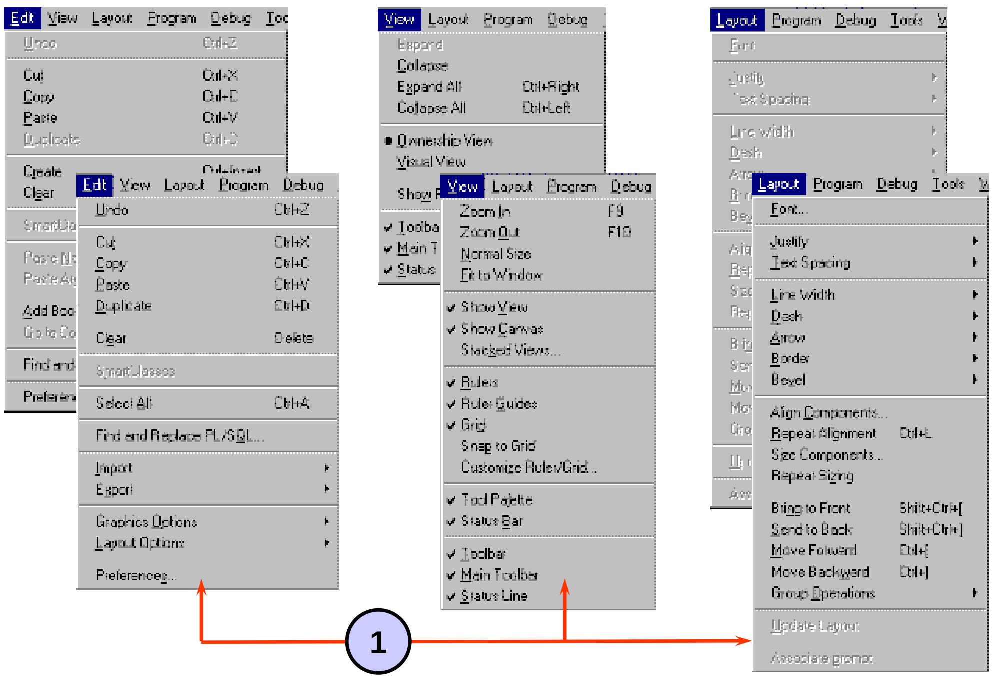
You can close or minimize the Layout Editor window or windows as you would any window.

Instructor Note

Demonstration

- Accessing through the Tools menu
- Access through the Navigator
- Closing and minimizing the Layout Editor window
- Reopening the Layout Editor

Layout Editor: Components



ORACLE

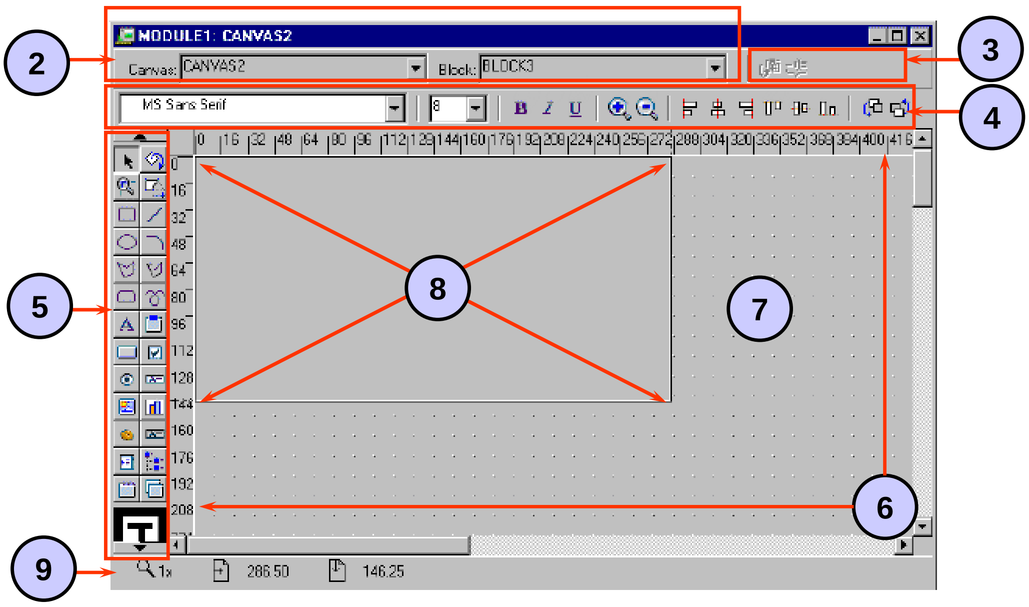
F-6 Copyright © 2004, Oracle. All rights reserved.

Components of the Layout Editor

Common components of the Layout Editor are:

- **Menu facilities:** While the Layout Editor window is active, the options available on the Edit, View, and Layout Builder menus expand. The new choices are submenus for controlling the Layout Editor.
  The slide above shows the Edit, View, and Layout menus. The top screenshots show what these menus look like when the Object Navigator is active, while those on the bottom show the expanded choices when the Layout Editor is active.

Layout Editor: Components



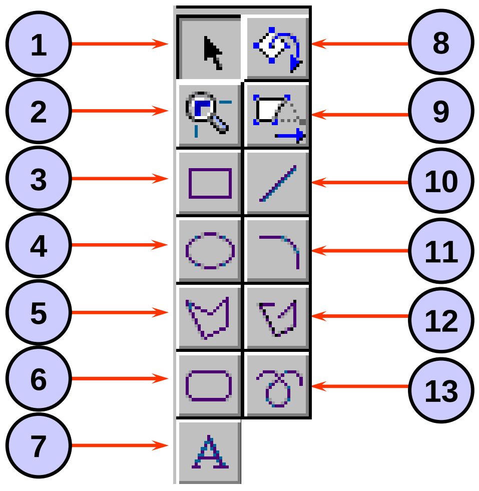
Components of the Layout Editor (continued)

- **Title bar:** Displays the name of the current form, the name of the canvas being edited, and the name of the current block. When you create an item by drawing it on a canvas in the Layout Editor, Forms Builder assigns the item to the current block, as indicated by Layout Editor block context. You can change Layout Editor block context using the Block poplist on the title bar.
- **Horizontal toolbar:** Contains buttons that enable you to update the layout or associate text with an item prompt.
- **Style bar:** Appears under the title bar and horizontal toolbar. The style bar contains buttons that are a subset of the View and Layout menus.
- **Vertical toolbar:** Contains the tools for creating and modifying objects on the layout. Note that some tools in the palette may be hidden if you have reduced the size of the Layout Editor window. If so, scroll buttons appear on the vertical toolbar. There are three types of tools:
  - Graphics tools for creating and modifying lines and shapes
  - Tools for creating specific types of items
  - Manipulation tools for controlling color and patterns

Components of the Layout Editor (continued)

- **Rulers and ruler guides:** Rulers are horizontal and vertical markers to aid alignment; they appear at the top and side of the layout region. You can switch these off or have their units altered, as required. Drag ruler guides from the rulers across the layout region to mark positions in the layout.
- **Layout/Painting region:** The main area where you can place and manipulate objects. A grid pattern can be displayed in this area to aid alignment of objects. You can switch off or rescale this grid if required (the grid is hidden in the canvas portion of the layout/painting region if the View > Show Canvas option is switched on).
- **Canvas:** The portion of the layout/painting area where objects must be placed in order for the form module to successfully compile. If objects are outside this area upon compilation, you will receive an error.
- **Status line:** Appears at the bottom of the window. It shows you the mouse position and drag distance (when moving objects) and the current magnification level.

Tool Palette



Creating and Modifying Objects in the Layout

You can perform actions in the Layout Editor by selecting from the vertical toolbar and the Builder menus and by controlling objects directly in the layout region.

Creating Lines and Shapes

Create geometric lines and shapes by selecting from the graphics tools in the vertical toolbar. These include: Rectangles/squares, ellipses/circles, polygons and polylines, lines and arcs, and the freehand tool.

1	Select	8	Rotate
2	Magnify	9	Reshape
3	Rectangle	10	Line
4	Ellipse	11	Arc
5	Polygon	12	Polyline
6	Rounded rectangle	13	Freehand
7	Text		

Creating and Modifying Objects in the Layout (continued)

Creating a New Line or Shape

1. Select the required graphic tool from the vertical toolbar with a mouse click. This selects the tool for a single operation on the layout (a double-click causes the tool to remain active for subsequent operations).
2. Position the mouse at the start point for the new object in the layout, and then click-and-drag to the required size and shape.
3. Release the mouse button.

Notice that the object remains selected after this procedure (selection handles appear on its boundaries) until you deselect it by clicking elsewhere.

Note: You can produce constrained shapes (for example, a circle instead of an ellipse) by pressing the [Shift] key during step 2.

Creating Text

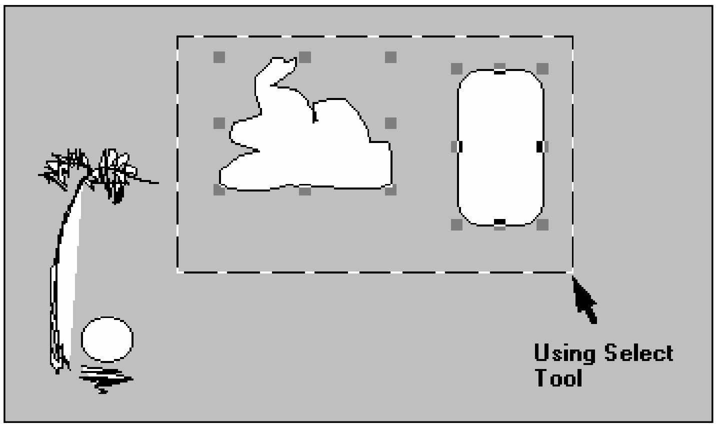
The Text tool (T) lets you open a Boilerplate Text object on the layout. You can type one or more lines of text into this object while it is selected with the Text tool. Uses of text objects vary according to the Oracle Developer tool in which you create them.

Instructor Note

Show in the Layout Editor:

- Creating a line and at least one other shape
- Using the Freehand tool
- Creating a circle or square using Constrained mode
- Creating boilerplate text, with text over multiple lines
- Double-clicking the palette tool to retain selection

Selecting Objects



F-11 Copyright © 2004, Oracle. All rights reserved.

Selecting Objects for Modification

The default tool in the vertical toolbar is Select, with which you can select and move objects. If the Select tool is active, you can select one or more objects on the layout to move or modify.

To select *one* object, do one of the following:

- Click the object with the mouse
- Draw a bounding box around it

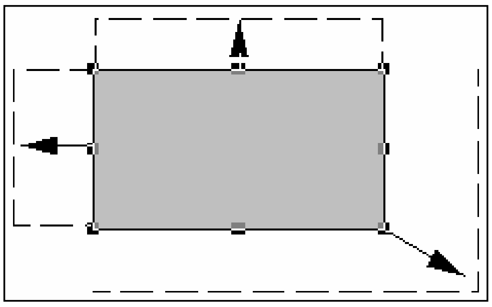
If the object is small or narrow, it is sometimes easier to use the second method. Also, an object may be transparent (No Fill), which can present a similar problem where it has no center region to select.

It is convenient to select *several* objects, so that an operation can be performed on them simultaneously.

To select several objects together, do one of the following:

- Hold down the [Shift] key and then click each object to be selected
- Draw a bounding box around the objects (providing the objects are adjacent to each other)

Manipulating Objects



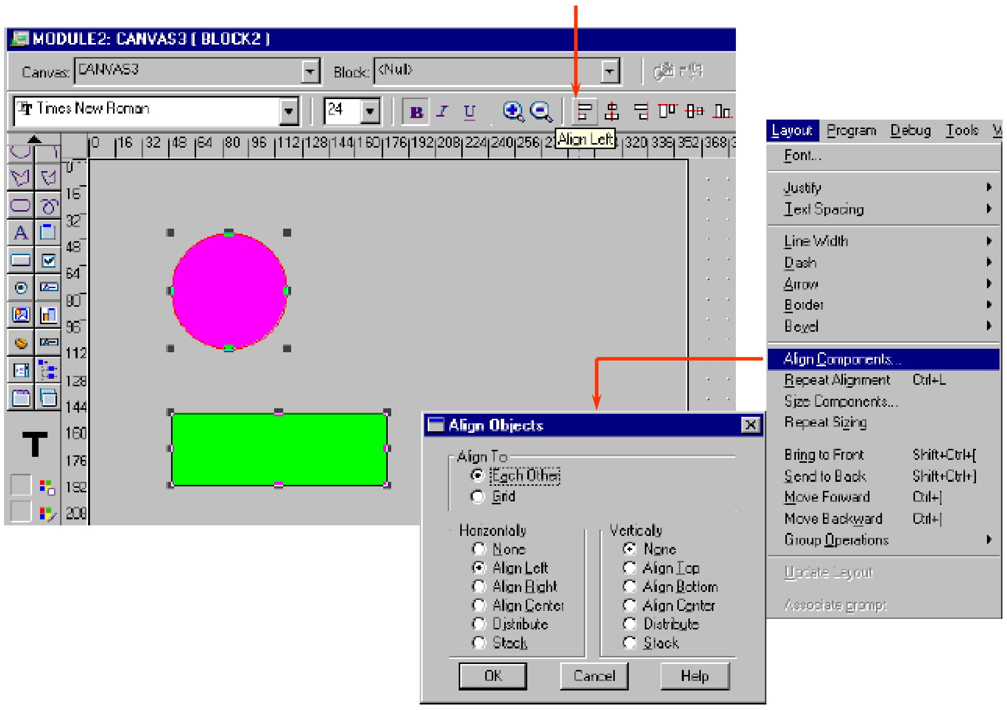
Changing the Size or Ratio

When an object is selected, two types of selection handles are visible:

- Corner handles: Position the mouse on one of these to change the size or ratio of the object diagonally.
- Midpoint handles: Position the mouse on one of these to change the size or ratio in a horizontal or vertical direction.

Note: By holding down the [Shift] key, you can resize an object without changing its ratios. This means that squares remain as squares, and images do not become distorted when resized.

Moving, Aligning, and Overlapping



Moving and Aligning Objects

When one or more objects are selected in the layout, you can:

- **Move them to a new location:** Do this by dragging to the required position and releasing the mouse button. You can use the grid and ruler lines to help you position them properly.
- **Align the objects with each other:** Objects can be aligned with the leftmost, rightmost, highest, or lowest object selected. They can also be centered and aligned with the grid. You can do this using the Alignment feature in the Layout menu: Select Layout > Align Components, and then set the options required in the Align Objects dialog box. You can also use the Align icons in the horizontal toolbar.

A Note on Grid-Snap Alignment: You can ensure that all objects that you move align with snap points that are defined on the grid. To activate these, select View > Snap to Grid from the menu. You can also use the View options to change the grid-snap spacing and units.

Note: If you position an object using one grid-snap spacing, and then try to position other objects under different settings, it may prove difficult to align them with each other. Try to stick to the same snap units, if you use grid-snap at all.

Moving and Aligning Objects (continued)

Overlapping Objects

You can position objects on top of each other. If they are transparent, then one object can be seen through another (this is explained in detail later in this lesson).

Change the stacking order of overlapping objects by selecting the object to move and then choosing the following as required, from the Layout menu:

- Bring to Front
- Send to Back
- Move Forward
- Move Backward

Instructor Note

Create your own objects to show the following:

- Selecting multiple objects by pressing and holding [Shift] while you click the objects
- Selecting by a bounding box
- Losing the association after you deselect
- Expanding and contracting in different directions, pointing out handles
- Moving an object on the layout
- Lining up several objects using the Align Objects dialog box
- Moving objects while grid-snap is on
- How grid and grid-snap can be turned on/off and how units can be altered (pointing out the warning opposite)
- Overlapping some objects and changing layering through the Layout menu

Groups in the Layout

- Groups allow several objects to be repeatedly treated as one.
- Groups can be colored, moved, or resized.
- Tool-specific operations exist for groups.
- Groups have a single set of selection handles.
- Members can be added or removed.

Manipulating Objects As a Group

Sometimes, you want to group objects together in the layout so that they behave as a single object.

Placing Objects into a Group

1. Select the objects in the layout that are to be grouped together.
2. Select Layout > Group Operations > Group from the menu.

You will notice there is now a single set of selection handles for the group, which you can treat as a single object whenever you select a member within it. The Layout menu also gives you options to add and remove members. Resizing, moving, coloring, and other operations will now apply to the group.

Manipulating Individual Group Members

To manipulate group members individually, select the group and then click the individual member. You can use options in the Group Operations menu to remove objects from the group or to reselect the parent group.

Manipulating Objects As a Group (continued)

Other Tools for Manipulating Objects

- **Rotate:** You can rotate a line or shape through an angle, using the tool’s selection handles.
- **Reshape:** You can change size/ratio of a shape that has been rotated or change the sweep angle of an arc. You can also reshape a polygon or polyline.
- **Magnify:** You can increase magnification when you click at a desired zoom position on the layout region. Pressing and holding [Shift] while you click reduces magnification.

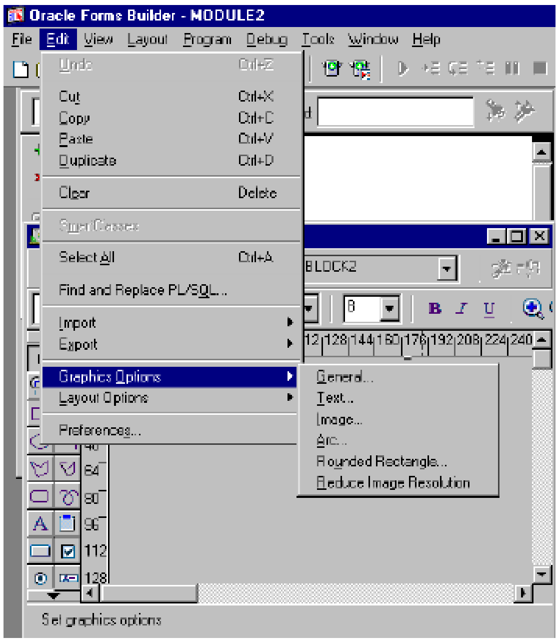
Note: You can undo your previous action in the current Layout Editor session by selecting Edit > Undo from the menu.

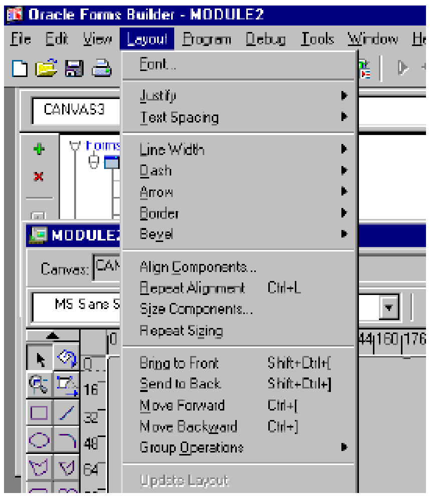
Instructor Note

Create your own objects to do the following:

- Place some objects into a group, and then show how they can be manipulated as one. Point out the single set of handles.
- Point out or mention the other grouping options (Group Operations menu).
- Demonstrate zooming in/out using the Magnify tool.
- Show Undo/Redo of a Layout Editor operation from the Edit menu.

Edit and Layout Menus





ORACLE

Formatting Objects in the Layout

The Builder’s Layout and Edit menus provide a variety of facilities for changing the style and appearance of objects in the layout. These include:

- Font sizes and styles
- Spacing in lines of text
- Alignment of text in a text object
- Line thickness and dashing of lines
- Bevel (3D) effects on objects
- General drawing options (for example, style of curves and corners)

Whichever formatting option you intend to use, first select the objects that you intend to change on the layout, and then choose the necessary option from the menu.

Some of the format options are available from the style bar.

Formatting Objects in the Layout (continued)

Changing Fonts on Textual Objects

There are a number of ways to change the font characteristics of textual objects. The common steps provided by the Layout Editor are:

- Select the objects in the layout whose content text you want to change (these may be boilerplate text objects and other textual object types).
- Select the font style and size that you require from the style bar or select Layout from the Builder menu, and the select the font style and size that you require.

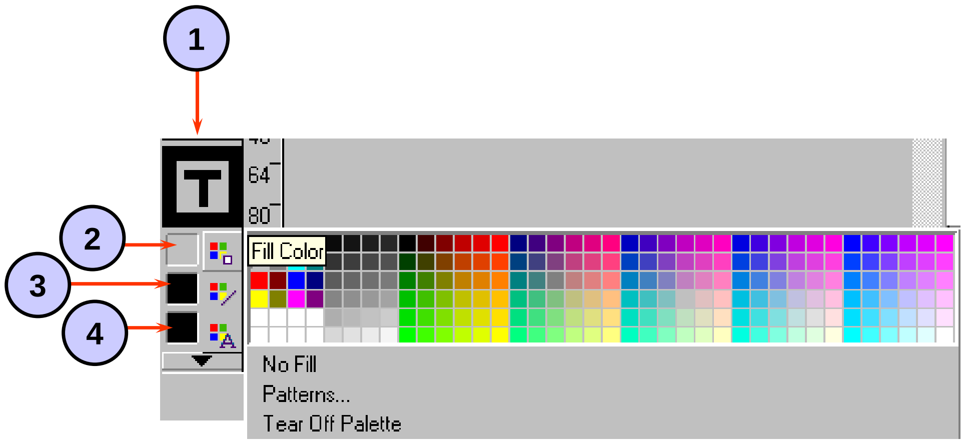
Note: In Microsoft Windows, selecting Font from the menu opens the standard Windows Font dialog. In other GUI environments, the font choices may appear in the Layout menu itself. Font settings are ignored if the application is run in character mode.

Instructor Note

Provide an overview of the Layout and Edit menu options and demonstrate:

- Changing the line thickness of a shape
- Producing bevel effects on this shape
- Changing the font size and style of boilerplate text

Color and Pattern Tools



Coloring Objects and Text

The vertical toolbar contains tools for coloring objects. These are:

1. **Sample Window:** Shows the fill, line, and text color of the currently selected object. If no object is selected, shows the settings for new objects.
2. **Fill Color:** Use this tool to define the colors and pattern for an object's body.
3. **Line Color:** Use this tool to define the color of a line or the boundary line around an object.
4. **Text Color:** Use this tool to choose the color for the text.

Coloring Objects

You can separately color the fill area (body) of an object and its boundary line (a line object has no body).

Coloring a Line

- With the desired layout objects selected, click the Line Color tool. The color palette opens.
- Select the color for lines and bounding lines from this color palette. Select No Line at the bottom of this window to remove the objects' boundary lines.

Note: If you set both No Fill and No Line, the affected objects become invisible.

Coloring Objects and Text (continued)

Coloring an Area

1. Select the objects whose color you want to change.
2. Select the Fill Color tool. The color palette appears.
   If you want the objects to become transparent, select No Fill at the bottom of the color palette window.
3. Select a color from the color palette. If you want the fill area to be patterned instead of plain, then select Patterns from the bottom of the color palette, and follow the next step.
4. If you select Patterns, another window appears with patterns for you to choose. Select one. You can define separate colors for the two shades of the pattern by selecting the pattern color buttons at the bottom of the Fill Pattern palette; each button opens a further color palette.

Coloring Text

1. Select the textual objects whose color you want to change in the layout.
2. Select the Text Color tool. The color palette appears, showing the available colors.
3. Select a color from the color palette (click the appropriate square).
4. Notice that the selected objects on the layout have adopted the chosen text color. Also, the sample area in the vertical toolbar shows a T with the selected color, and this color appears next to the Text Color tool. This indicates the current text color setting.

Altering the Color Palette

By selecting Edit > Layout Options > Color Palette from the menu, you can edit the color palette that is presented when selecting colors. This option is available only when the Builder option Color Palette Mode is set to Editable in the Preferences dialog (Edit > Preferences). Changes to the color palette are saved with the current module.

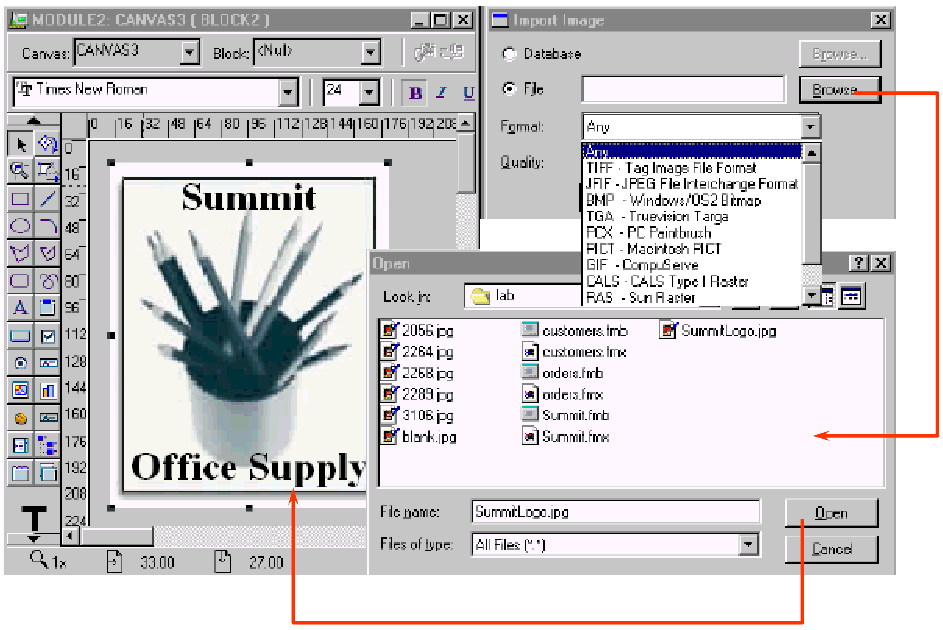
Note: Modifications to the color palette will not be apparent until you close the document and reopen it.

Instructor Note

Show the following:

- Coloring the body of a shape
- Changing the fill pattern and how elements of the pattern can be colored
- Making an object transparent and how other objects can then be seen through it
- Coloring lines and removing them completely
- Coloring text
- How color palettes can be torn off and repositioned
- How color settings are retained in the Layout Editor
- How to modify the color palette

Importing Images



Importing Images

Oracle Forms Developer allows both static and dynamic bit-mapped images to be integrated into the application. This section discusses how you can import static images—that is, those that are fixed on the layout. These might include:

- Company logos
- Photographs for inclusion in a report or display
- Background artwork

You can import images onto the layout from either the file system or the database. Select Edit > Import > Image from the menus, and set the Import Image options:

- **File/Database radio buttons:** Specify location from which to import
- **Filename:** File to be imported (click Browse to locate file)
- **Format:** Image format; choices are TIFF , JFIF (JPEG), BMP, TGA, PCX, PICT, GIF, CALS, RAS
- **Quality:** Range from Excellent to Poor. There is a trade-off between image resolution and memory required.

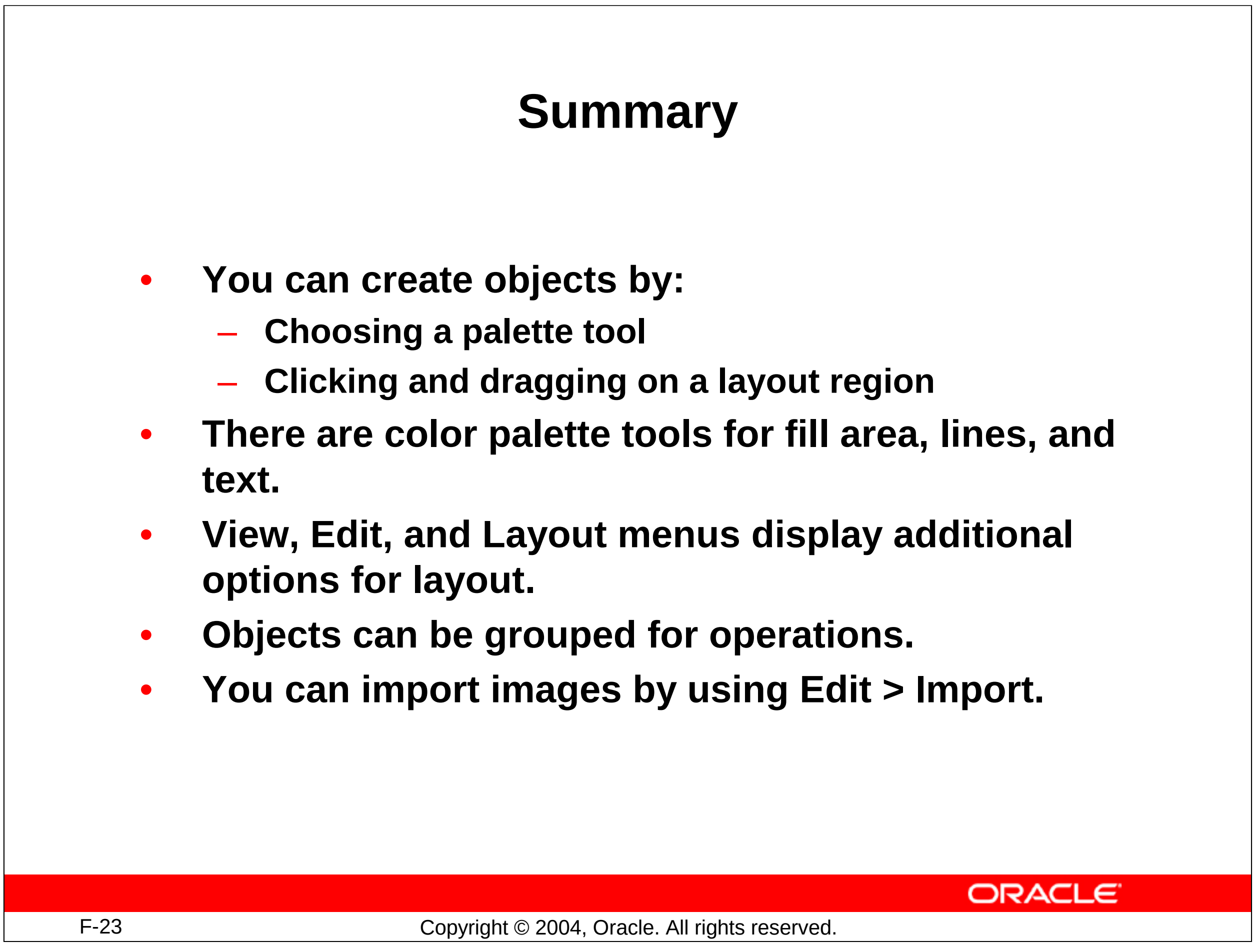
Importing Images (continued)

Manipulating the Imported Image

When the imported image appears on the layout, you can move it or resize it like other objects on the layout. It is usually better to resize images in Constrained mode (while pressing the [Shift] key) so the image does not become distorted.

Instructor Note

Show how to import an image from the file system, pointing out the various file formats that are supported. Recommend using `SummitLogo.jpg`.
Show that boilerplate images can be moved and resized on the layout.



Summary

- You can create objects by:
  - Choosing a palette tool
  - Clicking and dragging on a layout region
- There are color palette tools for fill area, lines, and text.
- View, Edit, and Layout menus display additional options for layout.
- Objects can be grouped for operations.
- You can import images by using Edit > Import.

F-23

Summary

- You can create objects by:
  - Choosing the correct toolbar tool
  - Clicking and dragging the layout region
- There are color palette tools for fill area, lines, and text.
- View, Edit, and Layout menus display additional options while you are in the Layout Editor.
- You can group objects for operations.
- You can import images by selecting Edit > Import.