

# **Oracle Forms Developer 10g: Build Internet Applications**

**Volume II • Student Guide**

D17251GC20

Edition 2.0

September 2006

D47400

**ORACLE®**

<b>Author</b>	<b>Copyright © 2006, Oracle. All rights reserved.</b>
Pam Gamer	
<b>Technical Contributors and Reviewers</b>	
Laurent Dereac	
Sujatha Kalastriraju	
Manish Pawar	
Bryan Roberts	
Raza Siddiqui	
Lex van der Werff	
<b>Editors</b>	
Aju Kumar	
Nita Pavitran	
<b>Graphic Designer</b>	
Steve Elwood	
<b>Publisher</b>	
Srividya Rameshkumar	
<b>Disclaimer</b>	
This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.	
The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.	
<b>Restricted Rights Notice</b>	
If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:	
<b>U.S. GOVERNMENT RIGHTS</b>	
The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.	
<b>Trademark Notice</b>	
Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.	

Oracle Internal & OAI Use Only

# Contents

## Preface

### I Introduction

Objectives I-2

Course Objectives I-3

Course Content I-5

### 1 Introduction to Oracle Forms Developer and Oracle Forms Services

Objectives 1-2

Internet Computing Solutions 1-3

Plugging into the Grid 1-4

Oracle Enterprise Grid Computing 1-5

Oracle 10g Products and Forms Development 1-7

Oracle Application Server 10g Architecture 1-8

Oracle Application Server 10g Components 1-9

Oracle Forms Services: Overview 1-10

Forms Services Architecture 1-11

Benefits and Components of Oracle Developer Suite 10g 1-12

Oracle Developer Suite 10g Application Development 1-13

Oracle Developer Suite 10g Business Intelligence 1-14

Oracle Forms Developer: Overview 1-15

Oracle Forms Developer 10g: Key Features 1-16

Summit Office Supply Schema 1-17

Summit Application 1-18

Summary 1-20

### 2 Running a Forms Developer Application

Objectives 2-2

Testing a Form: OC4J Overview 2-3

Testing a Form: Starting OC4J 2-4

Running a Form 2-5

Running a Form: Browser 2-6

Java Runtime Environment 2-7

Starting a Run-Time Session 2-8

Forms Servlet 2-11

Forms Client 2-12  
Forms Listener Servlet 2-13  
Forms Runtime Engine 2-14  
What You See at Run Time 2-15  
Identifying the Data Elements 2-17  
Navigating a Forms Developer Application 2-18  
Modes of Operation: Enter-Query Mode 2-20  
Modes of Operation: Normal Mode 2-21  
Retrieving Data 2-22  
Retrieving Restricted Data 2-23  
Using the Query/Where Dialog Box 2-25  
Inserting, Updating, and Deleting 2-27  
Making Changes Permanent 2-29  
Displaying Errors 2-30  
Summary 2-31  
Practice 2: Overview 2-34

### **3 Working in the Forms Developer Environment**

Objectives 3-2  
Forms Developer Executables 3-3  
Forms Developer Module Types 3-5  
Forms Builder: Key Features 3-7  
Forms Builder Components: Object Navigator 3-8  
Forms Builder Components: Layout Editor 3-10  
Getting Started in the Forms Builder Interface 3-12  
Forms Builder: Menu Structure 3-14  
Blocks, Items, and Canvases 3-16  
Navigation in a Block 3-18  
Data Blocks 3-19  
Forms and Data Blocks 3-21  
Form Module Hierarchy 3-23  
Customizing Your Forms Builder Session 3-25  
Saving Preferences 3-27  
Using the Online Help System 3-28  
Defining Forms Environment Variables for Design Time 3-29  
Defining Forms Environment Variables for Run Time 3-31  
Environment Variables and Y2K Compliance 3-32  
Forms Files to Define Run-Time Environment Variables 3-34  
Testing a Form: The Run Form Button 3-35  
Summary 3-37  
Practice 3: Overview 3-39

#### **4 Creating a Basic Form Module**

Objectives 4-2  
Creating a New Form Module 4-3  
Form Module Properties 4-6  
Creating a New Data Block 4-8  
Navigating the Wizards 4-10  
Launching the Data Block Wizard 4-11  
Data Block Wizard: Type Page 4-12  
Data Block Wizard: Table Page 4-13  
Launching the Layout Wizard 4-15  
Layout Wizard: Items Page 4-16  
Layout Wizard: Style Page 4-17  
Layout Wizard: Rows Page 4-18  
Data Block Functionality 4-19  
Template Forms 4-20  
Saving a Form Module 4-21  
Compiling a Form Module 4-22  
Module Types and Storage Formats 4-23  
Deploying a Form Module 4-25  
Text Files and Documentation 4-26  
Summary 4-27  
Practice 4: Overview 4-28

#### **5 Creating a Master-Detail Form**

Objectives 5-2  
Form Block Relationships 5-3  
Data Block Wizard: Master-Detail Page 5-5  
Relation Object 5-7  
Creating a Relation Manually 5-8  
Join Condition 5-9  
Deletion Properties 5-10  
Modifying a Relation 5-11  
Coordination Properties 5-12  
Running a Master-Detail Form Module 5-13  
Modifying the Structure of a Data Block 5-14  
Modifying the Layout of a Data Block 5-15  
Summary 5-17  
Practice 5: Overview 5-18

#### **6 Working with Data Blocks and Frames**

Objectives 6-2

Managing Object Properties 6-3  
Displaying the Property Palette 6-4  
Property Palette: Features 6-5  
Property Controls 6-6  
Visual Attributes 6-8  
How to Use Visual Attributes 6-9  
Font, Pattern, and Color Pickers 6-10  
Controlling Data Block Behavior and Appearance 6-11  
Navigation Properties 6-12  
Records Properties 6-13  
Database Properties 6-15  
Scroll Bar Properties 6-18  
Controlling Frame Properties 6-19  
Displaying Multiple Property Palettes 6-21  
Setting Properties on Multiple Objects 6-22  
Copying Properties 6-24  
Creating a Control Block 6-26  
Deleting a Data Block 6-27  
Summary 6-28  
Practice 6: Overview 6-29

## 7 Working with Text Items

Objectives 7-2  
Text Item: Overview 7-3  
Creating a Text Item 7-4  
Modifying the Appearance of a Text Item: General and Physical Properties 7-6  
Modifying the Appearance of a Text Item: Records Properties 7-7  
Modifying the Appearance of a Text Item: Font and Color Properties 7-8  
Modifying the Appearance of a Text Item: Prompts 7-9  
Associating Text with an Item Prompt 7-10  
Controlling the Data of a Text Item 7-11  
Controlling the Data of a Text Item: Format 7-12  
Controlling the Data of a Text Item: Values 7-13  
Controlling the Data of a Text Item: Copy Value from Item 7-15  
Controlling the Data of a Text Item: Synchronize with Item 7-16  
Altering Navigational Behavior of Text Items 7-17  
Enhancing the Relationship Between Text Item and Database 7-18  
Adding Functionality to a Text Item 7-19  
Adding Functionality to a Text Item: Conceal Data Property 7-20  
Adding Functionality to a Text Item: Keyboard Navigable and Enabled 7-21  
Adding Functionality to a Text Item: Multi-line Text Items 7-22

Displaying Helpful Messages: Help Properties	7-23
Summary	7-24
Practice 7: Overview	7-26
<b>8 Creating LOVs and Editors</b>	
Objectives	8-2
Overview of LOVs and Editors	8-3
LOVs and Record Groups	8-6
Creating an LOV Manually	8-8
Creating an LOV with the LOV Wizard: SQL Query Page	8-9
Creating an LOV with the LOV Wizard: Column Selection Page	8-10
Creating an LOV with the LOV Wizard: Column Properties Page	8-11
Creating an LOV with the LOV Wizard: Display Page	8-12
Creating an LOV with the LOV Wizard: Advanced Properties Page	8-13
Creating an LOV with the LOV Wizard: Assign to Item Page	8-14
LOV Properties	8-15
Setting LOV Properties	8-16
LOVs: Column Mapping	8-17
Defining an Editor	8-19
Setting Editor Properties	8-20
Associating an Editor with a Text Item	8-21
Summary	8-22
Practice 8: Overview	8-23
<b>9 Creating Additional Input Items</b>	
Objectives	9-2
Input Items: Overview	9-3
Check Boxes: Overview	9-4
Creating a Check Box	9-5
Converting Existing Item to Check Box	9-6
Creating a Check Box in the Layout Editor	9-7
Setting Check Box Properties	9-8
Check Box Mapping of Other Values	9-10
List Items: Overview	9-11
Creating a List Item	9-13
Converting Existing Item to List Item	9-14
Creating a List Item in the Layout Editor	9-15
Setting List Item Properties	9-16
List Item Mapping of Other Values	9-17
Radio Groups: Overview	9-18
Creating a Radio Group	9-19

Converting Existing Item to Radio Group 9-20  
Creating Radio Group in the Layout Editor 9-21  
Setting Radio Properties 9-22  
Radio Group Mapping of Other Values 9-23  
Summary 9-24  
Practice 9: Overview 9-25

## **10 Creating Noninput Items**

Objectives 10-2  
Noninput Items: Overview 10-3  
Display Items 10-4  
Creating a Display Item 10-5  
Image Items 10-6  
Image File Formats 10-8  
Creating an Image Item 10-9  
Setting Image-Specific Item Properties 10-10  
Push Buttons 10-12  
Push Button Actions 10-13  
Creating a Push Button 10-14  
Setting Push Button Properties 10-15  
Calculated Items 10-16  
Creating a Calculated Item by Setting Properties 10-17  
Setting Item Properties for the Calculated Item 10-18  
Summary Functions 10-19  
Calculated Item Based on a Formula 10-20  
Rules for Calculated Item Formulas 10-21  
Calculated Item Based on a Summary 10-22  
Rules for Summary Items 10-23  
Creating a Hierarchical Tree Item 10-24  
Setting Hierarchical Tree Item Properties 10-25  
Bean Area Items 10-26  
Creating a Bean Area Item 10-27  
Setting Bean Area Item Properties 10-28  
JavaBean at Run Time 10-29  
Summary 10-30  
Practice 10: Overview 10-32

## **11 Creating Windows and Content Canvases**

Objectives 11-2  
Windows and Canvases 11-3  
Window, Canvas, and Viewport 11-4

Content Canvas	11-5
Relationship Between Windows and Content Canvases	11-6
Default Window	11-7
Displaying a Form Module in Multiple Windows	11-8
Creating a New Window	11-9
Setting Window Properties	11-10
GUI Hints	11-11
Displaying a Form Module on Multiple Layouts	11-12
Creating a New Content Canvas	11-13
Setting Content Canvas Properties	11-15
Summary	11-16
Practice 11: Overview	11-17

## **12 Working with Other Canvas Types**

Objectives	12-2
Overview of Canvas Types	12-3
Stacked Canvas	12-4
Creating a Stacked Canvas	12-6
Setting Stacked Canvas Properties	12-8
Toolbar Canvas	12-9
MDI Toolbar	12-10
Creating a Toolbar Canvas	12-11
Setting Toolbar Properties	12-12
Tab Canvas	12-13
Creating a Tab Canvas	12-14
Creating a Tab Canvas in the Object Navigator	12-15
Creating a Tab Canvas in the Layout Editor	12-16
Setting Tab Canvas, Tab Page, and Item Properties	12-17
Placing Items on a Tab Canvas	12-18
Summary	12-19
Practice 12: Overview	12-21

## **13 Introduction to Triggers**

Objectives	13-2
Trigger: Overview	13-3
Grouping Triggers into Categories	13-4
Defining Trigger Components	13-6
Trigger Type	13-7
Trigger Code	13-9
Trigger Scope	13-10

Specifying Execution Hierarchy 13-12  
Summary 13-14

## **14 Producing Triggers**

Objectives 14-2  
Creating Triggers in Forms Builder 14-3  
Creating a Trigger 14-4  
Setting Trigger Properties 14-7  
PL/SQL Editor: Features 14-8  
Database Trigger Editor 14-10  
Writing Trigger Code 14-11  
Using Variables in Triggers 14-13  
Forms Builder Variables 14-14  
Adding Functionality with Built-in Subprograms 14-16  
Limits of Use 14-18  
Using Built-in Definitions 14-19  
Useful Built-Ins 14-21  
Using Triggers: When-Button-Pressed Trigger 14-23  
Using Triggers: When-Window-Closed Trigger 14-24  
Summary 14-25  
Practice 14: Overview 14-27

## **15 Debugging Triggers**

Objectives 15-2  
Debugging Process 15-3  
Debug Console 15-4  
Debug Console: Stack Panel 15-5  
Debug Console: Variables Panel 15-6  
Debug Console: Watch Panel 15-7  
Debug Console: Form Values Panel 15-8  
Debug Console: PL/SQL Packages Panel 15-9  
Debug Console: Global/System Variables Panel 15-10  
Debug Console: Breakpoints Panel 15-11  
Debug Console 15-12  
Setting Breakpoints in Client Code 15-13  
Setting Breakpoints in Stored Code 15-14  
Debugging Tips 15-15  
Running a Form in Debug Mode 15-16  
Remotely Debugging a Running Form 15-17  
Stepping Through Code 15-19  
Debug: Example 15-20

Summary 15-22  
Practice 15: Overview 15-23

## **16 Adding Functionality to Items**

Objectives 16-2  
Item Interaction Triggers 16-3  
Coding Item Interaction Triggers 16-5  
Interacting with Check Boxes 16-7  
Changing List Items at Run Time 16-8  
Using Dynamic LOVs 16-9  
Displaying LOVs from Buttons 16-10  
LOVs and Buttons 16-12  
Populating Image Items 16-14  
Loading the Right Image 16-16  
Displaying Hierarchical Trees 16-17  
Populating Hierarchical Trees with Queries 16-19  
Populating Hierarchical Trees with Record Groups 16-20  
Queries to Display Hierarchical Trees 16-22  
Interacting with JavaBeans 16-23  
Summary 16-29  
Practice 16: Overview 16-31

## **17 Run-Time Messages and Alerts**

Objectives 17-2  
Run-Time Messages and Alerts: Overview 17-3  
Detecting Run-Time Errors 17-5  
Errors and Built-Ins 17-7  
Message Severity Levels 17-9  
Suppressing Messages 17-11  
`FORM_TRIGGER_FAILURE` Exception 17-13  
Triggers for Intercepting System Messages 17-15  
Handling Informative Messages 17-17  
Setting Alert Properties 17-19  
Planning Alerts 17-21  
Controlling Alerts 17-22  
`SHOW_ALERT` Function 17-24  
Directing Errors to an Alert 17-26  
Causes of Oracle Server Errors 17-27  
Trapping Server Errors 17-29  
Summary 17-30  
Practice 17: Overview 17-33

## **18 Query Triggers**

Objectives 18-2  
Query Processing: Overview 18-3  
SELECT Statements Issued During Query Processing 18-5  
WHERE Clause 18-7  
ONETIME\_WHERE Property 18-8  
ORDER BY Clause 18-9  
Writing Query Triggers: Pre-Query Trigger 18-10  
Writing Query Triggers: Post-Query Trigger 18-11  
Writing Query Triggers: Using SELECT Statements in Triggers 18-12  
Query Array Processing 18-13  
Coding Triggers for Enter-Query Mode 18-15  
Overriding Default Query Processing 18-19  
Obtaining Query Information at Run Time 18-22  
Summary 18-25  
Practice 18: Overview 18-27

## **19 Validation**

Objectives 19-2  
Validation Process 19-3  
Controlling Validation Using Properties: Validation Unit 19-5  
Controlling Validation Using Properties: Validate from List 19-7  
Controlling Validation by Using Triggers 19-9  
Example: Validating User Input 19-11  
Using Client-Side Validation 19-13  
Tracking Validation Status 19-16  
Controlling When Validation Occurs with Built-Ins 19-18  
Summary 19-20  
Practice 19: Overview 19-22

## **20 Navigation**

Objectives 20-2  
Navigation: Overview 20-3  
Internal Navigation 20-5  
Using Object Properties to Control Navigation 20-7  
Mouse Navigate Property 20-9  
Writing Navigation Triggers 20-10  
Navigation Triggers 20-11  
When-New-<object>-Instance Triggers 20-12

SET_<object>_PROPERTY: Examples	20-13
Pre- and Post-Triggers	20-15
Post-Block Trigger: Example	20-17
Navigation Trap	20-18
Using Navigation Built-ins in Triggers	20-19
Summary	20-21
Practice 20: Overview	20-23

## **21 Transaction Processing**

Objectives	21-2
Transaction Processing: Overview	21-3
Commit Sequence of Events	21-6
Characteristics of Commit Triggers	21-8
Common Uses for Commit Triggers	21-10
Life of an Update	21-12
Delete Validation	21-14
Assigning Sequence Numbers	21-16
Keeping an Audit Trail	21-18
Testing the Results of Trigger DML	21-19
DML Statements Issued During Commit Processing	21-21
Overriding Default Transaction Processing	21-23
Running Against Data Sources Other than Oracle	21-25
Getting and Setting the Commit Status	21-27
Array DML	21-31
Effect of Array DML on Transactional Triggers	21-32
Implementing Array DML	21-33
Summary	21-34
Practice 21: Overview	21-38

## **22 Writing Flexible Code**

Objectives	22-2
What Is Flexible Code?	22-3
Using System Variables for Current Context	22-4
System Status Variables	22-6
GET_<object>_PROPERTY Built-Ins	22-7
SET_<object>_PROPERTY Built-Ins	22-9
Referencing Objects by Internal ID	22-11
FIND_ Built-Ins	22-12
Using Object IDs	22-13
Increasing the Scope of Object IDs	22-15

Referencing Items Indirectly 22-17

Summary 22-20

Practice 22: Overview 22-22

## **23 Sharing Objects and Code**

Objectives 23-2

Benefits of Reusable Objects and Code 23-3

Copying and Subclassing Objects and Code 23-5

Subclassing 23-6

What Are Property Classes? 23-8

Creating a Property Class 23-9

Inheriting from a Property Class 23-11

What Are Object Groups? 23-13

Creating and Using Object Groups 23-14

What Are Object Libraries? 23-16

Benefits of the Object Library 23-18

Working with Object Libraries 23-19

What Is a SmartClass? 23-20

Working with SmartClasses 23-21

Reusing PL/SQL 23-22

What Are PL/SQL Libraries? 23-24

Writing Code for Libraries 23-25

Creating Library Program Units 23-26

Attach Library Dialog Box 23-27

Calls and Searches 23-28

Summary 23-30

Practice 23: Overview 23-32

## **24 Using WebUtil to Interact with the Client**

Objectives 24-2

WebUtil: Overview 24-3

Benefits of the WebUtil Utility 24-4

Integrating WebUtil into a Form 24-11

When to Use the WebUtil Functionality 24-13

Interacting with the Client 24-14

Example: Opening a File Dialog Box on the Client 24-15

Example: Reading an Image File into Forms from the Client 24-16

Example: Writing Text Files on the Client 24-17

Example: Executing Operating System Commands on the Client 24-18

Example: Performing OLE Automation on the Client 24-19

Example: Obtaining Environment Information About the Client 24-22

Summary 24-23  
Practice 24: Overview 24-24

## **25 Introducing Multiple Form Applications**

Objectives 25-2  
Multiple Form Applications: Overview 25-3  
Multiple Form Session 25-4  
Benefits of Multiple Form Applications 25-5  
Starting Another Form Module 25-6  
Defining Multiple Form Functionality 25-8  
Conditional Opening 25-10  
Closing the Session 25-11  
Closing a Form with `EXIT_FORM` 25-12  
Other Useful Triggers 25-13  
Sharing Data Among Modules 25-15  
Linking by Global Variables 25-16  
Global Variables: Opening Another Form 25-17  
Global Variables: Restricted Query at Startup 25-18  
Assigning Global Variables in the Opened Form 25-19  
Linking by Parameters 25-20  
Linking by Parameter Lists: The Calling Form 25-21  
Linking by Parameter Lists: The Called Form 25-22  
Linking by Global Record Groups 25-23  
Linking by Shared PL/SQL Variables 25-24  
Summary 25-26  
Practice 25: Overview 25-28

### **Appendix A: Practice Solutions**

### **Appendix B: Table Descriptions**

### **Appendix C: Introduction to Query Builder**

### **Appendix D: Locking in Forms**

### **Appendix E: Oracle Object Features**

### **Appendix F: Using the Layout Editor**



# 16

## Adding Functionality to Items

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

Oracle Internal & OAI Use Only

# Objectives

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

- Supplement the functionality of input items by using triggers and built-ins
- Supplement the functionality of noninput items by using triggers and built-ins



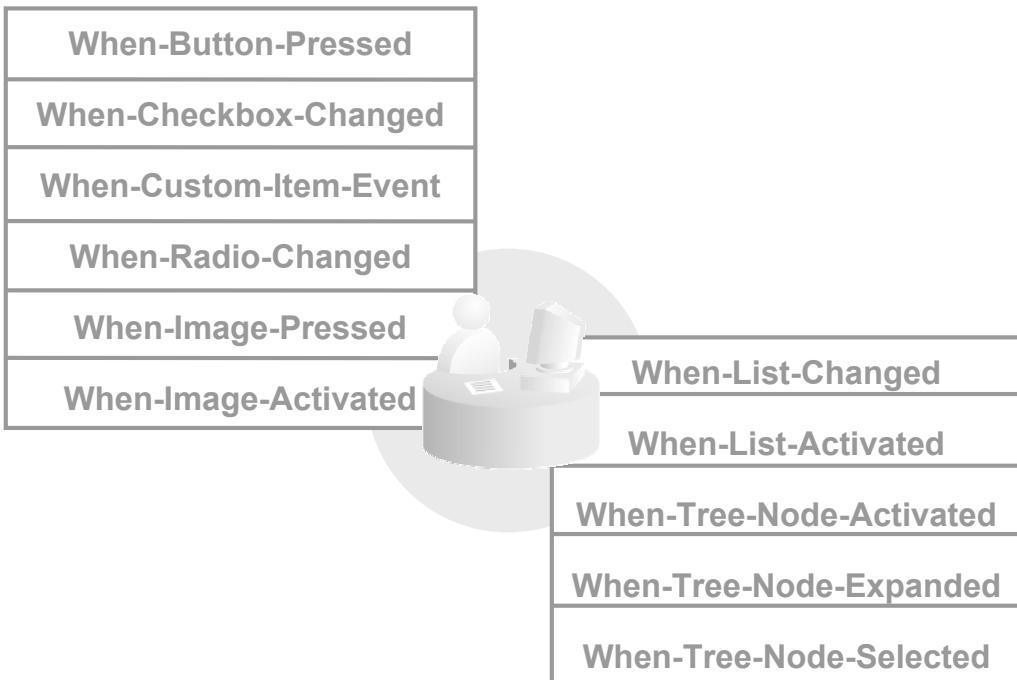
Copyright © 2006, Oracle. All rights reserved.

## Introduction

### Overview

In this lesson, you will learn how to use triggers to provide additional functionality to GUI items in form applications.

# Item Interaction Triggers



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Item Interaction Triggers

There are several types of GUI items that the user can interact with by using the mouse or by pressing a function key. Most of these items have default functionality. For example, by selecting a radio button, the user can change the value of the radio group item.

You will often want to add triggers to provide customized functionality when these events occur. For example:

- Performing tests and appropriate actions as soon as the user clicks a radio button, a list, or a check box
- Conveniently displaying an image when the user clicks an image item
- Defining the functionality of a push-button (which has none until you define it)

## Item Interaction Triggers (continued)

The following triggers fire due to user interaction with an item, as previously described. They can be defined at any scope.

Trigger	Firing Event
When-Button-Pressed	User clicks or uses function key to select
When-Checkbox-Changed	User changes check box state by clicking or by pressing a function key
When-Custom-Item-Event	User selects or changes the value of a JavaBean component
When-Radio-Changed	User selects a different option, or deselects the current option, in a radio group
When-Image-Pressed	User clicks image item
When-Image-Activated	User double-clicks image item
When-List-Changed	User changes value of a list item
When-List-Activated	User double-clicks element in a T-list
When-Tree-Node-Activated	User double-clicks a node or presses [Enter] when a node is selected
When-Tree-Node-Expanded	User expands or collapses a node
When-Tree-Node-Selected	User selects or deselects a node

# Coding Item Interaction Triggers

- **Valid commands:**
  - **SELECT statements**
  - **Standard PL/SQL constructs**
  - **All built-in subprograms**
- **Do not fire during:**
  - **Navigation**
  - **Validation (use When-Validate-“object” to code actions to take place during validation)**

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Command Types in Item Interaction Triggers

You can use standard SQL and PL/SQL statements in these triggers, like the example on the next page. However, you will often want to add functionality to items by calling built-in subprograms, which provide a wide variety of mechanisms.

Although Forms allows you to use DML (INSERT, UPDATE, or DELETE) statements in any trigger, it is best to use them in commit triggers only. Otherwise, the DML statements are not included in the administration kept by Forms concerning commit processing. This may lead to unexpected and unwanted results. You learn about commit triggers in the lesson titled “Transaction Processing.”

**Note:** During an unhandled exception, the trigger terminates and sends the Unhandled Exception message to the operator. The item interaction triggers do not fire on navigation or validation events.

## Command Types in Item Interaction Triggers (continued)

### Example of When-Radio-Changed

When-Radio-Changed trigger on :CUSTOMERS.Credit\_Limit: When the user changes the credit limit, this trigger immediately confirms whether the customer has outstanding orders exceeding the new credit limit. If so, a message warns the user.

```
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;
```

**Note:** Displaying alerts is discussed in the lesson titled “Run-Time Messages and Alerts.”

# Interacting with Check Boxes

First Name  Last Name   
 Perform case-sensitive query on name?

## When-Checkbox-Changed

```
IF CHECKBOX_CHECKED('CONTROL.case_sensitive') 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;
```

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Defining Functionality for Input Items

You have already seen an example of adding functionality to radio groups; you now look at adding functionality to other items that accept user input.

### Check Boxes

When the user selects or deselects a check box, the associated value for the state is set. You may want to perform trigger actions based on this change. Note that the CHECKBOX\_CHECKED function enables you to test the state of a check box without needing to know the associated values for the item.

### Example

The When-Checkbox-Changed trigger (shown in the slide) on the :CONTROL.Case\_Sensitive item enables a query to be executed without regard to case if the box is not selected.

# Changing List Items at Run Time

## Triggers:

- When-List-Changed
- When-List-Activated

## Built-ins:

- ADD\_LIST\_ELEMENT
- DELETE\_LIST\_ELEMENT

Index
Excellent
1
Good
2
Poor
3

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## List Items

You can use the When-List-Changed trigger to trap user selection of a list value. For Tlists, you can trap double-clicks with When-List-Activated.

With Forms Builder, you can also change the selectable elements in a list as follows:

- Periodically update the list from a two-column record group.
- Add or remove individual list elements through the ADD\_LIST\_ELEMENT and DELETE\_LIST\_ELEMENT built-ins, respectively:  
`ADD_LIST_ELEMENT('list_item_name', index, 'label', 'value');  
DELETE_LIST_ELEMENT('list_item_name', index);`

Parameter	Description
Index	Number identifying the element position in the list (top is 1)
Label	The name of the element
Value	The new value for the element

**Note:** You can eliminate the Null list element of a list by changing Required to Yes.

At run time, when the block contains queried or changed records, Forms may not allow you to add or delete elements from a list item.

# Using Dynamic LOVs

- Enables you to use one LOV for multiple items
- Associate LOV with item at run time:  
`SET_ITEM_PROPERTY('lov_name', 'my_lov');`
- Can use a dynamic record group:

```
rg_id :=  
POPULATE_GROUP_WITH_QUERY('my_rg',  
'SELECT ...');  
SET_LOV_PROPERTY('my_lov', GROUP_NAME,  
'my_rg');
```

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Using Dynamic LOVs

LOVs can be made more flexible by assigning them to items at run time rather than at design time. You can use the `SET_ITEM_PROPERTY` built-in to accomplish this.

You can use a single LOV for multiple items by modifying the query of its underlying record group at run time by using the `POPULATE_GROUP_WITH_QUERY` built-in. For example, a Key-Listval trigger for an item could contain the following code:

```
DECLARE rg_id NUMBER;  
BEGIN  
    SET_LOV_PROPERTY('emp_lov', TITLE, 'Sales Reps');  
    rg_id := POPULATE_GROUP_WITH_QUERY('EMP_RG', 'SELECT  
        employee_id ID, first_name || ' ' || last_name NAME  
        FROM employees WHERE job_id = ''SA_REP'''');  
    SYNCHRONIZE;  
    LIST_VALUES;  
END;
```

# Displaying LOVs from Buttons

- **Uses:**
  - Convenient alternative for accessing LOVs
  - Can display independently of text items
- **Needs:**
  - When-Button-Pressed trigger
  - LIST\_VALUES or SHOW\_LOV built-in

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Defining Functionality for Noninput Items

### Displaying LOVs from Buttons

If you have attached an LOV to a text item, then the user can invoke the LOV from the text item by selecting Edit > Display List or by pressing the List Values key.

However, it is always useful if a button is available to display an LOV. The button has two advantages:

- It is a convenient alternative for accessing the LOV.
- It displays an LOV independently of a text item (using SHOW\_LOV).

There are two built-ins that you can call to invoke an LOV from a trigger. These are LIST\_VALUES and SHOW\_LOV.

#### **LIST\_VALUES Procedure**

This built-in procedure invokes the LOV that is attached to the current text item in the form. It has an optional argument, which may be set to RESTRICT, meaning that the current value of the text item is used as the initial search string on the LOV. The default for this argument is NO\_RESTRICT.

## Defining Functionality for Noninput Items (continued)

### SHOW\_LOV Function

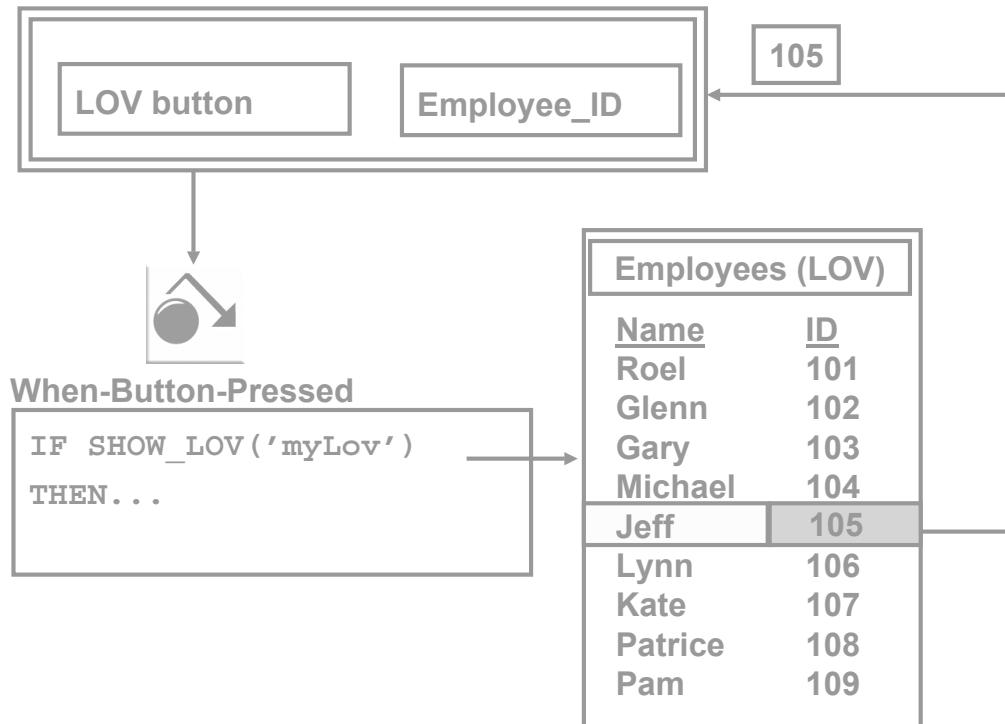
This built-in function, without arguments, invokes the LOV of the current item. However, there are arguments that you can use to define which LOV is to be displayed, what the x and y coordinates are, and where its window should appear:

```
SHOW_LOV( 'lov_name', x, y )
SHOW_LOV( lov_id, x, y )
```

You should note that either the LOV name (in quotation marks) or the LOV ID (without quotation marks) can be supplied in the first argument.

**Note:** The `lov_id` is a PL/SQL variable where the internal ID of the object is stored. Internal IDs are a more efficient way of identifying an object.

## LOVs and Buttons



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Using the SHOW\_LOV Function

The SHOW\_LOV function returns a Boolean value:

- TRUE indicates that the user selected a record from the LOV.
- FALSE indicates that the user dismissed the LOV without choosing a record, or that the LOV returned 0 records from its Record Group.

#### Note

- You can use the FORM\_SUCCESS function to differentiate between the two causes of SHOW\_LOV returning FALSE.
- Create the LOV button with a suitable label, such as "Pick," and arrange it on the canvas where the user intuitively associates it with the items that the LOV supports (even though the button has no direct connection with text items). This is usually adjacent to the main text item that the LOV returns a value to.
- You can use the SHOW\_LOV function to display an LOV that is not even attached to a text item, providing that you identify the LOV in the first argument of the function. When called from a button, this invokes the LOV to be independent of cursor location.

## Using the SHOW\_LOV Function (continued)

- Switch off the button's Mouse Navigate property of the button. When using LIST\_VALUES, the cursor needs to reside in the text item that is attached to the LOV. With SHOW\_LOV, this also maintains the cursor to its original location after the LOV is closed, wherever that may be.

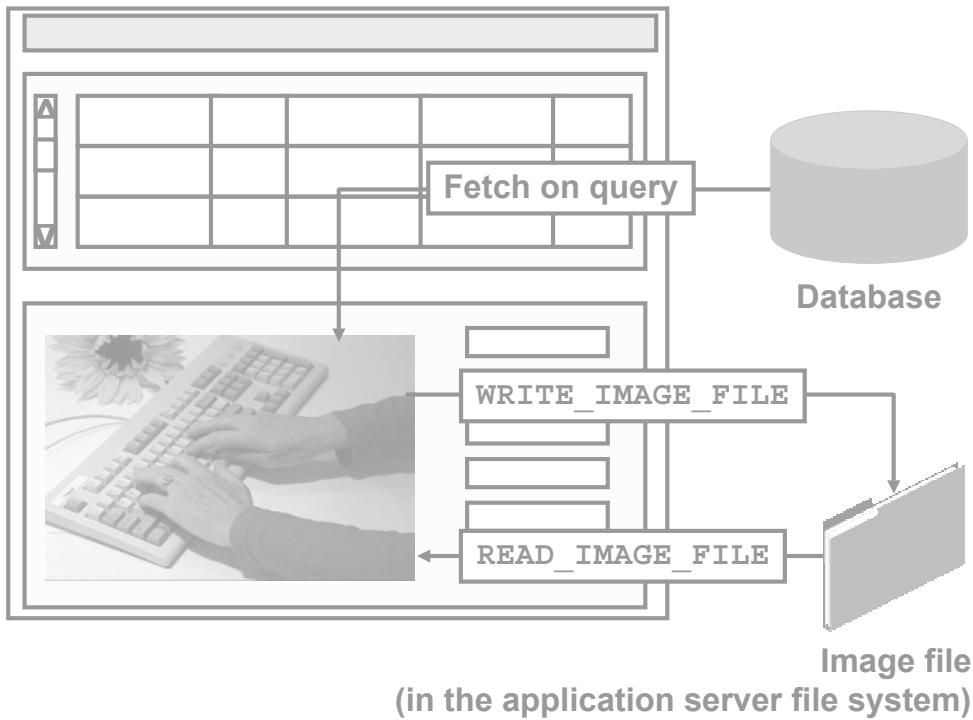
### Example

This When-Button-Pressed trigger on the Customer\_Lov\_Button invokes an LOV in a PL/SQL loop, until the function returns TRUE. Because SHOW\_LOV returns TRUE when the user selects a record, the LOV continues to redisplay if the user does not select a record.

```
LOOP
    EXIT WHEN SHOW_LOV( 'customer_lov' );
    MESSAGE('You must select a value from list');
END LOOP;
```

Oracle Internal & OAI Use Only

## Populating Image Items



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Image Items

Image items that have the Database Item property set to Yes automatically populate in response to a query in the owning block (from a LONG RAW or BLOB column in the base table).

Non-base-table image items, however, need to be populated by other means. For example, from an image file in the file system: the READ\_IMAGE\_FILE built-in procedure.

You might decide to populate an image item from a button trigger, using When-Button-Pressed, but there are two triggers that fire when the user interacts with an image item directly:

- When-Image-Pressed (fires for a click on image item)
- When-Image-Activated (fires for a double-click on image item)

**Note:** The READ\_IMAGE\_FILE built-in procedure loads an image file from the application server file system. If you need to load an image file from the file system on the client, use a JavaBean.

## Image Items (continued)

### READ\_IMAGE\_FILE Procedure

You can use this built-in procedure to load an image file, in a variety of formats, into an image item: `READ_IMAGE_FILE('filename','filetype','item_name');`

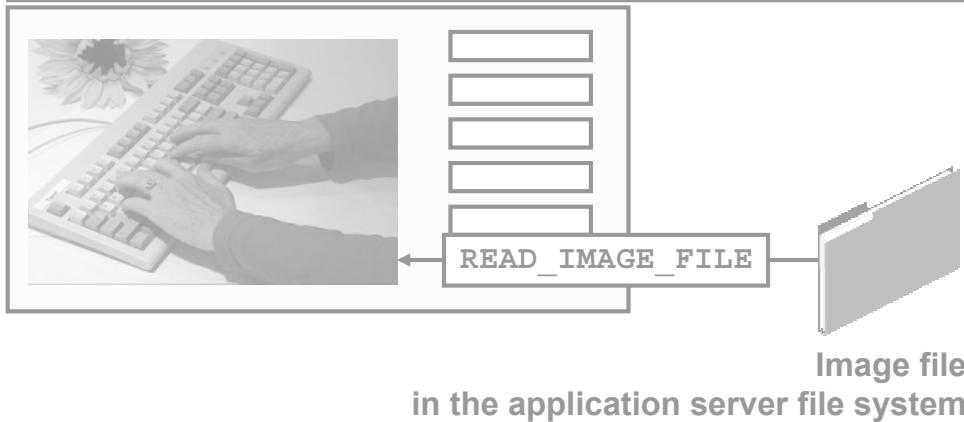
Parameter	Description
<code>filename</code>	Image file name (without a specified path, default path is assumed)
<code>filetype</code>	File type of the image (You can use ANY as a value, but it is recommended to set a specific file type for better performance. Refer to online Help for file types.)
<code>item_name</code>	Name of the image item (a variable holding the <code>Item_id</code> is also valid for this argument.) (This parameter is optional.)

#### Note

- The `filetype` parameter is optional in `READ_IMAGE_FILE`. If you omit `filetype`, you must explicitly identify the `item_name` parameter.
- The reverse procedure, `WRITE_IMAGE_FILE`, is also available.
- The `WRITE_IMAGE_FILE` built-in procedure writes an image file to the application server file system. If you need to write an image file to the file system on the client, use a JavaBean.

# Loading the Right Image

```
READ_IMAGE_FILE  
(TO_CHAR(:ORDER_ITEMS.product_id) || '.JPG',  
'JPEG', 'ORDER_ITEMS.product_image' );
```



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Example of Image Items

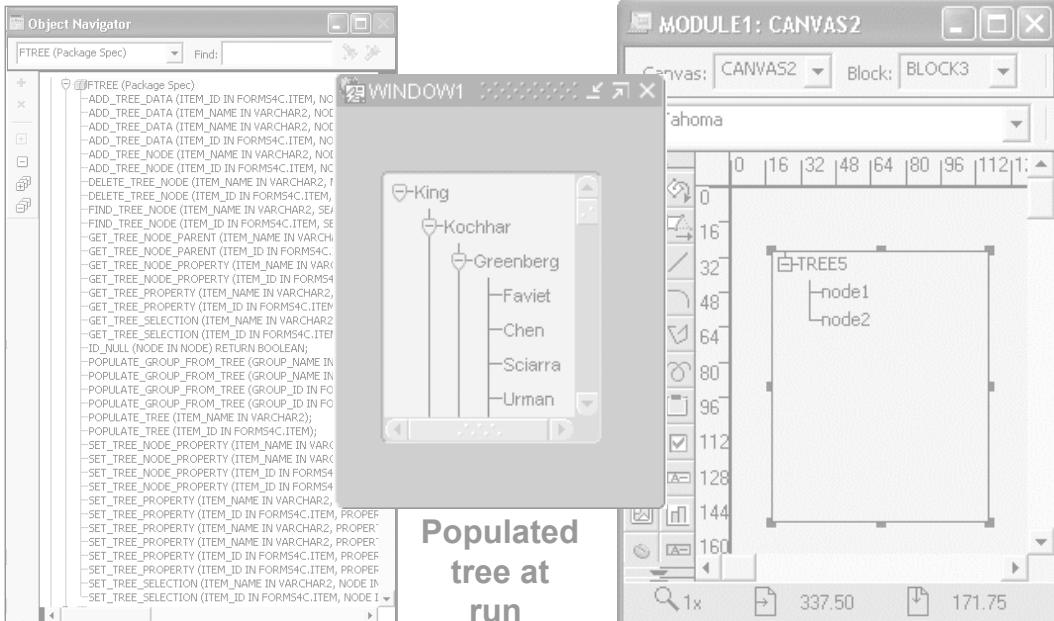
The following When-Image-Pressed trigger on the Product\_Image item displays a picture of the current product (in the ITEM block) when the user clicks the image item. This example assumes that the related file names have the format *<product\_id>.jpg*.

```
READ_IMAGE_FILE(TO_CHAR(:ORDER_ITEMS.product_id) || '.jpg',  
'JPEG', 'ORDER_ITEMS.product_image' );
```

Notice that as the first argument to this built-in is data type CHAR. The concatenated NUMBER item, which is *product\_id*, must be first converted by using the TO\_CHAR function.

**Note:** If you load an image into a base-table image item by using READ\_IMAGE\_FILE, then its contents will be committed to the database LONG RAW or BLOB column when you save changes in the form. You can use this technique to populate a table with images.

# Displaying Hierarchical Trees



FTREE package

Populated  
tree at  
run  
time

Tree at design time

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Displaying Hierarchical Trees

The hierarchical tree displays data in the form of a standard navigator, similar to the Object Navigator used in Oracle Forms Developer.

You can populate a hierarchical tree with values contained in a Record Group or Query Text. At run time, you can programmatically add, remove, modify, or evaluate elements in a hierarchical tree. You can also use the Property Palette to set the populate properties of the hierarchical tree, but when you do this, you must still programmatically populate the tree at run time.

### FTREE Package

The FTREE package contains built-ins and constants to interact with hierarchical tree items in a form. To utilize the built-ins and constants, you must precede their names with the name of the package.

You can add data to a tree view by:

- Populating a tree with values contained in a record group or query by using the POPULATE\_TREE built-in
- Adding data to a tree under a specific node by using the ADD\_TREE\_DATA built-in
- Modifying elements in a tree at run time by using built-in subprograms, such as SET\_TREE\_NODE\_PROPERTY
- Adding or deleting nodes and the data elements under the nodes with ADD\_TREE\_NODE or DELETE\_TREE\_NODE

## Displaying Hierarchical Trees (continued)

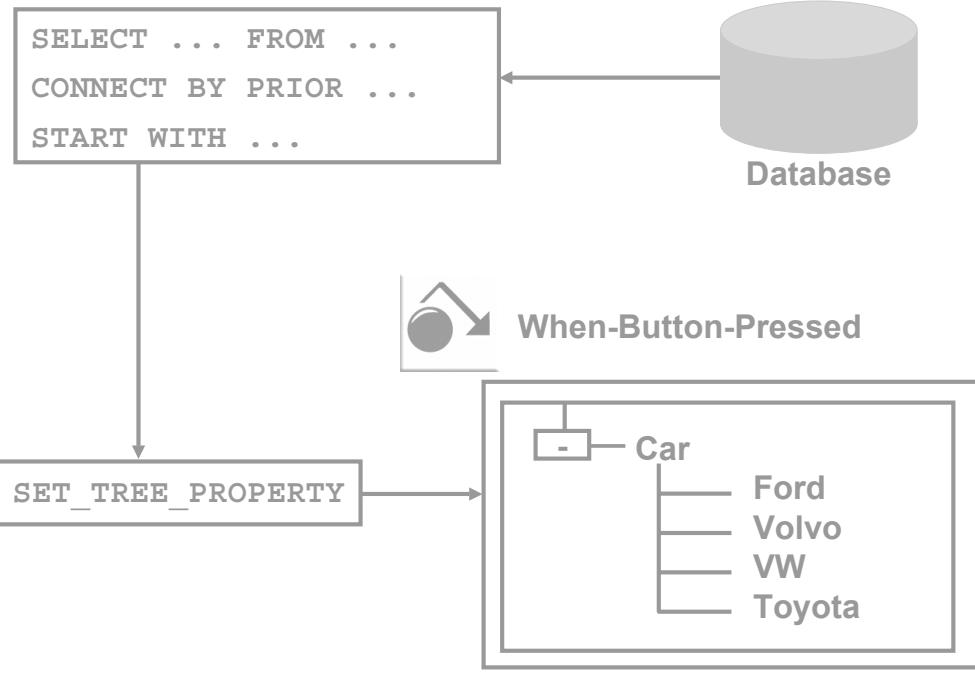
### **SET\_TREE\_PROPERTY Procedure**

This built-in procedure can be used to change certain properties for the indicated hierarchical tree item. It can also be used to populate the indicated hierarchical tree item from a record group.

```
FTREE.SET_TREE_PROPERTY(item_name, Ftree.property, value);
```

Parameter	Description
item_name	Specifies the name of the object created at design time. The data type of the name is VARCHAR2. A variable holding the Item_id is also valid for this argument.
property	Specifies one of the following properties:  RECORD_GROUP: Replaces the data set of the hierarchical tree with a record group and causes it to display  QUERY_TEXT: Replaces the data set of the hierarchical tree with a SQL query and causes it to display  ALLOW_EMPTY_BRANCHES: Possible values are PROPERTY_TRUE and PROPERTY_FALSE.
value	Specifies the value appropriate to the property you are setting

# Populating Hierarchical Trees with Queries



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Populating Hierarchical Trees with Queries

To use a query to populate a hierarchical tree, perform the following steps:

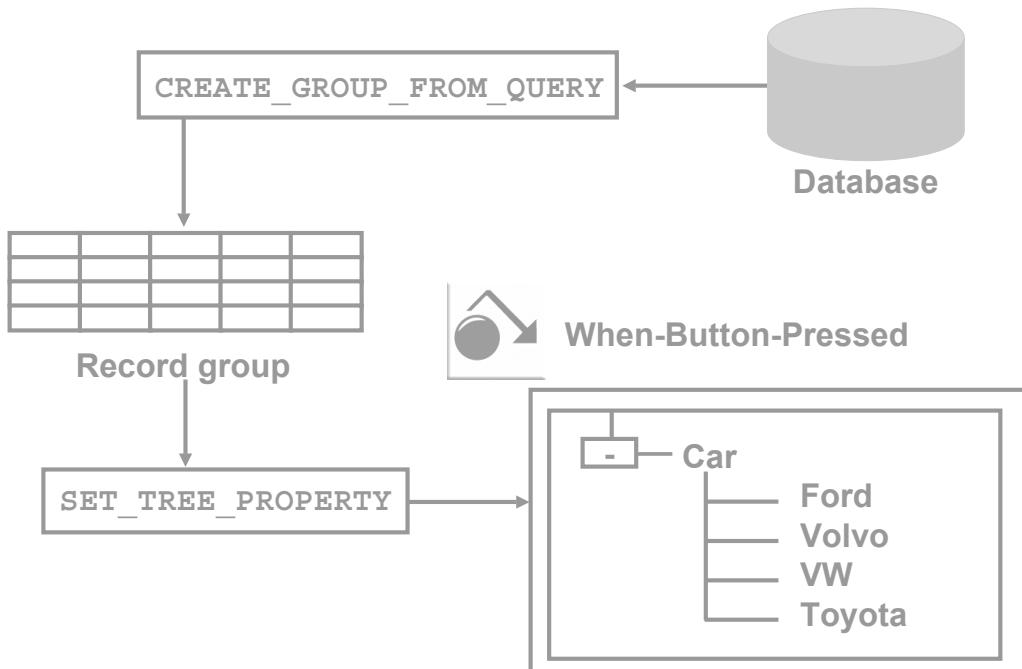
1. Set the Data Query property of the hierarchical tree item, either at design time or programmatically with **SET\_TREE\_PROPERTY**. If you do this programmatically, then this also executes the query and displays the query results in the tree item.
2. If you set the Data Query property at design time, use **POPULATE\_TREE** to execute the query and populate the tree with the query results.

### Example

This code could be used in a When-Button-Pressed or When-New-Form-Instance trigger to initially populate the hierarchical tree with data.

```
DECLARE
    htree ITEM;
    v_qt VARCHAR2(200) := 'SELECT 1, LEVEL, last_name, NULL,
                           TO_CHAR(employee_id) FROM employees
                           CONNECT BY PRIOR
                           employee_id = manager_id START WITH job_id = ''AD_PRES''';
BEGIN
    htree := FIND_ITEM('block3.tree5');
    FTREE.SET_TREE_PROPERTY(htree, Ftree.QUERY_TEXT,v_qt);
END;
```

# Populating Hierarchical Trees with Record Groups



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Populating Hierarchical Trees with Record Groups

To use a record group to populate a hierarchical tree, perform the following steps:

1. Create the record group, either at design time or programmatically.
2. Populate the record group at run time with `POPULATE_GROUP`.
3. Populate and display the hierarchical tree data:
  - Use `POPULATE_TREE` if you have already set the Record Group property of the hierarchical tree item, either in the Property Palette or programmatically.
  - Use `SET_TREE_PROPERTY` if you have not set the Record Group property of the tree; this also populates and displays the tree data if the populated record group exists.

## Populating Hierarchical Trees with Record Groups (continued)

### Example

This code could be used in a When-Button-Pressed or When-New-Form-Instance trigger to initially populate the hierarchical tree with data. The example locates the hierarchical tree first. Then, a record group is created and the hierarchical tree is populated.

```
DECLARE
    htree ITEM;
    v_ignore NUMBER;
    rg_emps RECORDGROUP;
BEGIN
    htree := FIND_ITEM('tree_block.htree3');
    rg_emps := CREATE_GROUP_FROM_QUERY('rg_emps',
        'select 1, LEVEL, last_name, NULL, TO_CHAR(employee_id) '
        || ' from employees ' ||
        'CONNECT BY PRIOR employee_id = manager_id ' ||
        'START WITH job_id = ''AD_PRES'''');
    v_ignore := POPULATE_GROUP(rg_emps);
    FTREE.SET_TREE_PROPERTY(htree,FTREE.RECORD_GROUP,rg_emps);
END;
```

Oracle Internal & OAI Use Only

# Queries to Display Hierarchical Trees

## When-Button-Pressed

```
rg_emps := CREATE_GROUP_FROM_QUERY('rg_emps',
    'SELECT 1, LEVEL, last_name, NULL,
    TO_CHAR(employee_id) ' ||
    'from employees ' ||
    'CONNECT BY PRIOR employee_id = manager_id ' ||
    'START WITH job_id = ''AD_PRES'''');

v_ignore := POPULATE_GROUP(rg_emps);

FTREE.SET_TREE_PROPERTY('block4.tree5',
    FTREE.RECORD_GROUP, rg_emps);
```

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

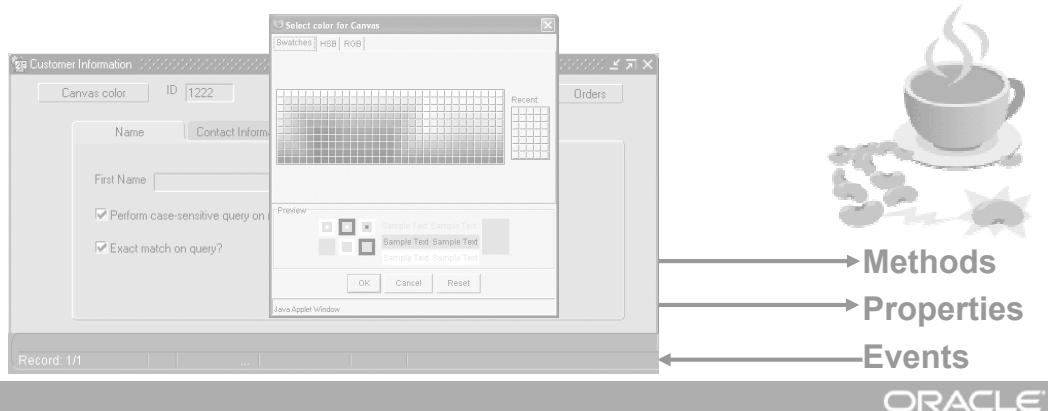
## Record Group or Query

The columns in a record group or query that are used to populate a hierarchical tree are the following:

- **Initial state:** 0 (not expandable), 1 (expanded), or -1 (collapsed)
- **Node tree depth:** Use LEVEL pseudocolumn.
- **Label for the node:** What the user sees
- **Icon for the node:** Picture displayed, if any
- **Data:** Actual value of the node

# Interacting with JavaBeans

- **Tell Forms about the bean: Register**
- **Communication from Forms to JavaBean:**
  - Invoke Methods
  - Get/Set Properties
- **Communication from JavaBean to Forms: Events**



Copyright © 2006, Oracle. All rights reserved.

## Interacting with JavaBeans

In the lesson titled “Creating Noninput Items,” you learned how to add a JavaBean to a form using the Bean Area item. The bean that you add to a form may have a visible component on the form itself, such as a Calendar bean that has its own button to invoke the bean. However, JavaBeans (such as the ColorPicker bean) do not always have visible components, so you may need to create a button or some other mechanism to invoke the bean.

Regardless of whether the bean is visible in the bean area, there must be some communication between the run-time form and the Java classes that comprise the bean. First, the form must be made aware of the bean, either by setting its Implementation Class property at design time or by registering the bean and its events at run time. When the form knows about the bean, it communicates to the bean by:

- Invoking the methods of the bean
- Getting and setting properties of the bean

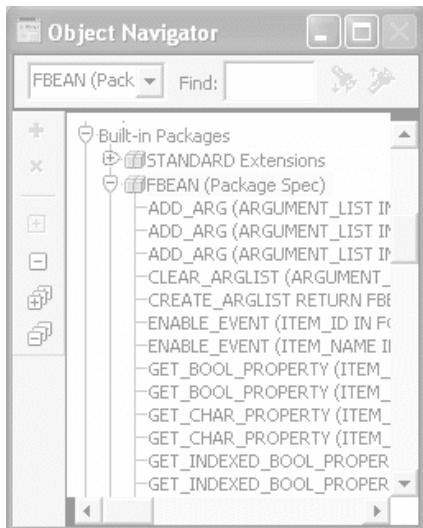
The bean communicates to the form by:

- Sending an event, such as the fact that the user selected a date or color
- Sending a list containing information needed by the form, such as what date or color was selected
- Returning a value from an invoked method

# Interacting with JavaBeans

The FBEAN package provides built-ins to:

- Register the bean
- Invoke methods of the bean
- Get and set properties on the bean
- Subscribe to bean events



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Interacting with JavaBeans (continued)

### FBEAN package

The FBEAN package contains Forms built-ins that enable you to code interactions with JavaBeans in PL/SQL, eliminating the need to know Java in order to communicate with the bean.

Many of the built-ins take some of the same arguments:

- **Item Name or Item ID (obtained with the FIND\_ITEM built-in):** The first argument for most of the FBEAN built-ins, referred to on the next page simply as ITEM.
- **Item Instance:** A reference to which instance of the item should contain the bean. This is applicable where the Bean Area is part of a multirow block and more than one instance of the Bean Area is displayed. This is referred to on the next page as INSTANCE. You can use the value ALL\_ROWS (or FBEAN.ALL\_ROWS) for the Item Instance value to indicate that command should apply to all of the instances of this Bean Area in the block.
- **Note:** This refers to the UI instance of the Bean Area, not the row number in the block. For example, in a block with 5 rows displayed and 100 rows queried, there will be 5 instances of the bean numbered 1 through 5, not 100 instances.
- **Value:** Can accept BOOLEAN, VARCHAR2, or NUMBER data types

## Interacting with JavaBeans (continued)

### FBEAN package (continued)

Some of the built-ins in the FBEAN package are:

- **GET\_PROPERTY (ITEM, INSTANCE, PROPERTY\_NAME)** (returns VARCHAR2):  
Is a function that retrieves the value of the specified property
- **SET\_PROPERTY (ITEM, INSTANCE, PROPERTY\_NAME, VALUE)**:  
Sets the specified property of the bean to the value indicated
- **INVOKE (ITEM, INSTANCE, METHOD\_NAME [, ARGUMENTS] )**:  
Invokes a method on the bean, optionally passing arguments to the method
- **REGISTER\_BEAN (ITEM, INSTANCE, BEAN\_CLASS)**:  
Registers the bean with the form at run time, making all its exposed attributes and methods available for the form's bean item (The last argument is the full class name of the bean, such as 'oracle.forms.demos.beans.ColorPicker').
- **ENABLE\_EVENT (ITEM, INSTANCE, EVENT\_LISTENER\_NAME, SUBSCRIBE)**:  
Is a BOOLEAN indicating whether to subscribe (TRUE) or unsubscribe (FALSE) to the event

Remember to precede calls to any of these built-ins with the package name and a dot, such as `FBEAN.GET_PROPERTY (...)`. You can pass arguments to these built-ins as either a delimited string or as an argument list.

### Deploying the Bean

Because the bean itself is a Java class or set of Java class files separate from the form module, you need to know where to put these files. You can locate these:

- On the middle-tier server, either in the directory structure referenced by the form applet's CODEBASE parameter or in the server's CLASSPATH. CODEBASE is by default the `forms\java` subdirectory of ORACLE\_HOME.
- If using JInitiator, in a JAR file in the middle-tier server's CODEBASE directory, and included in the ARCHIVE parameter so that the JAR file is downloaded to and cached on the client. For example:

```
archive_jini=frmall_jinit.jar,colorpicker.jar
```

(The CODEBASE and ARCHIVE parameters are set in the `formsweb.cfg` file.)

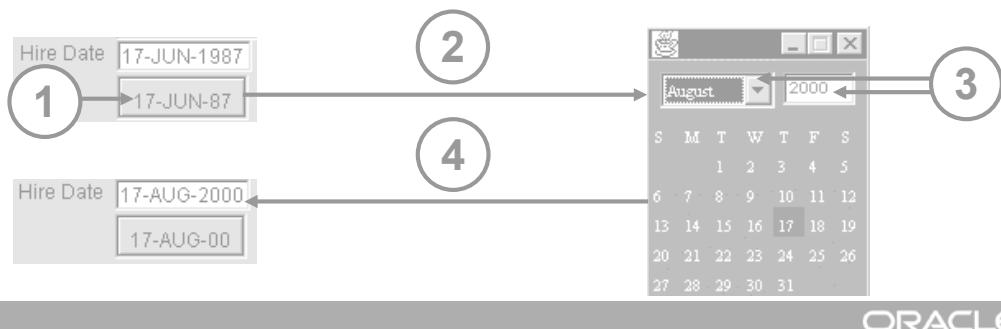
# Interacting with JavaBeans

- **Register a listener for the event:**

```
FBEAN.ENABLE_EVENT('MyBeanArea', 1, 'mouseListener',  
, true);
```

- **When an event occurs on the bean:**

- The When-Custom-Item-Event trigger fires
- The name and information are sent to Forms in:
  - :SYSTEM.CUSTOM\_ITEM\_EVENT
  - :SYSTEM.CUSTOM\_ITEM\_EVENT\_PARAMETERS



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Interacting with JavaBeans (continued)

### Responding to Events

When a user interacts with a JavaBean at run time, it usually causes an event to occur. You can use FBEAN.ENABLE\_EVENT to register a listener for the event, so that when the event occurs, Forms will fire the When-Custom-Item-Event trigger. In this trigger, you can code a response to the event. The :SYSTEM.CUSTOM\_ITEM\_EVENT and :SYSTEM.CUSTOM\_EVENT\_PARAMETERS variables contain the name of the event and information the bean is sending to the form.

A typical interaction that could occur includes the following steps:

1. The user clicks the bean area for a Calendar bean. This bean area has a visible component on the form that looks like a button. The label is set to the hire date for an employee.
2. The Calendar bean is invoked and displays a calendar initially set to the employee's hire date.
3. The user changes the date on the bean by picking a new month and year, and then clicking on a day, which initiates the DateChanged event.
4. The When-Custom-Item-Event trigger obtains the changed date and assigns it back to the employee hire\_date item, also changing the label on the bean area "button."

## Interacting with JavaBeans (continued)

### Coding a When-Custom-Item-Event Trigger

In a When-Custom-Item-Event trigger to respond to JavaBeans events, you code the action that you want to take place when an event occurs.

For example, when a user selects a date from the Calendar bean, the DateChange event takes place and the When-Custom-Item-Event trigger fires. In the code for the When-Custom-Item-Event trigger on the Calendar bean area item, you need to obtain the name of the event. If it is the DateChange event, you must obtain the new date and assign it to a form item, such as the employee's hire date. You can use the system variables containing the event and parameter information:

```
declare
    hBeanEventDetails ParamList;
    eventName varchar2(80);
    paramType number;
    eventType varchar2(80);
    newDateVal varchar2(80);
    newDate date := null;
begin
    hBeanEventDetails := get_parameter_list
        (:system.custom_item_event_parameters);
    eventName := :system.custom_item_event;
    if(eventName = 'DateChange') then
        get_parameter_attr(hBeanEventDetails,
            'DateValue', ParamType, newDateVal);
        newDate := to_date(newDateVal,'DD.MM.YYYY');
    end if;
    :employees.hire_date := newDate;
end;
```

The preceding example is for a bean that uses hand-coded integration. If you use the FBEAN package to integrate the bean, the name of the value passed back to the form is always called 'DATA'.

For example:

```
get_parameter_attr(:system.custom_item_event_parameters,
    'DATA', paramType, eventData);
(where paramType and eventData are PL/SQL variables you declare in the When-Custom-Item-Event trigger, like paramType and newDateVal in the preceding example).
```

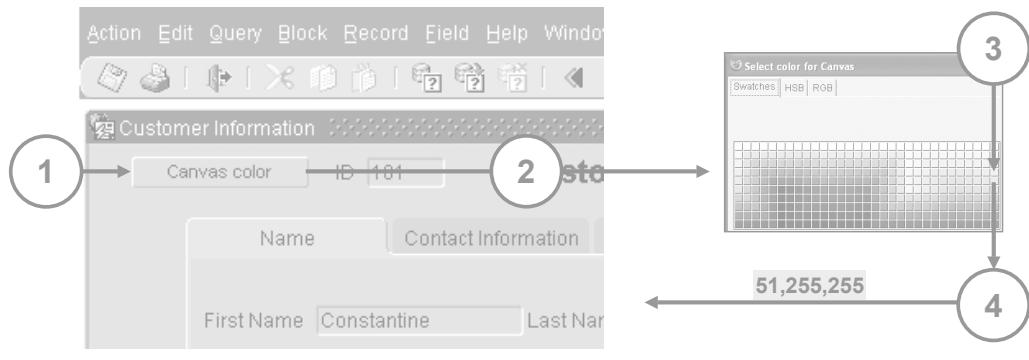
**Note:** There are many examples of JavaBeans in the Forms Demos that you can download from OTN:

[http://otn.oracle.com/sample\\_code/products/forms/index.html](http://otn.oracle.com/sample_code/products/forms/index.html)

# Interacting with JavaBeans

The JavaBean may:

- Not have a visible component
- Not communicate via events
- Return a value to the form when invoked (used like a function)



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Interacting with JavaBeans (continued)

### Getting Values from JavaBeans Without Events

Not all information from JavaBeans is obtained via events. For example, a JavaBean may return a value when one of its methods is invoked. This value may be assigned to a PL/SQL variable or Forms item, similar to the way a function returns a value.

An example of this is the ColorPicker bean that you added to the Customer form in the lesson titled “Creating Noninput Items.” It contains a single method that returns a value, and also has no visible component in the bean area of the form. To invoke the bean and obtain a value from it, you can use a Forms push button and trigger with code similar to the following:

```
vcNewColor := FBean.Invoke_char(hColorPicker,  
1,'showColorPicker','Select color for canvas');
```

The INVOKE\_CHAR built-in is used to call a method that returns a VARCHAR2 value.

1. User clicks button to invoke the bean.
2. The Color Picker component appears.
3. User selects a color.
4. The color value (RGB values in comma-separated list) is returned to the vcNewColor variable. The code can then use the color value to set the canvas color.

## Summary

In this lesson, you should have learned that:

- You can use triggers to supplement the functionality of:
  - Input items:  
When-[Checkbox | Radio]-Changed  
When-List-[Changed | Activated]
  - Noninput items:  
When-Button-Pressed  
When-Image-[Pressed | Activated]  
When-Tree-Node-[Activated | Expanded | Selected]  
When-Custom-Item-Event

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Summary

In this lesson, you should have learned to use triggers to provide functionality to the GUI items in form applications.

- The item interaction triggers accept SELECT statements and other standard PL/SQL constructs.

# Summary

In this lesson, you should have learned that:

- You can call useful built-ins from triggers:
  - CHECKBOX\_CHECKED
  - [ADD | DELETE]\_LIST\_ELEMENT
  - SHOW\_LOV
  - [READ | WRITE]\_IMAGE\_FILE
  - FTREE: POPULATE\_TREE, ADD\_TREE\_DATA,  
[GET | SET]\_TREE\_PROPERTY
  - FBEAN: [GET | SET]\_PROPERTY, INVOKE,  
REGISTER\_BEAN, ENABLE\_EVENT

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Summary (continued)

- There are built-ins for check boxes, LOV control, list item control, image file reading, hierarchical tree manipulation, interaction with JavaBeans, and so on.

## Practice 16: Overview

This practice covers the following topics:

- Writing a trigger to check whether the customer's credit limit has been exceeded
- Creating a toolbar button to display and hide product images
- Coding a button to enable users to choose a canvas color for a form

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Practice 16: Overview

In this practice, you create some additional functionality for a radio group. You also code interaction with a JavaBean. Finally, you add some triggers that enable interaction with buttons.

- Writing a trigger to check whether the customer's credit limit has been exceeded
- Coding a button to enable users to choose a canvas color for a form
- Creating a toolbar button to display and hide product images

**Note:** For solutions to this practice, see Practice 16 in Appendix A, "Practice Solutions."

## Practice 16

1. In the CUSTGXX 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.
2. Click Run Form to run the 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 want 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 should not receive a warning when changing this customer's credit limit.

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

The button should call a procedure named PickColor, with the imported text from 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.

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.
5. In the ORDGXX 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, width, height, and color properties like the other toolbar buttons.
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.
7. Save and compile the form. Click Run Form to run the form and test the functionality.

**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.

# Run-Time Messages and Alerts

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

Oracle Internal & OAI Use Only

# Objectives

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

- **Describe the default messaging behavior of a form**
- **Handle run-time failure of built-in subprograms**
- **Identify the different types of Forms messages**
- **Control system messages**
- **Create and control alerts**
- **Handle database server errors**

ORACLE®

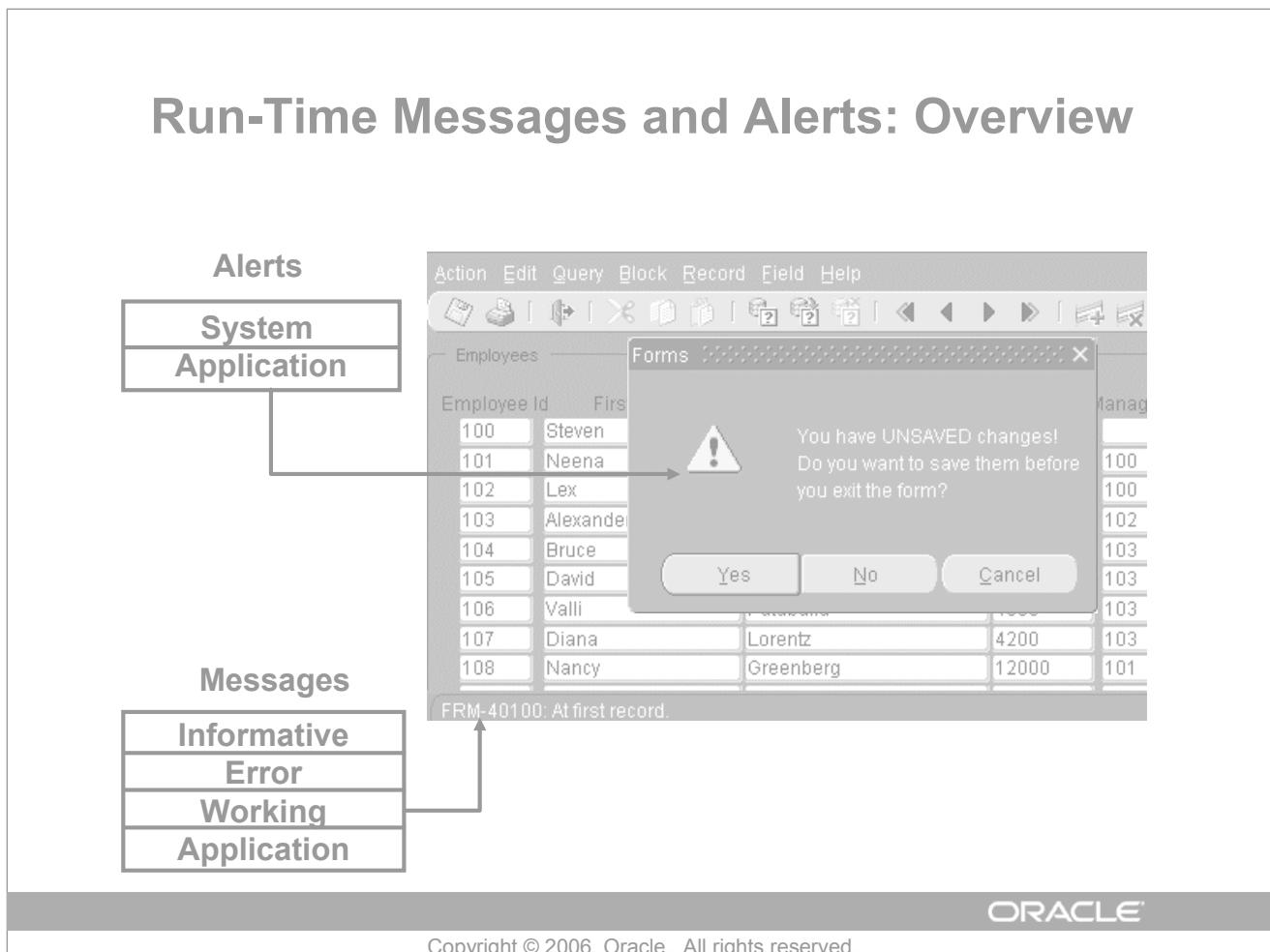
Copyright © 2006, Oracle. All rights reserved.

## Introduction

### Overview

This lesson shows you how to intercept system messages, and if desired, replace them with ones that are more suitable for your application. You will also learn how to handle errors by using built-in subprograms, and how to build customized alerts for communicating with users.

# Run-Time Messages and Alerts: Overview



## Run-Time Messages and Alerts: Overview

Forms displays messages at run time to inform the operator of events that occur in the session. As the designer, you may want to either suppress or modify some of these messages, depending on the nature of the application.

Forms can communicate with the user in the following ways:

- **Informative message:** A message tells the user the current state of processing, or gives context-sensitive information. The default display is on the message line. You can suppress its appearance with an On-Message trigger.
- **Error message:** This informs the user of an error that prevents the current action. The default display is on the message line. You can suppress message line errors with an On-Error trigger.
- **Working message:** This tells the operator that the form is currently processing (for example, Working...). This is shown on the message line. This type of message can be suppressed by setting the system variable `SUPPRESS_WORKING` to True:  
`:SYSTEM.SUPPRESS_WORKING := 'TRUE';`

## Run-Time Messages and Alerts: Overview (continued)

- **System alert:** Alerts give information to the operator that require either an acknowledgment or an answer to a question before processing can continue. This is displayed as a modal window. When more than one message is waiting to show on the message line, the current message is also displayed as an alert.

You can also build messages and alerts into your application:

- **Application message:** These are messages that you build into your application by using the MESSAGE built-in. The default display is on the message line.
- **Application alert:** These are alerts that you design as part of your application, and issue to the operator for a response by using the SHOW\_ALERT built-in.

Oracle Internal & OAI Use Only

# Detecting Run-Time Errors

- **FORM\_SUCCESS**
  - **TRUE:** Action successful
  - **FALSE:** Error/Fatal error occurred
- **FORM\_FAILURE**
  - **TRUE:** A nonfatal error occurred
  - **FALSE:** Action successful or a fatal error occurred
- **FORM\_FATAL**
  - **TRUE:** A fatal error occurred
  - **FALSE:** Action successful or a nonfatal error occurred

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Built-ins and Handling Errors

When a built-in subprogram fails, it does not directly cause an exception in the calling trigger or program unit. This means that subsequent code continues after a built-in fails, unless you take action to detect a failure.

### Example

A button in the CONTROL block called Stock\_Button is situated on the Toolbar canvas of the ORDERS form. When clicked, the When-Button-Pressed trigger navigates to the INVENTORIES block, and performs a query there:

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

If the GO\_BLOCK built-in procedure fails because the INVENTORIES block does not exist, or because it is nonenterable, then the EXECUTE\_QUERY procedure still executes, and attempts a query in the wrong block.

## Built-ins and Handling Errors (continued)

### Built-in Functions for Detecting Success and Failure

Forms Builder supplies some functions that indicate whether the latest action in the form was successful.

Built-In Function	Description of Returned Value
FORM_SUCCESS	TRUE: Action successful FALSE: Error or fatal error occurred
FORM_FAILURE	TRUE: A nonfatal error occurred FALSE: Either no error, or a fatal error
FORM_FATAL	TRUE: A fatal error occurred FALSE: Either no error, or a nonfatal error

**Note:** These built-in functions return success or failure of the latest action in the form. The failing action may occur in a trigger that fired as a result of a built-in from the first trigger. For example, the EXECUTE\_QUERY procedure can cause a Pre-Query trigger to fire, which may itself fail.

# Errors and Built-Ins

- **Built-in failure does not cause an exception.**
- **Test built-in success with the FORM\_SUCCESS function:**  
`IF FORM_SUCCESS THEN . . .`  
`or IF NOT FORM_SUCCESS THEN . . .`
- **What went wrong?**
  - ERROR\_CODE, ERROR\_TEXT, ERROR\_TYPE
  - MESSAGE\_CODE, MESSAGE\_TEXT, MESSAGE\_TYPE

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Errors and Built-Ins

It is usually most practical to use FORM\_SUCCESS because this returns FALSE if either a fatal or a nonfatal error occurs. You can then code the trigger to take appropriate action.

### Example of FORM\_SUCCESS

Here is the same trigger again. This time, the FORM\_SUCCESS function is used in a condition to decide whether the query should be performed, depending on the success of the GO\_BLOCK action.

```
GO_BLOCK('INVENTORIES');
IF FORM_SUCCESS THEN
  EXECUTE_QUERY;
ELSE
  MESSAGE('An error occurred while navigating to Stock');
END IF;
```

Triggers fail only if there is an unhandled exception or you raise the FORM\_TRIGGER\_FAILURE exception to fail the trigger in a controlled manner.

## Errors and Built-Ins (continued)

**Note:** The program unit CHECK\_PACKAGE\_FAILURE, which is written when you build master-detail blocks, may be called to fail a trigger if the last action was unsuccessful.

### Built-in Functions to Determine the Error

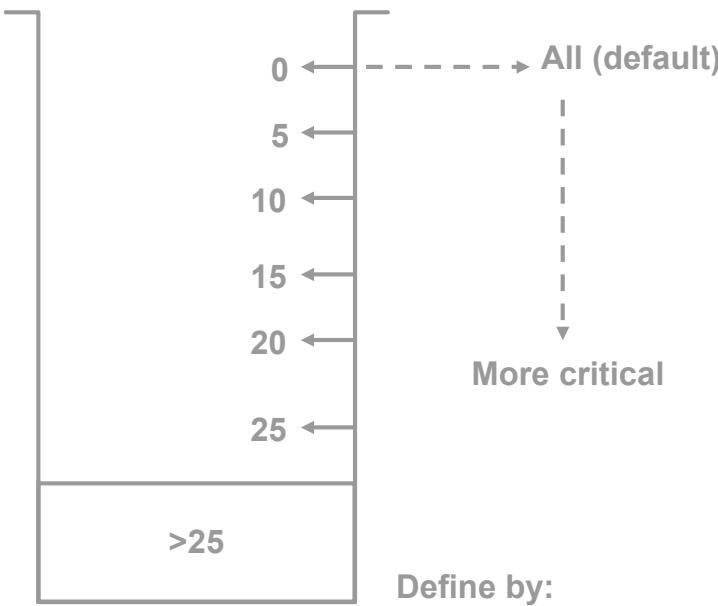
When you detect an error, you may need to identify it to take a specific action. Three more built-in functions provide this information:

Built-In Function	Description of Returned Value
ERROR_CODE	Error number (datatype NUMBER)
ERROR_TEXT	Error description (datatype CHAR)
ERROR_TYPE	FRM=Forms Builder error, ORA=Oracle error (datatype CHAR)

These built-ins are explained in detail later in this lesson.

Oracle Internal & OAI Use Only

# Message Severity Levels



Define by:

`:SYSTEM.MESSAGE_LEVEL`

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Controlling System Messages

### Suppressing Messages According to Their Severity

You can prevent system messages from being issued, based on their severity level. Forms Builder classifies every message with a severity level that indicates how critical or trivial the information is; the higher the numbers, the more critical the message. There are six levels that you can affect.

Severity Level	Description
0	All messages
5	Reaffirms an obvious condition
10	User has made a procedural mistake
15	User attempting action for which the form is not designed
20	Cannot continue intended action due to a trigger problem or some other outstanding condition
25	A condition that could result in the form performing incorrectly
> 25	Messages that cannot be suppressed

## **Controlling System Messages (continued)**

### **Suppressing Messages According to Their Severity (continued)**

In a trigger, you can specify that only messages above a specified severity level are to be issued by the form. You do this by assigning a value to the MESSAGE\_LEVEL system variable. Forms then issues only those messages that are above the severity level defined in this variable.

The default value for MESSAGE\_LEVEL (at form startup) is 0. This means that messages of all severities are displayed.

Oracle Internal & OAI Use Only

# Suppressing Messages

```
:SYSTEM.MESSAGE_LEVEL := '5';
UP;
IF NOT FORM_SUCCESS THEN
  MESSAGE('Already at the first Order');
END IF;
:SYSTEM.MESSAGE_LEVEL := '0';
```

```
:SYSTEM.SUPPRESS_WORKING := 'TRUE';
```



Copyright © 2006, Oracle. All rights reserved.

## Example of Suppressing Messages

The following When-Button-Pressed trigger moves up one record, using the built-in procedure UP. If the cursor is already on the first record, the built-in fails and the following message usually displays: FRM-40100: At first record.

This is a severity level 5 message. However, the trigger suppresses this, and outputs its own application message instead. The trigger resets the message level to normal (0) afterwards.

```
:SYSTEM.MESSAGE_LEVEL := '5';
UP;
IF      NOT FORM_SUCCESS THEN
  MESSAGE('Already at the first Order');
END IF;
:SYSTEM.MESSAGE_LEVEL := '0';
```

## Example of Suppressing Messages (continued)

### Suppressing Working Messages

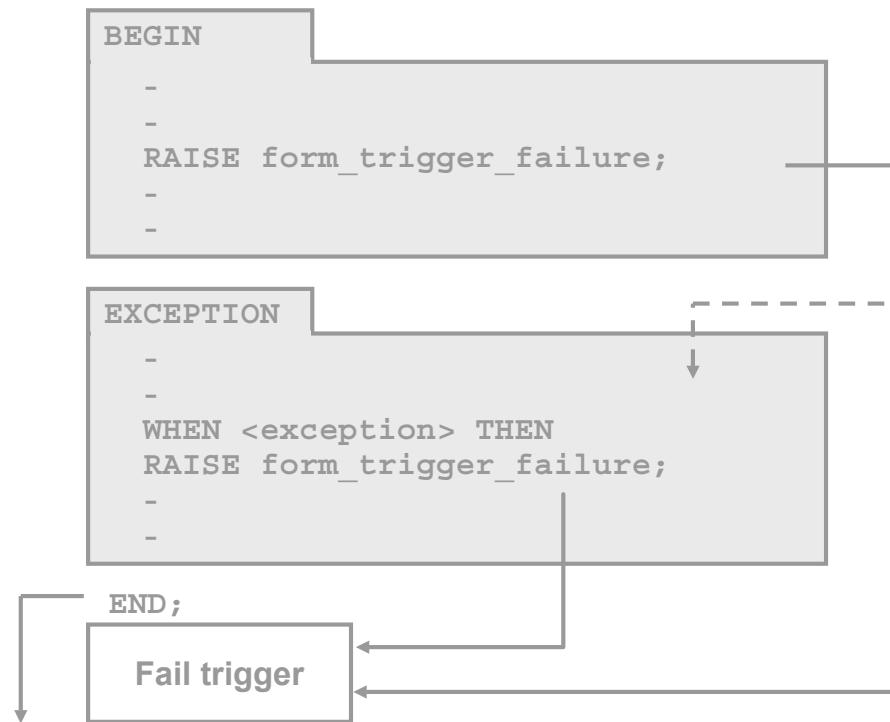
Working messages are displayed when the Forms is busy processing an action. For example, while querying you receive the message: Working.... You can suppress this message by setting the system variable SUPPRESS\_WORKING to True:

```
:SYSTEM.SUPPRESS_WORKING := 'TRUE';
```

**Note:** You can set these system variables as soon as the form starts up, if required, by performing the assignments in a When-New-Form-Instance trigger.

Oracle Internal & OAI Use Only

## FORM\_TRIGGER\_FAILURE Exception



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### FORM\_TRIGGER\_FAILURE Exception

Triggers fail only when one of the following occurs:

- During an unhandled exception
- When you request the trigger to fail by raising the built-in exception FORM\_TRIGGER\_FAILURE

This exception is defined and handled by Forms Builder, beyond the visible trigger text that you write. You can raise this exception:

- In the executable part of a trigger, to skip remaining actions and fail the trigger
- In an exception handler, to fail the trigger *after* your own exception-handling actions have been obeyed

In either case, Forms Builder has its own exception handler for FORM\_TRIGGER\_FAILURE, which fails the trigger but does *not* cause an unhandled exception. This means that you can fail the trigger in a controlled manner.

## **FORM\_TRIGGER\_FAILURE Exception (continued)**

### **Example**

This example adds an action to the exception handler of the When-Validate-Item trigger for the Customer\_ID item. It raises an exception to fail the trigger when the message is sent, and therefore, traps the user in the Customer\_ID item:

```
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('Customer with this ID not found');  
  RAISE form_trigger_failure;
```

Oracle Internal & OAI Use Only

# Triggers for Intercepting System Messages

- **On-Error:**
  - Fires when a system error message is issued
  - Is used to trap Forms and Oracle Server errors, and to customize error messages
- **On-Message:**
  - Fires when an informative system message is issued
  - Is used to suppress or customize specific messages

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Triggers for Intercepting System Messages

By writing triggers that fire on message events, you can intercept system messages before they are displayed on the screen. These triggers are:

- **On-Error:** Fires on issuance of a system error message
- **On-Message:** Fires on issuance of an informative system message

These triggers replace the display of a message, so that no message is seen by the operator unless you issue one from the trigger itself.

You can define these triggers at any level. For example, an On-Error trigger at item level intercepts only those error messages that occur while control is in that item. However, if you define one or both of these triggers at form level, all messages that cause them to fire will be intercepted regardless of which object in the current form causes the error or message.

### On-Error Trigger

Use this trigger to:

- Detect Forms and Oracle Server errors. This trigger can perform corrective actions based on the error that occurred.
- Replace the default error message with a customized message for this application

## Triggers for Intercepting System Messages (continued)

Remember that you can use the built-in functions ERROR\_CODE, ERROR\_TEXT, and ERROR\_TYPE to identify the details of the error, and possibly use this information in your own message.

### Example of an On-Error Trigger

This On-Error trigger sends a customized message for error 40202 (field must be entered), but reconstructs the standard system message for all other errors.

```
IF ERROR_CODE = 40202 THEN
    MESSAGE('You must fill in this field for an Order');
ELSE
    MESSAGE(ERROR_TYPE || '-' || TO_CHAR(ERROR_CODE) ||
    ': ' ||
    ERROR_TEXT);
END IF;
RAISE FORM_TRIGGER_FAILURE;
```

Oracle Internal & OAI Use Only

# Handling Informative Messages

- **On-Message trigger**
- **Built-in functions:**
  - **MESSAGE\_CODE**
  - **MESSAGE\_TEXT**
  - **MESSAGE\_TYPE**

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## On-Message Trigger

Use this trigger to suppress informative messages, replacing them with customized application messages, as appropriate.

You can handle messages in On-Message in a similar way to On-Error. However, because this trigger fires due to informative messages, you will use different built-ins to determine the nature of the current message.

Built-In Function	Description of Returned Value
MESSAGE_CODE	Number of informative message that would have displayed (NUMBER data type)
MESSAGE_TEXT	Text of informative message that would have displayed (CHAR data type)
MESSAGE_TYPE	FRM=Forms Builder message ORA=Oracle server message NULL>No message issued yet in this session (CHAR data type)

## On-Message Trigger (continued)

**Note:** These functions return information about the most recent message that was issued. If your applications must be supported in more than one national language, then use MESSAGE\_CODE in preference to MESSAGE\_TEXT when checking a message.

### Example of an On-Message trigger

This On-Message trigger modifies the “Query caused no records to be retrieved” message (40350) and the “Query caused no records to be retrieved. Re-enter.” message (40301):

```
IF      MESSAGE_CODE in (40350,40301) THEN
    MESSAGE('No Orders found-check your search values');
ELSE
    MESSAGE(MESSAGE_TYPE || '-' || TO_CHAR(MESSAGE_CODE)
           || : '|| MESSAGE_TEXT);
END IF;
```

Oracle Internal & OAI Use Only

# Setting Alert Properties

<input type="checkbox"/> Title	This is the Title
<input type="checkbox"/> Message	Alert Message (Maximum 200 characters) Can appear on multiple lines
<input type="checkbox"/> Alert Style	Caution
<input type="checkbox"/> Button 1 Label	Label 1
<input type="checkbox"/> Button 2 Label	Label 2
<input type="checkbox"/> Button 3 Label	Label 3
<input checked="" type="radio"/> Default Alert Button	Button 1

The diagram illustrates a generic alert window with the following components and their corresponding numbers:

- 1:** Title bar containing "This is the Title".
- 2:** Alert message area containing "Alert Message (Maximum 200 characters) Can appear on multiple lines".
- 3:** Alert Styles section showing three options: Caution (with exclamation mark icon), Stop (with circular icon), and Note (with clipboard icon).
- 4:** Button 1 Label: "Label 1".
- 5:** Button 2 Label: "Label 2".
- 6:** Button 3 Label: "Label 3".
- 7:** Default Alert Button.

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Setting Alert Properties: Example

The slide shows a generic example of an alert, showing all three icons and buttons that can be defined.

1	Title
2	Message
3	Alert Style (Caution, Stop, Note)
4	Button1 label
5	Button2 label
6	Button3 label
7	Default Alert Button

## Setting Alert Properties

### Creating and Controlling Alerts

Alerts are an alternative method for communicating with the operator. Because they are displayed in a modal window, alerts provide an effective way of drawing attention and forcing the operator to answer the message before processing can continue.

Use alerts when you need to perform the following:

- Display a message that the operator cannot ignore, and must acknowledge.
- Ask the operator a question where up to three answers are appropriate (typically Yes, No, or Cancel).

You handle the display and responses to an alert by using built-in subprograms. Alerts are, therefore, managed in two stages:

- Create the alert at design time, and define its properties in the Property Palette.
- Activate the alert at run time by using built-ins, and take action based on the operator's returned response.

### How to Create an Alert

Like the other objects that you create at design time, alerts are created from the Object Navigator.

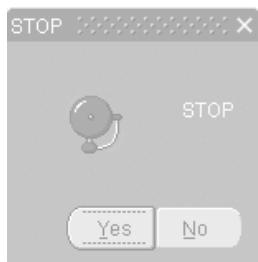
- Select the Alerts node in the Navigator, and then click Create.
- Define the properties of the alert in the Property Palette.

The properties that you can specify for an alert include the following:

Property	Description
Name	Name for this object
Title	Title that displays on alert
Alert Style	Symbol that displays on alert: Stop, Caution, or Note
Button1, Button2, Button3 Labels	Labels for each of the three possible buttons (Null indicates that the button will not be displayed.)
Default Alert Button	Specifies which button is selected if user presses [Enter]
Message	Message that will appear in the alert—can be multiple lines, but maximum of 200 characters

# Planning Alerts

Yes/No  
questions



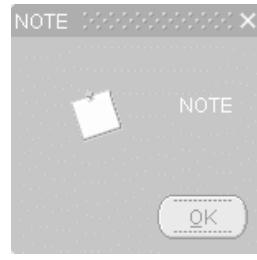
Yes/No/Cancel  
questions



Caution  
messages



Informative  
messages



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Planning Alerts: How Many Do You Need?

Potentially, you can create an alert for every separate alert message that you need to display, but this is usually unnecessary. You can define a message for an alert at run time, before it is displayed to the operator. A single alert can be used for displaying many messages if the available buttons are suitable for responding to the messages.

Create an alert for each combination of:

- Alert style required
- Set of available buttons (and labels) for operator response

For example, an application might require one Note-style alert with a single button (OK) for acknowledgment, one Caution alert with a similar button, and two Stop alerts that each provide a different combination of buttons for a reply. You can then assign a message to the appropriate alert before its display, through the `SET_ALERT_PROPERTY` built-in procedure.

# Controlling Alerts



Copyright © 2006, Oracle. All rights reserved.

## Controlling Alerts at Run Time

There are built-in subprograms to change an alert message to change alert button labels, and to display the alert, which returns the operator's response to the calling trigger.

### **SET\_ALERT\_PROPERTY** Procedure

Use this built-in to change the message that is currently assigned to an alert. At form startup, the default message (as defined in the Property Palette) is initially assigned:

```
SET_ALERT_PROPERTY('alert_name',property,'message')
```

Parameter	Description
Alert_name	The name of the alert as defined in Forms Builder (You can alternatively specify an alert_id (unquoted) for this argument.)
Property	The property being set (Use ALERT_MESSAGE_TEXT when defining a new message for the alert.)
Message	The character string that defines the message (You can give a character expression instead of a single quoted string, if required.)

## Controlling Alerts at Run Time (continued)

### SET\_ALERT\_BUTTON\_PROPERTY Procedure

Use this built-in to change the label on one of the alert buttons:

```
SET_ALERT_BUTTON_PROPERTY('alert_name', button, property,  
                           'value')
```

Parameter	Description
Alert_name	The name of the alert, as defined in Forms Builder (You can alternatively specify an alert_id (unquoted) for this argument.)
Button	The number that specifies the alert button (Use ALERT_BUTTON1, ALERT_BUTTON2, and ALERT_BUTTON3 constants.)
Property	The property being set; use LABEL
Value	The character string that defines the label

Oracle Internal & OAI Use Only

## SHOW\_ALERT Function

```
IF SHOW_ALERT('del_Check')=ALERT_BUTTON1 THEN
```

```
... . . .
```



Alert\_Button1  
Alert\_Button2  
Alert\_Button3

Copyright © 2006, Oracle. All rights reserved.

ORACLE

### SHOW\_ALERT Function

SHOW\_ALERT is how you display an alert at run time, and return the operator's response to the calling trigger:

```
selected_button := SHOW_ALERT('alert_name');  
... . . .
```

*Alert\_Name* is the name of the alert, as defined in the builder. You can alternatively specify an *Alert\_Id* (unquoted) for this argument.

SHOW\_ALERT returns a NUMBER constant, which indicates which of the three possible buttons the user clicked in response to the alert. These numbers correspond to the values of three PL/SQL constants, which are predefined by the Forms Builder:

If the number equals...	The operator selected...
ALERT_BUTTON1	Button 1
ALERT_BUTTON2	Button 2
ALERT_BUTTON3	Button 3

After displaying an alert that has more than one button, you can determine which button the operator clicked by comparing the returned value against the corresponding constants.

## **SHOW\_ALERT Function (continued)**

### **Example**

A trigger that fires when the user attempts to delete a record might invoke the alert shown in the slide to obtain confirmation. If the operator selects Yes, then the DELETE\_RECORD built-in is called to delete the current record from the block.

```
IF SHOW_ALERT('del_check') = ALERT_BUTTON1 THEN  
    DELETE_RECORD;  
END IF;
```

Oracle Internal & OAI Use Only

# Directing Errors to an Alert

```
PROCEDURE Alert_On_Failure IS
    n NUMBER;
BEGIN
    SET_ALERT_PROPERTY('error_alert',
        ALERT_MESSAGE_TEXT,ERROR_TYPE|||
        '-'||TO_CHAR(ERROR_CODE)|||
        ': '||ERROR_TEXT);
    n := SHOW_ALERT('error_alert');
END;
```



Copyright © 2006, Oracle. All rights reserved.

## Directing Errors to an Alert

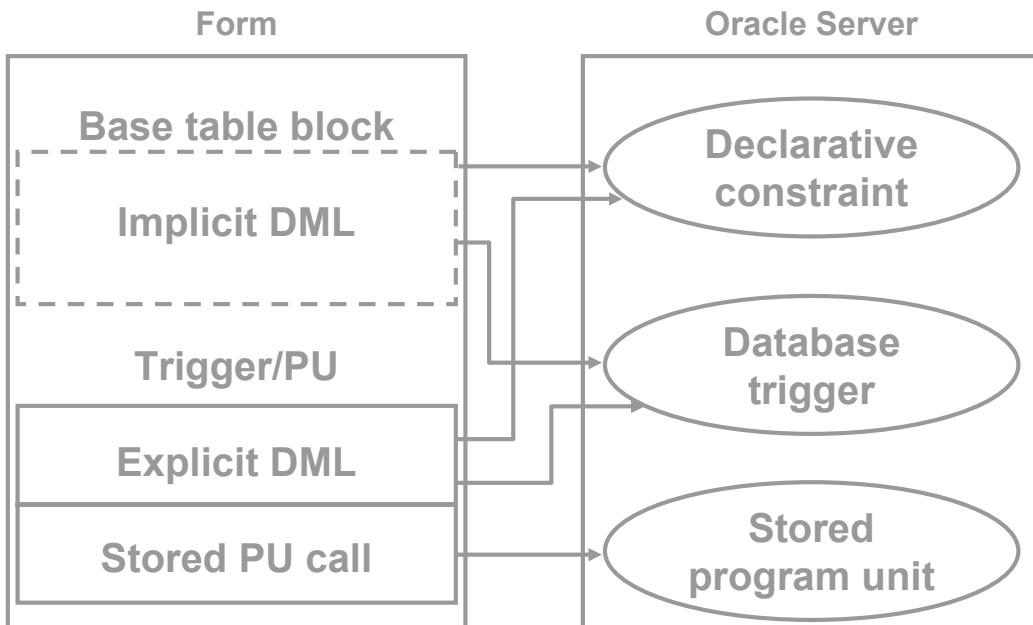
You may want to display errors automatically in an alert, through an On-Error trigger. The built-in functions that return error information, such as ERROR\_TEXT, can be used in the SET\_ALERT\_PROPERTY procedure, to construct the alert message for display.

**Example:** The following user-named procedure can be called when the last form action was unsuccessful. The procedure fails the calling trigger and displays Error\_Alert containing the error information.

```
PROCEDURE alert_on_failure IS
    n NUMBER;
BEGIN
    SET_ALERT_PROPERTY('error_alert',
        ALERT_MESSAGE_TEXT,
        ERROR_TYPE||'-'||TO_CHAR(ERROR_CODE)||': '|||
        ERROR_TEXT);
    n := SHOW_ALERT('error_alert');
END;
```

**Note:** If you want the trigger to fail, include a call to RAISE form\_trigger\_failure.

# Causes of Oracle Server Errors



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Handling Errors Raised by the Oracle Database Server

Oracle server errors can occur for many different reasons, such as violating a declarative constraint or encountering a stored program unit error. You should know how to handle errors that may occur in different situations.

### Causes of Oracle Server Errors

Cause	Error Message
Declarative constraint	Causes predefined error message
Database trigger	Error message specified in RAISE_APPLICATION_ERROR
Stored program unit	Error message specified in RAISE_APPLICATION_ERROR

## Handling Errors Raised by the Oracle Database Server (continued)

### Types of DML Statements

Declarative-constraint violations and firing of database triggers are in turn caused by DML statements. For error-handling purposes, you must distinguish between the following two types of DML statements:

Type	Description
Implicit DML	DML statements that are associated with base table blocks. Implicit DML is also called base table DML. By default, Forms constructs and issues these DML statements.
Explicit DML	DML statements that a developer explicitly codes in triggers or program units.

### FRM-Error Messages Caused by Implicit DML Errors

If an implicit DML statement causes an Oracle server error, Forms displays one of these FRM-error messages:

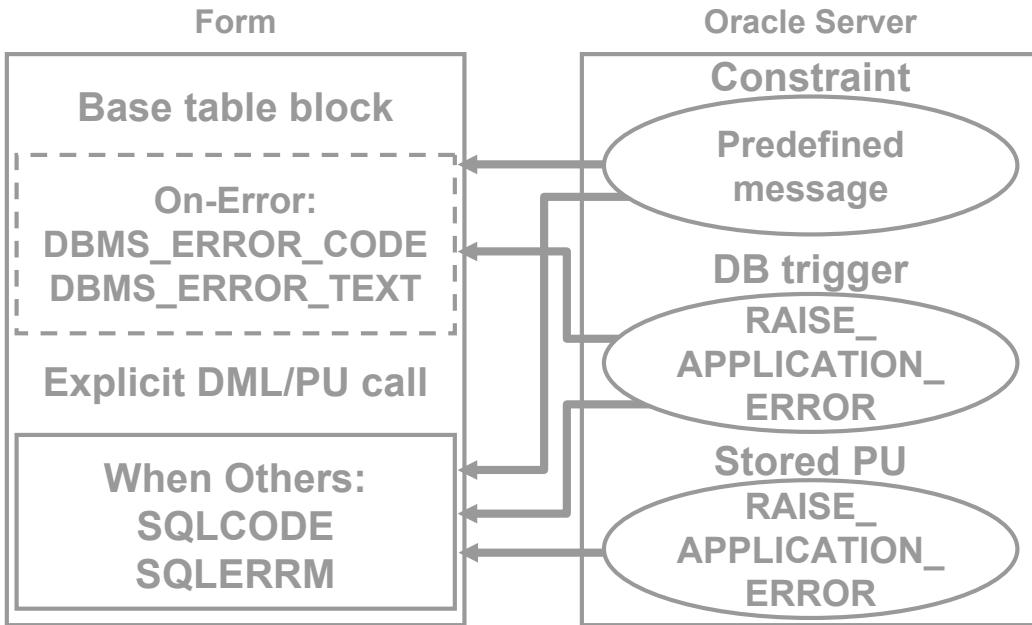
- FRM-40508: ORACLE error: unable to INSERT record.
- FRM-40509: ORACLE error: unable to UPDATE record.
- FRM-40510: ORACLE error: unable to DELETE record.

You can use `ERROR_CODE` to trap these errors in an On-Error trigger and then use `DBMS_ERROR_CODE` and `DBMS_ERROR_TEXT` to determine the ORA-error code and message.

### FRM-Error Messages with Web-Deployed Forms

Users may receive a generic FRM-99999 error. You can obtain meaningful information about this error from the JInitiator Control Panel.

# Trapping Server Errors



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## How to Trap Different Types of Oracle Database Server Errors

Type	Error Handling
Implicit DML	Use the Forms built-ins DBMS_ERROR_CODE and DBMS_ERROR_TEXT in an On-Error trigger.
Explicit DML	Use the PL/SQL functions SQLCODE and SQLERRM in a WHEN OTHERS exception handler of the trigger or program that issued the DML statements.

Declarative-constraint violations and database triggers may be caused by both implicit DML and explicit DML. Stored program units are always called explicitly from a trigger or program unit.

### Technical Note

The values of DBMS\_ERROR\_CODE and DBMS\_ERROR\_TEXT are the same as what a user would see after selecting Help > Display Error; the values are not automatically reset following successful execution.

# Summary

In this lesson, you should have learned:

- **Forms displays messages at run time to inform the operator of events that occur in the session**
- **You can use FORM\_SUCCESS to test for run-time failure of built-ins**
- **There are four types of Forms messages:**
  - Informative
  - Error
  - Working
  - Application

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Summary

In this lesson, you should have learned how to intercept system messages, and how to replace them with the ones that are more suitable for your application. You also learned how to build customized alerts for communicating with operators.

- Test for failure of built-ins by using the FORM\_SUCCESS built-in function or other built-in functions.
- Forms messages can be either Informative, Error, Working, or Application messages.

# Summary

In this lesson, you should have learned:

- You can control system messages with built-ins and triggers:
  - MESSAGE\_LEVEL
  - SUPPRESS\_WORKING
  - On-[Error | Message] triggers
  - [ERROR | MESSAGE]\_[CODE | TEXT | TYPE]
- About the types of alerts: Stop, Caution, Note
- About alert built-ins:
  - SHOW\_ALERT
  - SET\_ALERT\_PROPERTY
  - SET\_ALERT\_BUTTON\_PROPERTY

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Summary (continued)

- Set system variables to suppress system messages:
  - Assign a value to MESSAGE\_LEVEL to specify that only messages above a specific severity level are to be used by the form.
  - Assign a value of True to SUPPRESS\_WORKING to suppress all working messages.
- On-Error and On-Message triggers intercept system error messages and informative system messages.
- You can use the built-ins ERROR\_CODE, ERROR\_TEXT, ERROR\_TYPE, MESSAGE\_CODE, MESSAGE\_TEXT, or MESSAGE\_TYPE to obtain information about the number, text, and type of errors and messages.
- Alert types: Stop, Caution, and Note
- Up to three buttons are available for response (NULL indicates no button).
- Display alerts and change alert messages at run time with SHOW\_ALERT and SET\_ALERT\_PROPERTY.

## Summary

In this lesson, you should have learned:

- To handle database server errors:
  - Implicit DML: Use `DBMS_ERROR_CODE` and `DBMS_ERROR_TEXT` in On-Error trigger
  - Explicit DML: Use `SQLCODE` and `SQLERRM` in `WHEN OTHERS` exception handler



Copyright © 2006, Oracle. All rights reserved.

### Summary (continued)

- Intercept and handle server-side errors.

## Practice 17: Overview

This practice covers the following topics:

- Using an alert to inform the operator that the customer's credit limit has been exceeded
- Using a generic alert to ask the operator to confirm that the form should terminate

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Practice 17: Overview

In this practice, you create some alerts. These include a general alert for questions and a specific alert that is customized for credit limit.

- Using an alert to inform the operator that the customer's credit limit has been exceeded
- Using a generic alert to ask the operator to confirm that the form should terminate

**Note:** For solutions to this practice, see Practice 17 in Appendix A, "Practice Solutions."

## **Practice 17**

1. Create an alert in CUSTGXX called Credit\_Limit\_Alert with one OK button. The message should read “This customer’s current orders exceed the new credit limit”.
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.
3. Save and compile the form. Click Run Form to run the form and test the changes.
4. Create a generic alert in ORDGXX called Question\_Alert that allows Yes and No replies.
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.)
6. Save and compile the form. Click Run Form to run the form and test the changes.

Oracle Internal & OAI Use Only

# 18

## Query Triggers

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

Oracle Internal & OAI Use Only

# Objectives

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

- Explain the processes involved in querying a data block
- Describe query triggers and their scope
- Write triggers to screen query conditions
- Write triggers to supplement query results
- Control trigger action based on the form's query status



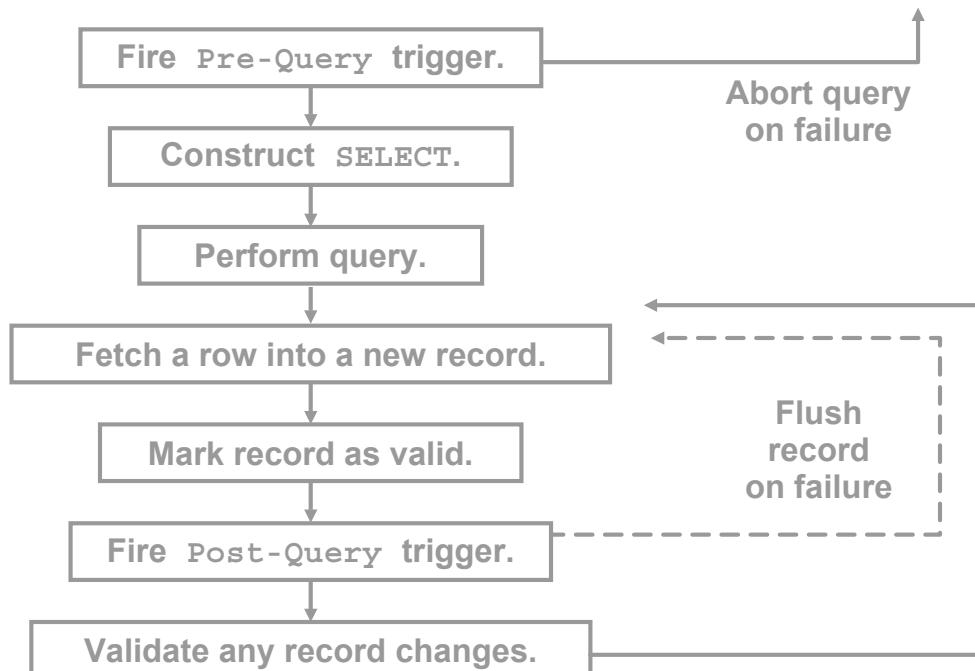
Copyright © 2006, Oracle. All rights reserved.

## Introduction

### Overview

In this lesson, you learn how to control events associated with queries on base-table data blocks. You can customize the query process as necessary, and supplement the results returned by a query.

# Query Processing: Overview



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Query Processing: Overview

Generally, triggers are associated with a query in one of two ways:

- A trigger fires due to the query process itself (for example, Pre-Query and Post-Query).
- An event can fire a trigger in Enter-Query mode, if the Fire in Enter-Query Mode property of the associated trigger is enabled.

The query triggers, Pre-Query and Post-Query, fire due to the query process itself, and are usually defined on the block where the query takes place.

With these triggers, you can add to the normal Forms processing of records, or possibly abandon a query before it is even executed, if the required conditions are not suitable.

## Forms Query Processing

When a query is initiated on a data block, either by the operator or by a built-in subprogram, the following major events take place:

1. Forms fires the Pre-Query trigger if defined.
2. If the Pre-Query succeeds, Forms constructs the query SELECT statement, based on any existing criteria in the block (either entered by the operator or by Pre-Query).

## Query Processing: Overview (continued)

3. The query is performed.
4. Forms fetches the column values of a row into the base-table items of a new record in the block.
5. The record is marked Valid.
6. Forms fires the Post-Query trigger. If it fails, this record is flushed from the block.
7. Forms performs item and record validation if the record has changed (due to a trigger).
8. Steps 4 through 7 are repeated for any remaining records of this query.

Oracle Internal & OAI Use Only

## SELECT Statements Issued During Query Processing

```
SELECT      base_column, ..., ROWID
INTO        :base_item, ..., :ROWID
FROM        base_table
WHERE       (default_where_clause OR
              onetime_where_clause)
AND         (example_record_conditions)
AND         (query_where_conditions)
ORDER BY    default_order_by_clause |
              query_where_order_by
```

Slightly different for COUNT

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## SELECT Statements Issued During Query Processing

If you have not altered default query processing, Forms issues a SELECT statement when you want to retrieve or count records.

```
SELECT      base_column,      base_column, ..., ROWID
INTO        :base_item,       :base_item, ..., :ROWID
FROM        base_table
WHERE       (default_where_clause OR onetime_where_clause)
AND         (example_record_conditions)
AND         (query_where_conditions)
ORDER BY    default_order_by_clause | query_where_order_by

SELECT      COUNT(*)
FROM        base_table
WHERE       (default_where_clause OR onetime_where_clause)
AND         (example_record_conditions)
AND         (query_where_conditions)
ORDER BY    default_order_by_clause | query_where_order_by
```

## **SELECT Statements Issued During Query Processing (continued)**

**Note:** The vertical bar ( | ) in the ORDER BY clause indicates that either of the two possibilities can be present. Forms retrieves the ROWID only when the Key Mode block property is set to Unique (the default). The entire WHERE clause is optional. The ORDER BY clause is also optional.

If you want to count records that satisfy criteria specified in the Query/Where dialog box, then enter one or more variables in the example record and press Count Query.

Oracle Internal & OAI Use Only

## WHERE Clause

- **Four sources for the WHERE clause:**
  - WHERE Clause block property
  - ONETIME\_WHERE block property
  - Example Record
  - Query/Where dialog box
- **WHERE clauses are combined by the AND operator, except that WHERE and ONETIME\_WHERE are mutually exclusive:**
  - ONETIME\_WHERE is used the first time the query executes.
  - WHERE is used for subsequent executions of the query.

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### WHERE and ORDER BY Clauses

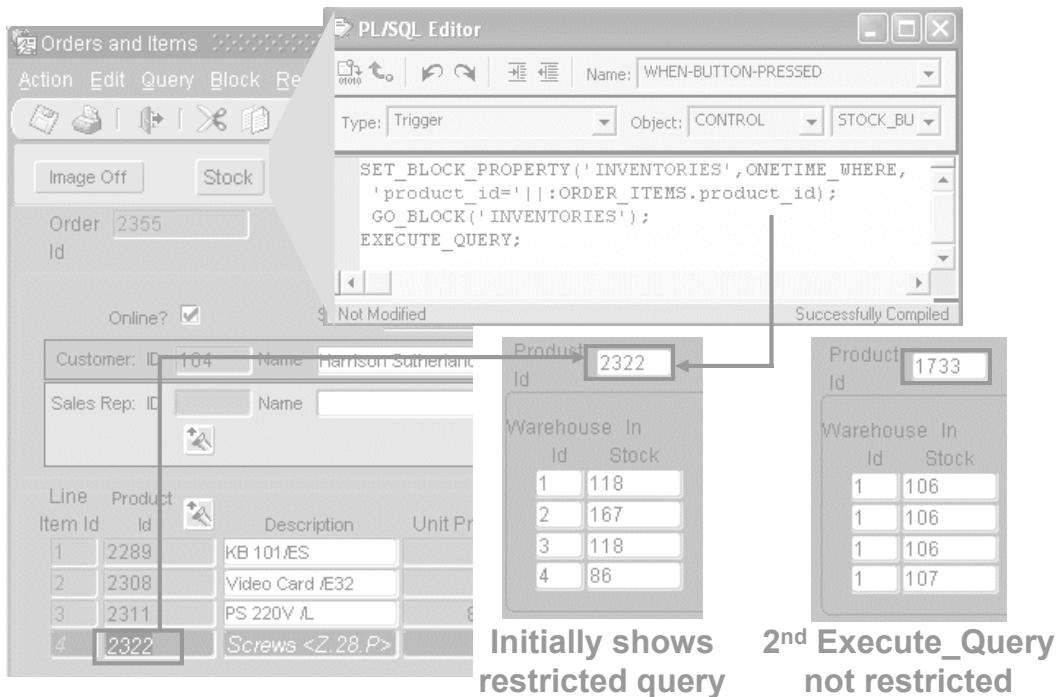
The WHERE and ORDER BY clauses of a default base table SELECT statement are derived from several sources. It is important to know how different sources interact.

#### Four Sources for the WHERE Clause

- The WHERE Clause block property (set in Forms Builder, or by setting the DEFAULT\_WHERE\_CLAUSE property programmatically)
- The ONETIME\_WHERE block property (set programmatically)
- Example Record
- Query/Where dialog box

If more than one source is present, the different conditions will all be used and linked with an AND operator. If the WHERE clause and the ONETIME\_WHERE clause are present, only one is used: the ONETIME\_WHERE clause for the first execution of the query, and the WHERE clause for subsequent executions.

## ONETIME\_WHERE Property



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### WHERE and ORDER BY Clauses (continued)

#### ONETIME\_WHERE Property

For instances where you want to restrict the query only once, you can programmatically set the ONETIME\_WHERE property on a block. This specifies a WHERE clause for the block that will be in effect only for the first query issued on the block after setting that property.

#### Example

From the ORDER\_ITEMS block, you want to display the INVENTORIES block, which is on a separate window. When the block is initially displayed, it should present information about the stock of the product selected in the ORDER\_ITEMS block. However, after this initial information is displayed, you want users to be able to query the stock of any product. You accomplish this by coding the Stock button to set the ONETIME\_WHERE property:

```
Set_Block_Property('INVENTORIES', ONETIME_WHERE,  
'product_id = '||:ORDER_ITEMS.PRODUCT_ID);  
Go_block('INVENTORIES');  
Execute_Query;
```

## ORDER BY Clause

- Two sources for the ORDER BY clause are:
  - ORDER BY Clause block property
  - Query/Where dialog box
- The second source for the ORDER BY clause overrides the first one.

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### WHERE and ORDER BY Clauses (continued)

#### Two Sources for the ORDER BY Clause

- ORDER BY Clause block property
- Query/Where dialog box

An ORDER BY clause specified in the Query/Where dialog box overrides the value of the ORDER BY Clause block property.

**Note:** You can use the SET\_BLOCK\_PROPERTY built-in to change the WHERE Clause and ORDER BY Clause block properties at run time.

# Writing Query Triggers: Pre-Query Trigger

- **Defined at block level**
- **Fires once, before query is performed**

```
IF    TO_CHAR (:ORDERS.ORDER_ID) ||  
      TO_CHAR (:ORDERS.CUSTOMER_ID)  
IS NULL THEN  
    MESSAGE('You must query by  
          Order ID or Customer ID');  
    RAISE form_trigger_failure;  
END IF;
```



Copyright © 2006, Oracle. All rights reserved.

## Writing Query Triggers: Pre-Query Trigger

You must define this trigger at block level or above. It fires for either a global or restricted query, just before Forms executes the query. You can use Pre-Query to:

- Test the operator's query conditions, and to fail the query process if the conditions are not satisfactory for the application
- Add criteria for the query by assigning values to base-table items

### Example

The Pre-Query trigger on the ORDERS block shown in the slide permits queries only if there is a restriction on either the Order ID or Customer ID. This prevents very large queries.

**Note:** Pre-Query is useful for assigning values passed from other Oracle Forms Developer modules, so that the query is related to data elsewhere in the session. You will learn how to do this in the lesson titled “Introducing Multiple Form Applications.”

## Writing Query Triggers: Post-Query Trigger

- Fires for each fetched record (except during array processing)
- Is used to populate nondatabase items and calculate statistics

```
SELECT COUNT(order_id)
INTO :ORDERS.lineitem_count
FROM ORDER_ITEMS
WHERE order_id = :ORDERS.order_id;
```



Copyright © 2006, Oracle. All rights reserved.

### Writing Query Triggers: Post-Query Trigger

This trigger is defined at block level or above. Post-Query fires for each record that is fetched into the block as a result of a query. Note that the trigger fires only on the initial fetch of a record, not when a record is subsequently scrolled back into view a second or third time.

Use Post-Query as follows:

- To populate nondatabase items as records are returned from a query
- To calculate statistics

#### Example

The Post-Query trigger on the ORDERS block, shown in the slide, selects the total count of line items for the current Order, and displays this number as a summary value in the non-base-table item :Lineitem\_count.

# Writing Query Triggers: Using SELECT Statements in Triggers

- **Forms Builder variables are preceded by a colon.**
- **The query must return one row for success.**
- **Code exception handlers.**
- **The INTO clause is mandatory, with a variable for each selected column or expression.**
- **ORDER BY is not relevant.**

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Writing Query Triggers

### Using SELECT Statements in Triggers

The previous trigger example populates the Lineitem\_Count item through the INTO clause. Again, colons are required in front of Forms Builder variables to distinguish them from PL/SQL variables and database columns.

Here is a reminder of some other rules regarding SELECT statements in PL/SQL:

- A single row must be returned from the query, or else an exception is raised that terminates the normal executable part of the PL/SQL block. You usually want to match a form value with a unique column value in your restriction.
- Code exception handlers in your PL/SQL block to deal with possible exceptions raised by SELECT statements.
- The INTO clause is mandatory, and must define a receiving variable for each selected column or expression. You can use PL/SQL variables, form items or global variables in the INTO clause.
- ORDER BY and other clauses that control multiple-row queries are not relevant (unless they are part of an Explicit Cursor definition).

# Query Array Processing

- Reduces network traffic
- Enables Query Array processing:
  - Enable Array Processing option
  - Set Query Array Size property
- Query Array Size property
- Query All Records property

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Query Array Processing

The default behavior of Forms is to process records one at a time. With array processing, a structure (array) containing multiple records is sent to or returned from the server for processing.

Forms supports both array fetch processing and array DML processing. For both querying and DML operations, you can determine the array size to optimize performance for your needs. This lesson focuses on array query processing.

You can enable array processing for queries by:

- Setting preferences:
  - Select Edit > Preferences.
  - Click the Runtime tab.
  - Select the Array Processing check box.
- Setting block properties:
  - In the Object Navigator, double-click the node of the block to display the Property Palette.
  - Under the Records category, set the Query Array Size property to a number that represents the number of records in the array for array processing.

## Query Array Processing (continued)

**Query Array Size Property:** This property specifies the maximum number of records that Forms should fetch from the database at one time. If set to zero, the query array size defaults to the number of records displayed in the block.

A size of 1 provides the fastest perceived response time, because Forms fetches and displays only one record at a time. By contrast, a size of 10 fetches up to ten records before displaying any of them; however, the larger size reduces overall processing time by making fewer calls to the database for records.

**Query All Records Property:** This property specifies whether all the records matching the query criteria should be fetched into the data block when a query is executed.

- **Yes:** Fetches all records from query
- **No:** Fetches the number of records specified by the Query Array Size block property

# Coding Triggers for Enter-Query Mode

- Some triggers may fire in Enter-Query mode.
- Set the Fire in Enter-Query Mode property.
- Test mode during execution with :SYSTEM.MODE
  - NORMAL
  - ENTER-QUERY
  - QUERY

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Coding Triggers for Enter-Query Mode

Some triggers that fire when the form is in Normal mode (during data entry and saving) may also be fired in Enter-Query mode. You need to consider the trigger type and actions in these cases.

### Fire in Enter-Query Mode Property

To create a trigger that fires in Enter-Query mode, in its Property Palette set the Fire in Enter-Query Mode property to Yes. This property determines whether Forms fires a trigger if the associated event occurs in Enter-Query mode. Not all triggers can do this; consult Forms Builder online Help, which lists each trigger and whether this property can be set.

By default, the Fire in Enter-Query Mode property is set to Yes for triggers that accept this. Set it to No in the Property Palette if you want the trigger to fire only in Normal mode.

## Coding Triggers for Enter-Query Mode (continued)

### Example

If you provide a button for the operator to invoke an LOV, and the LOV is required to help with query criteria as well as data entry, then the When-Button-Pressed trigger should fire in both modes. This trigger has Fire in Enter-Query Mode set to Yes (default for this trigger type):

```
IF SHOW_LOV('Customers') THEN
    MESSAGE('Selection successful');
END IF;
```

Oracle Internal & OAI Use Only

## Coding Triggers for Enter-Query Mode

- **Example:**

```
IF :SYSTEM.MODE = 'NORMAL'  
THEN ENTER_QUERY;  
ELSE EXECUTE_QUERY;  
END IF;
```

- Some built-ins are illegal.
- Consult online Help.
- You cannot navigate to another record in the current form.



Copyright © 2006, Oracle. All rights reserved.

### Coding Triggers for Enter-Query Mode (continued)

#### Finding Out the Current Mode

When a trigger fires in both Enter-Query mode and Normal modes, you may need to know the current mode at execution time for the following reasons:

- Your trigger needs to perform different actions depending on the mode.
- Some built-in subprograms cannot be used in Enter-Query mode.

The read-only system variable, MODE, stores the current mode of the form. Its value (always in uppercase) is one of the following:

Value of :SYSTEM.MODE	Definition
NORMAL	Form is in Normal processing mode.
ENTER-QUERY	Form is in Enter-Query mode.
QUERY	Form is in Fetch-processing mode, meaning that Forms is currently performing a fetch. (For example, this value always occurs in a Post-Query trigger.)

## Coding Triggers for Enter-Query Mode (continued)

### Example

Consider the following When-Button-Pressed trigger for the Query button.

If the operator clicks the button in Normal mode, then the trigger places the form in Enter-Query mode (using the ENTER\_QUERY built-in). Otherwise, if already in Enter-Query mode, the button executes the query (using the EXECUTE\_QUERY built-in).

```
IF      :SYSTEM.MODE = 'NORMAL' THEN
    ENTER_QUERY;
ELSE
    EXECUTE_QUERY;
END IF;
```

### Using Built-ins in Enter-Query Mode

Some built-in subprograms are illegal if a trigger is executed in Enter-Query mode. Again, consult the Forms Builder online Help, which specifies whether an individual built-in can be used in this mode.

One general restriction is that in Enter-Query mode you cannot navigate to another record in the current form. So any built-in that would potentially enable this is illegal. These include GO\_BLOCK, NEXT\_BLOCK, PREVIOUS\_BLOCK, GO\_RECORD, NEXT\_RECORD, PREVIOUS\_RECORD, UP, DOWN, OPEN\_FORM, and so on.

# Overriding Default Query Processing

## Additional Transactional Triggers for Query Processing

Trigger	Do-the-Right-Thing Built-in
On-Close	
On-Count	COUNT_QUERY
On-Fetch	FETCH_RECORDS
Pre-Select	
On-Select	SELECT_RECORDS
Post-Select	

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Overriding Default Query Processing

You can use certain transactional triggers to replace default commit processing. Some of the transactional triggers can also be used to replace default query processing.

You can call “Do-the-right-thing” built-ins from transactional triggers to augment default query processing. The “Do-the-right-thing” built-ins perform the same actions as the default processing would. You can then supplement the default processing with your own code.

# Overriding Default Query Processing

- **On-Fetch continues to fire until:**
  - It fires without executing `CREATE_QuERIED_RECORD`
  - The query is closed by the user or by `ABORT_QUERY`
  - It raises `FORM_TRIGGER_FAILURE`
- **On-Select replaces open cursor, parse, and execute phases.**

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Overriding Default Query Processing (continued)

### Using Transactional Triggers for Query Processing

Transactional triggers for query processing are primarily intended to access certain data sources other than Oracle. However, you can also use these triggers to implement special functionality by augmenting default query processing against an Oracle database.

## Overriding Default Query Processing (continued)

Transactional Triggers for query processing have the following characteristics:

Trigger	Characteristic
On-Close	Fires when Forms closes a query (It augments, rather than replaces, default processing.)
On-Count	Fires when Forms would usually perform default Count Query processing to determine the number of rows that match the query conditions
On-Fetch	Fires when Forms performs a fetch for a set of rows (You can use the CREATE_QUERIED_RECORD built-in to create queried records if you want to replace default fetch processing.) The trigger continues to fire until: <ul style="list-style-type: none"><li>• No queried records are created during a single execution of the trigger</li><li>• The query is closed by the user or by the ABORT_QUERY built-in executed from another trigger</li><li>• The trigger raises FORM_TRIGGER_FAILURE</li></ul>
Pre-Select	Fires after Forms has constructed the block SELECT statement based on the query conditions, but before it issues this statement
On-Select	Fires when Forms would usually issue the block SELECT statement (The trigger replaces the open cursor, parse, and execute phases of a query.)
Post-Select	Fires after Forms has constructed and issued the block SELECT statement, but before it fetches the records

# Obtaining Query Information at Run Time

- **SYSTEM.MODE**
- **SYSTEM.LAST\_QUERY**
  - Contains bind variables (`ORD_ID = :1`) before `SELECT_RECORDS`
  - Contains actual values (`ORD_ID = 102`) after `SELECT_RECORDS`

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Obtaining Query Information at Run Time

You can use system variables and built-ins to obtain information about queries.

### Using **SYSTEM.MODE**

Use the **SYSTEM.MODE** system variable to obtain the form mode. The three values are **NORMAL**, **ENTER\_QUERY**, and **QUERY**. This system variable was discussed previously in this lesson.

### Using **SYSTEM.LAST\_QUERY**

Use **SYSTEM.LAST\_QUERY** to obtain the text of the base-table `SELECT` statement that was last executed by Forms. If a user has entered query conditions in the Example Record, the exact form of the `SELECT` statement depends on when this system variable is used.

If the system variable is used before Forms has implicitly executed the `SELECT_RECORDS` built-in, the `SELECT` statement contains bind variables (for example, `ORDER_ID= :1`). If used after Forms has implicitly executed the `SELECT_RECORDS` built-in, the `SELECT` statement contains the actual search values (for example, `ORDER_ID=102`). The system variable contains bind variables during the Pre-Select trigger and actual search values during the Post-Select trigger.

Unlike most system variables, **SYSTEM.LAST\_QUERY** may contain a mixture of uppercase and lowercase letters.

# Obtaining Query Information at Run Time

- **GET\_BLOCK\_PROPERTY**  
**SET\_BLOCK\_PROPERTY**
  - **Get and set:**  
DEFAULT WHERE  
ONETIME WHERE  
ORDER BY  
QUERY ALLOWED  
QUERY HITS
  - **Get only:**  
QUERY OPTIONS  
RECORDS TO FETCH

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Obtaining Query Information at Run Time (continued)

### Using **GET\_BLOCK\_PROPERTY** and **SET\_BLOCK\_PROPERTY**

The following block properties may be useful for obtaining query information. Only the properties marked with an asterisk can be set.

- DEFAULT WHERE (\*)
- ONETIME WHERE (\*)
- ORDER BY (\*)
- QUERY ALLOWED (\*)
- QUERY HITS (\*)
- QUERY OPTIONS
- RECORDS TO FETCH

# Obtaining Query Information at Run Time

- `GET_ITEM_PROPERTY`
- `SET_ITEM_PROPERTY`
  - **Get and set:**  
`CASE_INSENSITIVE_QUERY`  
`QUERYABLE`  
`QUERY_ONLY`
  - **Get only:**  
`QUERY_LENGTH`

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Obtaining Query Information at Run Time (continued)

### Using `GET_ITEM_PROPERTY` and `SET_ITEM_PROPERTY`

The following item properties may be useful for getting query information. Only the properties marked with an asterisk can be set.

- `CASE_INSENSITIVE_QUERY` (\*)
- `QUERYABLE` (\*)
- `QUERY_ONLY` (\*)
- `QUERY_LENGTH`

# Summary

In this lesson, you should have learned that:

- Query processing includes the following steps:
  1. Pre-Query trigger fires
  2. SELECT statement constructed
  3. Query performed
  4. Record fetched into block
  5. Record marked Valid
  6. Post-Query trigger fires
  7. Item and record validation if the record has changed (due to a trigger)
  8. Steps 4 through 7 repeat till all fetched

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Summary

In this lesson, you learned how to control the events associated with queries on base-table blocks.

- **The query process:** Before beginning the query, the Pre-Query trigger fires once for each query. Then the query statement is constructed and executed. For each record retrieved by the query, the record is fetched into the block and marked as valid, the Post-Query trigger fires for that record, and item and record validation occurs if a trigger has changed the record.

# Summary

In this lesson, you should have learned that:

- The query triggers, which must be defined at block or form level, are:
  - Pre-Query: Use to screen query conditions (set ONETIME\_WHERE or DEFAULT\_WHERE properties, or assign values to use as query criteria)
  - Post-Query: Use to supplement query results (populate non-base-table items, perform calculations)
- You can use transactional triggers to override default query processing
- You can control trigger action based on the form's query status by checking SYSTEM.MODE values: NORMAL, ENTER-QUERY, or QUERY

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Summary (continued)

- The Pre-Query trigger fires before the query executes. This trigger is defined at the block level or above. Use the Pre-Query trigger to check or modify query conditions.
- The Post-Query trigger fires as each record is fetched (except array processing). This trigger is defined at the block level or above. Use the Post-Query trigger to perform calculations and populate additional items.
- Some triggers can fire in both Normal and Enter-Query modes.
  - Use SYSTEM.MODE to test the current mode.
  - Some built-ins are illegal in Enter-Query mode.
- Override default query processing by using transactional triggers; to replace the default functionally, use “Do-the-right-thing” built-ins.
- Obtain query information at run-time by using:
  - SYSTEM.MODE, SYSTEM.LAST\_QUERY
  - Some properties of GET/SET\_BLOCK\_PROPERTY and GET/SET\_ITEM\_PROPERTY

## Practice 18: Overview

This practice covers the following topics:

- Populating customer names and sales representative names for each row of the ORDERS block
- Populating descriptions for each row of the ORDER\_ITEMS block
- Restricting the query on the INVENTORIES block for only the first query on that block
- Disabling the effects of the Exit button and changing a radio group in Enter-Query mode
- Adding two check boxes to enable case-sensitive and exact match query

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Practice 18: Overview

In this practice, you create two query triggers to populate non-base-table items. You will also change the default query interface to enable case-sensitive and exact match query.

- Populating customer names and sales representative names for each row of the ORDERS block
- Populating descriptions for each row of the ORDER\_ITEMS block
- Restricting the query on the INVENTORIES block for only the first query on that block
- Disabling the effect of the Exit button and changing a radio group in Enter-Query mode
- Adding two check boxes to the Customers form to enable case-sensitive and exact match query

**Note:** For solutions to this practice, see Practice 18 in Appendix A, “Practice Solutions.”

## Practice 18

1. In the ORDGXX form, write a trigger that populates 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.
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.
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.
4. Ensure that the Exit\_Button has no effect in Enter-Query mode.
5. Click Run Form to run the form and test it.
6. Open the CUSTGXX 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.  
**Note:** If the background color looks different from that of the canvas, click in the Background Color property and click Inherit on the Property Palette toolbar.
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 When-Checkbox-Changed Trigger.  
**Note:** If the background color looks different from that of the canvas, click in the Background Color property and click Inherit on the Property Palette toolbar.
8. Modify the properties of the CONTROL block so that the check boxes can be checked or unchecked at run time.  
**Hint:** The CONTROL block contains a single new record.
9. Ensure that the When-Radio-Changed trigger for the Credit\_Limit item does not fire when in Enter-Query mode.
10. Click Run Form to run the form and test the changes.

# 19

Validation

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

Oracle Internal & OAI Use Only

# Objectives

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

- Explain the effects of the validation unit upon a form
- Control validation:
  - Using object properties
  - Using triggers
  - Using Pluggable Java Components
- Describe how Forms tracks validation status
- Control when validation occurs

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Introduction

### Overview

In this lesson, you will learn how to supplement item validation by using both object properties and triggers. You will also learn to control when validation occurs.

# Validation Process

Forms validates at the following levels:



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Validation Process

### Validation Levels

Forms performs a validation process at several levels to ensure that records and individual values follow appropriate rules. If validation fails, then control is passed back to the appropriate level, so that the operator can make corrections. Validation occurs at:

- **Item level:** Forms records a status for each item to determine whether it is currently valid. If an item has been changed and is not yet marked as valid, then Forms first performs standard validation checks to ensure that the value conforms to the item's properties. These checks are carried out before firing any When-Validate-Item triggers that you have defined. Standard checks include the following:
  - Format mask
  - Required (if so, then is the item null?)
  - Data type
  - Range (Lowest-Highest Allowed Value)
  - Validate from List (see later in this lesson)

# Validation Process

## Validation occurs when:

- **The Enter key is pressed or the ENTER built-in procedure is run**
- **The operator or trigger leaves the validation unit (includes a Commit)**

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Validation Process (continued)

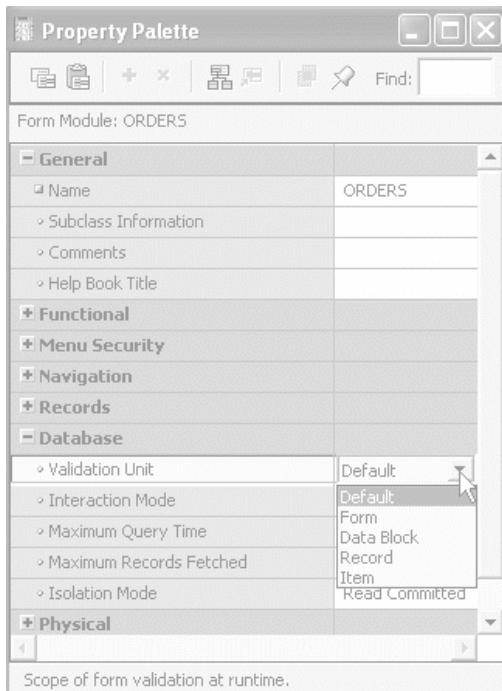
- **Record level:** After leaving a record, Forms checks to see whether the record is valid. If not, then the status of each item in the record is checked, and a When-Validate-Record trigger is then fired, if present. When the record passes these checks, it is set to valid.
- **Block or form level:** At block or form level, all records below that level are validated. For example, if you commit (save) changes in the form, then all records in the form are validated, unless you have suppressed this action.

## When Does Validation Occur?

Forms carries out validation for the validation unit under the following conditions:

- The Enter key is pressed or the ENTER built-in procedure is run. The purpose of ENTER built-in is to force validation immediately.  
**Note:** The ENTER action is not necessarily mapped to the key that is physically labeled Enter.
- The operator or a trigger navigates out of the validation unit. This includes when changes are committed. The default validation unit is item, but can also be set to record, block, or form by the designer. The validation unit is discussed in the next section.

# Controlling Validation Using Properties: Validation Unit



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Using Object Properties to Control Validation

You can control when and how validation occurs in a form, even without triggers. Do this by setting properties for the form and for individual items within it.

### Validation Unit

The validation unit defines the maximum amount of data an operator can enter in the form before Forms initiates validation. Validation unit is a property of the form module, and it can be set in the Property Palette to any of the following:

- Default
- Item
- Record
- Block
- Form

The default setting is item level. The default setting is usually chosen.

In practice, an item-level validation unit means that Forms validates changes when an operator navigates out of a changed item. This way, standard validation checks and firing the When-Validate-Item trigger of that item can be done immediately. As a result, operators are aware of validation failure as soon as they attempt to leave the item.

## **Using Object Properties to Control Validation (continued)**

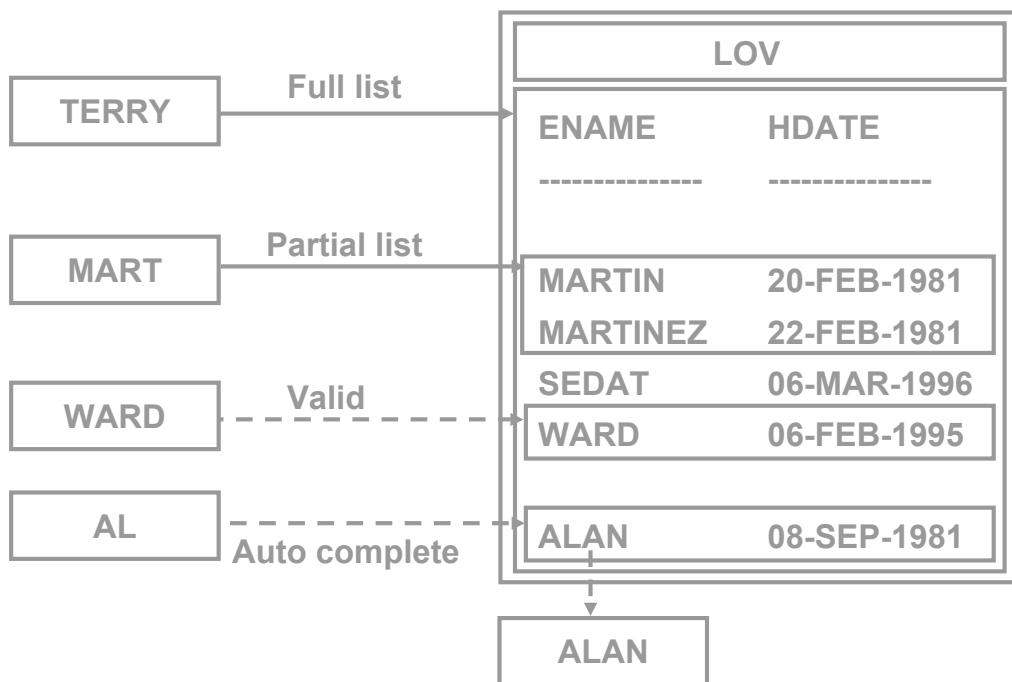
At higher validation units (record, block, or form level), the above checks are postponed until navigation moves out of that unit. All outstanding items and records are validated together, including the firing of When-Validate-Item and When-Validate-Record triggers.

You might set a validation unit above item level under one of the following conditions:

- Validation involves database references, and you want to postpone traffic until the operator has completed a record (record level).
- The application runs on the Internet and you want to improve performance by reducing round trips to the application server.

Oracle Internal & OAI Use Only

## Controlling Validation Using Properties: Validate from List



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Using LOVs for Validation

When you attach an LOV to a text item by setting the LOV property of the item, you can optionally use the LOV contents to validate data entered in the item.

#### The Validate from List Property

Do this by setting the Validate from List property to Yes for the item. During validation, Forms then automatically uses the item value as a non-case-sensitive search string on the LOV contents. The following events then occur, depending on the circumstances:

- If the value in the text item matches one of the values in the first column of the LOV, validation succeeds, the LOV is not displayed, and processing continues normally.
- If the item's value causes a single record to be found in the LOV, but is a partial value of the LOV value, then the full LOV column value is returned to the item (providing that the item is defined as the return item in the LOV). The item then passes this validation phase.
- If the item value causes multiple records to be found in the LOV, Forms displays the LOV and uses the text item value as the search criteria to automatically reduce the list, so that the operator must choose.
- If no match is found, then the full LOV contents are displayed to the operator.

## Using LOVs for Validation (continued)

**Note:** Make sure that the LOVs you create for validation purposes have the validation column defined first, with a display width greater than 0. You also need to define the Return Item for the LOV column as the item being validated.

For performance reasons, do not use the LOV for Validation property for large LOVs.

Oracle Internal & OAI Use Only

# Controlling Validation by Using Triggers

- **Item level:**  
**When-Validate-Item**
- **Block level:**  
**When-Validate-Record**

```
IF :ORDERS.order_date > SYSDATE THEN
    MESSAGE('Order Date is later than today!');
    RAISE form_trigger_failure;
END IF;
```



Copyright © 2006, Oracle. All rights reserved.

## Controlling Validation by Using Triggers

There are triggers that fire due to validation, which let you add your own customized actions. There are also some built-in subprograms that you can call from triggers that affect validation.

### When-Validate-Item Trigger

You have already used this trigger to add item-level validation. The trigger fires after standard item validation, and input focus is returned to the item if the trigger fails.

#### Example

The When-Validate-Item trigger on ORDERS.Order\_Date shown in the slide, ensures that the Order Date is not later than the current (database) date.

## Controlling Validation by Using Triggers (continued)

### When-Validate-Record Trigger

This trigger fires after standard record-level validation, when the operator has left a new or changed record. Because Forms has already checked that required items for the record are valid, you can use this trigger to perform additional checks that may involve more than one of the record's items, in the order they were entered.

When-Validate-Record must be defined at block level or above.

#### Example

This When-Validate-Record trigger on block ORDER\_ITEMS warns the operator when a line item for a new credit order causes the customer's credit limit to be exceeded.

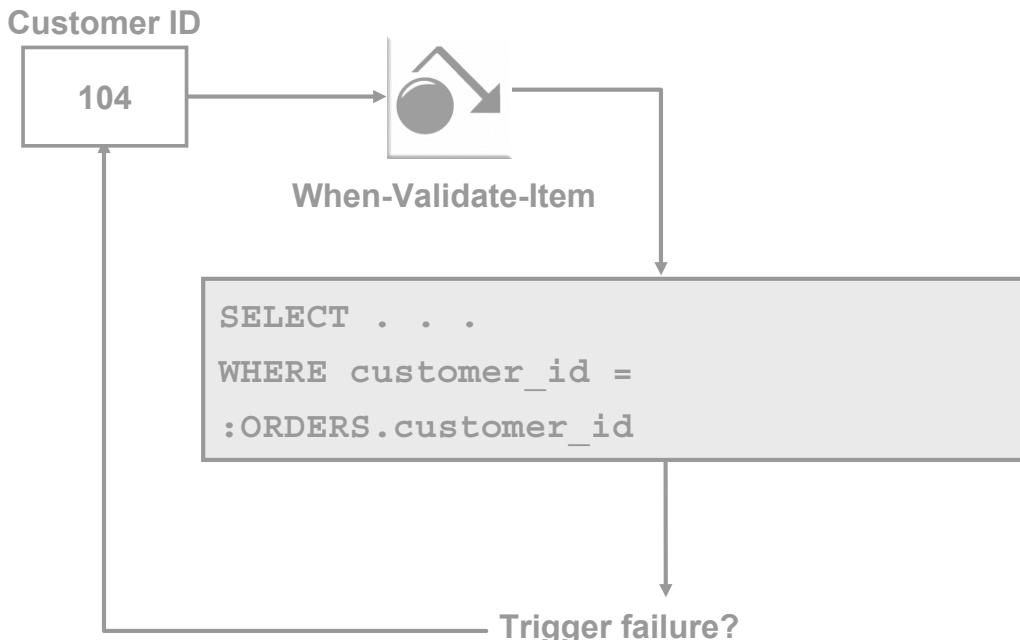
```
DECLARE
    cred_limit number;
    n number;
BEGIN
    -- Order status of 4 is new credit order
    IF :orders.order_status = 4 THEN
        SELECT credit_limit INTO cred_limit FROM customers
            WHERE customer_id = :orders.customer_id;
        IF :control.total > cred_limit THEN
            n := show_alert('credit_limit_alert');
        END IF;
    END IF;
END;
```

**Note:** If you want to stop the operator from navigating out of the item when validation fails, you can raise an exception to fail the trigger:

```
RAISE form_trigger_failure;
```

In the example above, you would include this code immediately after displaying the alert.

## Example: Validating User Input



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Controlling Validation by Using Triggers (continued)

#### Example: Validating User Input

While populating other items, if the user enters an invalid value in the item, a matching row will not be found, and the SELECT statement will cause an exception. The success or failure of the query can, therefore, be used to validate user input.

The exceptions that can occur when a single row is not returned from a SELECT statement in a trigger are:

- NO\_DATA\_FOUND: No rows are returned from the query.
- TOO\_MANY\_ROWS: More than one row is returned from the query.

#### Example

The following When-Validate-Item trigger is again placed on the Customer\_ID item, and returns the name that corresponds to the Customer ID entered by the user.

```
SELECT cust_first_name || ' ' || cust_last_name
INTO :ORDERS.customer_name
FROM customers
WHERE customer_id = :ORDERS.customer_id;
```

## Controlling Validation by Using Triggers (continued)

### Example: Validating User Input (continued)

If the Customer\_ID item contains a value that is not found in the table, the NO\_DATA\_FOUND exception is raised, and the trigger will fail because there is no exception handler to prevent the exception from propagating to the end of the trigger.

**Note:** A failing When-Validate-Item trigger prevents the cursor from leaving the item.

For an unhandled exception, as above, the user receives the message:

FRM-40735: <trigger type> trigger raised unhandled exception  
<exception>

Oracle Internal & OAI Use Only

# Using Client-Side Validation

- **Forms validation:**
  - Occurs on middle tier
  - Involves network traffic
- **Client-side validation:**
  - Improves performance
  - Implemented with PJC

The image contains two side-by-side screenshots of an Oracle Forms application interface. Both screenshots show a grid of data with columns: Line, Product Id, Description, Unit Price, and Quantity.

**Screenshot 1: Using number datatype**

In this screenshot, the 'Quantity' column has a numeric data type. A user has attempted to enter the alphabetic characters 'abcdef' into the fourth row's Quantity field. A validation error message, 'FRM-50016: Legal characters are 0-9 - + E.', is displayed below the grid. An arrow points from the error message to the Quantity field.

**Screenshot 2: Attempt to enter alphabetic characters**

In this screenshot, the 'Quantity' column uses a KeyFilter PJC. The user has attempted to enter the alphabetic character 'A' into the fourth row's Quantity field. A validation error message, 'Enter a numeric value', is displayed below the grid. An arrow points from the error message to the Quantity field.

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Using Client-Side Validation

It is common practice to process input to an item using a When-Validate-Item trigger. The trigger itself is processed on the Forms Services. Even validation that occurs with a format mask on an item involves a trip to the middle tier. In the first example in the slide, the number data type on the Quantity item is not checked until the operator presses [Enter] to send the input to the Forms Services machine, which returns the FRM-50016 error.

You should consider using Pluggable Java Components (PJC)s to replace the default functionality of standard client items, such as text boxes. Then validation of items, such as the date or maximum or minimum values, is contained within an item. This technique opens up opportunities for more complex, application-specific validation, such as automatic formatting of input—for example, telephone numbers with the format (XXX) XXX-XXXX. Even a simple numeric format is enforced instantly, not allowing alphabetic keystrokes to be entered into the item.

This validation is performed on the client without involving a network round trip, thus improving performance. In the second example in the slide, the KeyFilter PJC does not allow the operator to enter an alphabetic character into the Quantity item. The only message that is displayed on the message line is the item's Hint.

## **Using Client-Side Validation (continued)**

Pluggable Java Components are similar to JavaBeans, and in fact, the two terms are often used interchangeably. Although both are Java components that you can use in a form, there are the following differences between them:

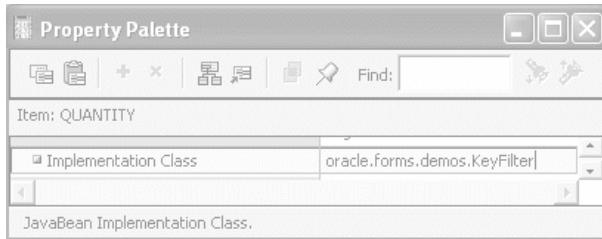
- JavaBeans are implemented in a bean area item, whereas PJC's are implemented in a native Forms item, such as a text item or check box.
- PJC's must always have the implementation class specified in the Property Palette, but JavaBeans may be registered at run time with the FBean package.

Oracle Internal & OAI Use Only

# Using Client-Side Validation

## To use a PJC:

- Set the item's Implementation Class property



- Set properties for the PJC

```
SET_CUSTOM_PROPERTY('order_items.quantity',
1,'FILTER_TYPE','NUMERIC');
```

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Using Client-Side Validation (continued)

You implement a PJC to replace an item by setting the item's Implementation Class property to the class of the PJC. You may use the SET\_CUSTOM\_PROPERTY built-in to set properties of the PJC that restrict input or otherwise validate the item. At run time, Forms looks for the Java class contained on the middle tier or in the archive files with the path specified in the Implementation Class for the item. If you open keyfilter.jar in WinZip, you find that the path to KeyFilter.class is oracle\forms\demos.

You deploy the PJC as you would a JavaBean, which was discussed in the lesson titled "Adding Functionality to Items." You can locate the Java class file:

- On the middle-tier server, either in the directory structure referenced by the form applet's CODEBASE parameter or in the server's CLASSPATH. CODEBASE is by default the forms\java subdirectory of ORACLE\_HOME.
- If using JInitiator, in a JAR file in the middle-tier server's CODEBASE directory, and included in the ARCHIVE parameter so that the JAR file is downloaded to and cached on the client. For example:

```
archive_jini=frmall_jinit.jar,keyfilter.jar
```

(The CODEBASE and ARCHIVE parameters are set in the formsweb.cfg file.)

# Tracking Validation Status

- **NEW**
  - When a record is created
  - Also for Copy Value from Item or Initial Value
- **CHANGED**
  - When changed by user or trigger
  - When any item in new record is changed
- **VALID**
  - When validation has been successful
  - After records are fetched from database
  - After a successful post or commit
  - Duplicated record inherits status of source

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Tracking Validation Status

When Forms leaves an object, it usually validates any changes that were made to the contents of the object. To determine whether validation must be performed, Forms tracks the validation status of items and records.

## Tracking Validation Status (continued)

### Item Validation Status

Status	Definition
NEW	When a record is created, Forms marks every item in that record as new. This is true even if the item is populated by the Copy Value from Item or Initial Value item properties, or by the When-Create-Record trigger.
CHANGED	Forms marks an item as changed under the following conditions: <ul style="list-style-type: none"><li>• When the item is changed by the user or a trigger</li><li>• When any item in a new record is changed, all of the items in the record are marked as changed.</li></ul>
VALID	Forms marks an item as valid under the following conditions: <ul style="list-style-type: none"><li>• All items in the record that are fetched from the database are marked as valid.</li><li>• If validation of the item has been successful</li><li>• After successful post or commit</li><li>• Each item in a duplicated record inherits the status of its source.</li></ul>

### Record Validation Status

Status	Definition
NEW	When a record is created, Forms marks that record as new. This is true even if the item is populated by the Copy Value from Item or Initial Value item properties, or by the When-Create-Record trigger.
CHANGED	Whenever an item in a record is marked as changed, Forms marks that record as changed.
VALID	Forms marks a record as valid under the following conditions: <ul style="list-style-type: none"><li>• After all items in the record have been successfully validated</li><li>• All records that are fetched from the database are marked as valid</li><li>• After successful post or commit</li><li>• A duplicate record inherits the status of its source</li></ul>

# Controlling When Validation Occurs with Built-Ins

- **CLEAR\_BLOCK, CLEAR\_FORM, EXIT\_FORM**
- **ENTER**
- **SET\_FORM\_PROPERTY**
  - **(..., VALIDATION)**
  - **(..., VALIDATION\_UNIT)**
- **ITEM\_IS\_VALID item property**
- **VALIDATE (scope)**

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Controlling When Validation Occurs with Built-Ins

You can use the following built-in subprograms in triggers to affect validation.

### **CLEAR\_BLOCK, CLEAR\_FORM, and EXIT\_FORM**

The first parameter to these built-ins, COMMIT\_MODE, controls what will be done with unapplied changes when a block is cleared, the form is cleared, or the form is exited, respectively. When the parameter is set to NO\_VALIDATE, changes are neither validated nor committed (by default, the operator is prompted for the action).

### **ITEM\_IS\_VALID Item Property**

You can use GET\_ITEM\_PROPERTY and SET\_ITEM\_PROPERTY built-ins with the ITEM\_IS\_VALID parameter to get or set the validation status of an item. You cannot directly get and set the validation status of a record. However, you can get or set the validation status of all the items in a record.

### **ENTER**

The ENTER built-in performs the same action as the Enter key. That is, it forces validation of data in the current validation unit.

## Controlling When Validation Occurs with Built-Ins (continued)

### **SET\_FORM\_PROPERTY**

You can use this to disable Forms validation. For example, suppose you are testing a form, and you need to bypass normal validation. Set the Validation property to Property\_False for this purpose:

```
SET_FORM_PROPERTY('form_name', VALIDATION, PROPERTY_FALSE);
```

You can also use this built-in to change the validation unit programmatically:

```
SET_FORM_PROPERTY('form_name', VALIDATION_UNIT, scope);
```

### **VALIDATE**

VALIDATE(scope) forces Forms to immediately execute validation processing for the indicated scope.

**Note:** Scope is one of DEFAULT\_SCOPE, BLOCK\_SCOPE, RECORD\_SCOPE, or ITEM\_SCOPE.

# Summary

In this lesson, you should have learned that:

- The validation unit specifies how much data is entered before validation occurs.
- You can control validation using:
  - Object properties: Validation Unit (form); Validate from List (item)
  - Triggers: When-Validate-Item (item level); When-Validate-Record (block level)
  - Pluggable Java Components for client-side validation

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Summary

In this lesson, you learned to use additional validation features in Forms Builder, and to control when validation occurs.

- Validation occurs at several levels: Item, Record, Block, and Form.
- Validation happens when:
  - The Enter key is pressed or the ENTER built-in procedure is run (to force validation immediately.)
  - Control leaves the validation unit due to navigation or Commit.
- Standard validation occurs before trigger validation.
- The Default validation unit is item level.
- The When-Validate-“*object*” triggers supplement standard validation.
- You can use Pluggable Java Components to perform client-side validation.

## Summary

In this lesson, you should have learned that:

- Forms tracks validation status of items and records, which are either NEW, CHANGED, or VALID.
- You can use built-ins to control when validation occurs:
  - CLEAR\_BLOCK
  - CLEAR\_FORM
  - EXIT\_FORM
  - ENTER
  - ITEM\_IS\_VALID
  - VALIDATE

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Summary (continued)

- Forms tracks validation status internally: NEW, CHANGED, or VALID
- You can use built-ins to control when validation occurs.

## Practice 19: Overview

This practice covers the following topics:

- Validating the Sales Representative item value by using an LOV
- Writing a validation trigger to check that online orders are CREDIT orders
- Populating customer names, sales representative names, and IDs when a customer ID is changed
- Writing a validation trigger to populate the name and the price of the product when the product ID is changed
- Restricting user input to numeric characters using a Pluggable Java Component

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Practice 19: Overview

In this practice, you introduce additional validation to the CUSTGXX and ORDGXX form modules.

- Validating sales representative item value by using an LOV
- Writing a validation trigger to check that all online orders are CREDIT orders
- Populating customer names, sales representative names, and IDs when a customer ID is changed
- Writing a validation trigger to populate the name and the price of the product when the product ID is changed
- Implementing client-side validation on the item quantity using a Pluggable Java Component

**Note:** For solutions to this practice, see Practice 19 in Appendix A, “Practice Solutions.”

## Practice 19

1. In the CUSTGXX form, cause the Account\_Mgr\_Lov to be displayed whenever the user enters an Account\_Mgr\_Id that does not exist in the database.
2. Save and compile the form. Click Run Form to run the form and test the functionality.
3. In the ORDGXX 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.
4. In the ORDGXX form, create triggers to write the correct values to the Customer\_Name whenever validation occurs on Customer\_Id, and to the Sales\_Rep\_Name when validation occurs on Sales\_Rep\_Id. Fail the triggers if the data is not found. You can import text from pr19\_4a.txt and pr19\_4b.txt.
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.
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.
7. Save and compile the form. Click Run Form to run the 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.

Oracle Internal & OAI Use Only



# 20

## Navigation

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

Oracle Internal & OAI Use Only

# Objectives

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

- Distinguish between internal and external navigation
- Control navigation with properties
- Describe and use navigation triggers to control navigation
- Use navigation built-ins in triggers

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Introduction

### Overview

Forms Builder offers a variety of ways to control cursor movement. This lesson looks at the different methods of forcing navigation both visibly and invisibly.

# Navigation: Overview

- **What is the navigational unit?**
  - Outside the form
  - Form
  - Block
  - Record
  - Item
- **Entering and leaving objects**
- **What happens if navigation fails?**

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Navigation: Overview

The following sections introduce a number of navigational concepts to help you to understand the navigation process.

### What Is the Navigational Unit?

The navigational unit is an invisible, internal object that determines the navigational state of a form. Forms uses the navigational unit to keep track of the object that is currently the focus of a navigational process. The navigational unit can be one of the objects in the following hierarchy:

- Outside the form
- Form
- Block
- Record
- Item

When Forms navigates, it changes the navigational unit moving through this object hierarchy until the target item is reached.

## **Navigation: Overview (continued)**

### **Entering and Leaving Objects**

During navigation, Forms leaves and enters objects. Entering an object means changing the navigational unit from the object above in the hierarchy. Leaving an object means changing the navigational unit to the object above.

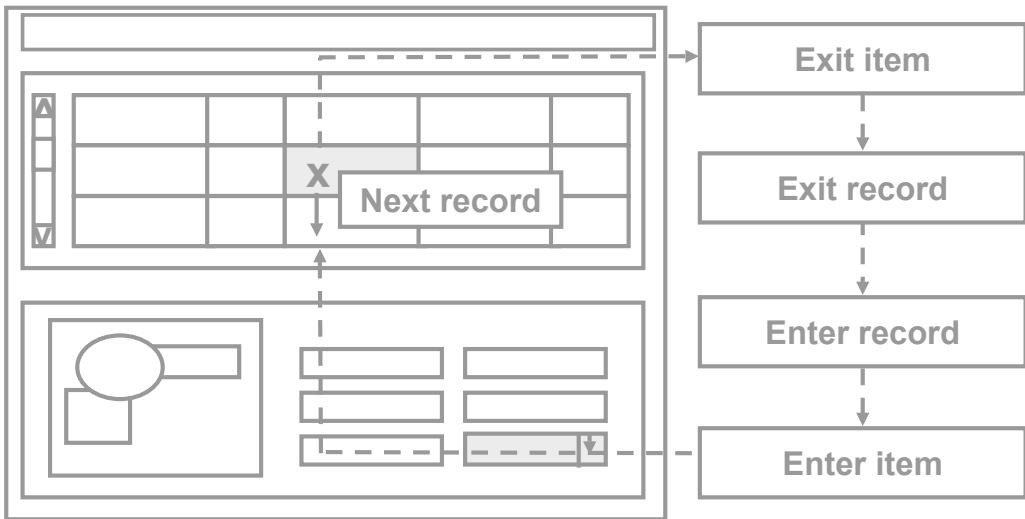
### **The Cursor and How It Relates to the Navigational Unit**

The cursor is a visible, external object that indicates the current input focus. Forms will not move the cursor until the navigational unit has successfully become the target item. In this sense, the navigational unit acts as a probe.

### **What Happens if Navigation Fails?**

If navigation fails, Forms reverses the navigation path and attempts to move the navigational unit back to its initial location. Note that the cursor is still at its initial position. If Forms cannot move the navigational unit back to its initial location, it exits the form.

# Internal Navigation



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Internal Navigation

Navigation occurs when the user or a trigger causes the input focus to move to another object. You have seen that navigation involves changing the location of the input focus on the screen. In addition to the visible navigation that occurs, some logical navigation takes place. This logical navigation is also known as internal navigation.

### Example

When you enter a form module, you see the input focus in the first enterable item of the first navigation block. You do not see the internal navigation events that must occur for the input focus to enter the first item. These internal navigation events are as follows:

- Entry to form
- Entry to block
- Entry to record
- Entry to item

## **Internal Navigation (continued)**

### **Example**

When you commit your inserts, updates, and deletes to the database, you do not see the input focus moving. However, the following navigation events must occur internally before commit processing begins:

- Exit current item
- Exit current record
- Exit current block

### **Performance Note**

Forms uses smart event bundling: All the events that are triggered by navigation between two objects are delivered as one packet to Forms Services on the middle tier for subsequent processing.

Oracle Internal & OAI Use Only

# Using Object Properties to Control Navigation

- **Block:**
  - Navigation Style
  - Previous Navigation Data Block
  - Next Navigation Data Block
- **Item:**
  - Enabled
  - Keyboard Navigable
  - Mouse Navigate
  - Previous Navigation Item
  - Next Navigation Item

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

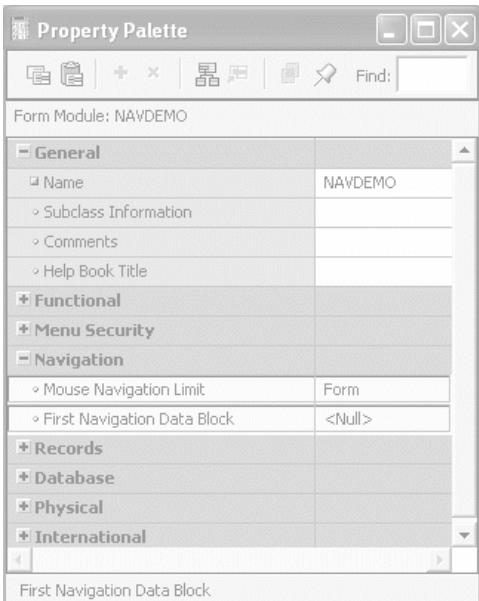
## Controlling Navigation Using Object Properties

You can control the path through an application by controlling the order in which the user navigates to objects. You have seen navigation properties for blocks and items:

Object	Property
Block	Navigation Style Previous Navigation Data Block Next Navigation Data Block
Item	Enabled Keyboard Navigable Mouse Navigate Previous Navigation Item Next Navigation Item

**Note:** You can use the mouse to navigate to any enabled item regardless of its position in the navigational order.

# Using Object Properties to Control Navigation



- **Form module:**
  - Mouse Navigation Limit
  - First Navigation Data Block

ORACLE®

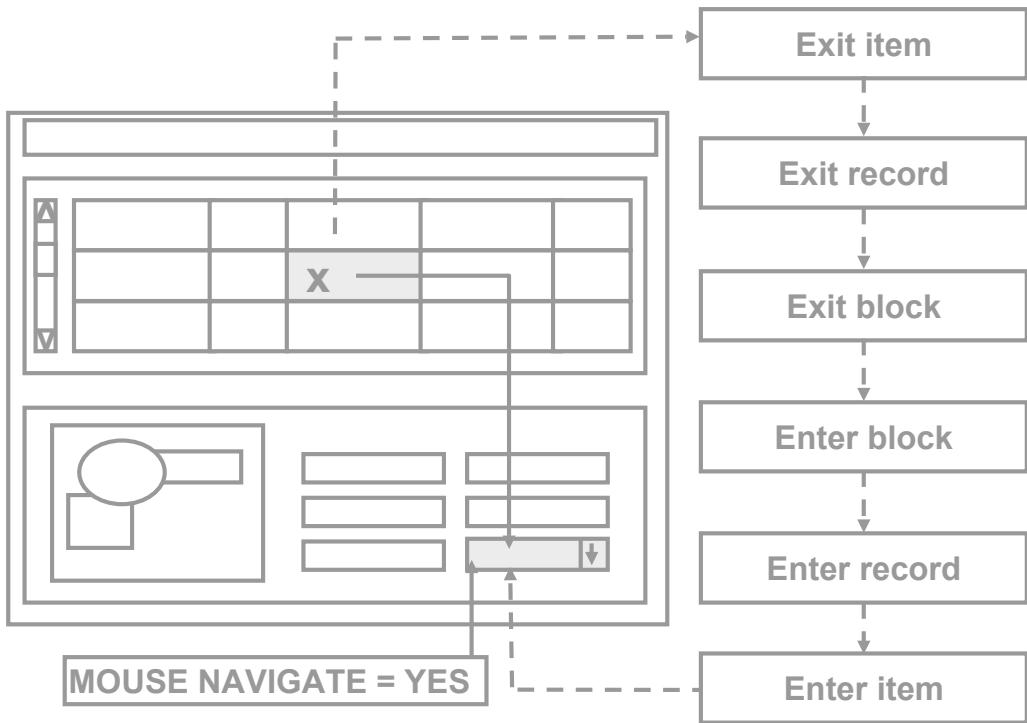
Copyright © 2006, Oracle. All rights reserved.

## Controlling Navigation Using Object Properties (continued)

There are two other navigation properties that you can set for the form module: Mouse Navigation Limit and First Navigation Data Block.

Form Module Property	Function
Mouse Navigation Limit	Determines how far outside the current item the user can navigate with the mouse
First Navigation Block	Specifies the name of the block to which Forms should navigate on form startup (Setting this property does not override the order used for committing.)

## Mouse Navigate Property



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Mouse Navigate Property

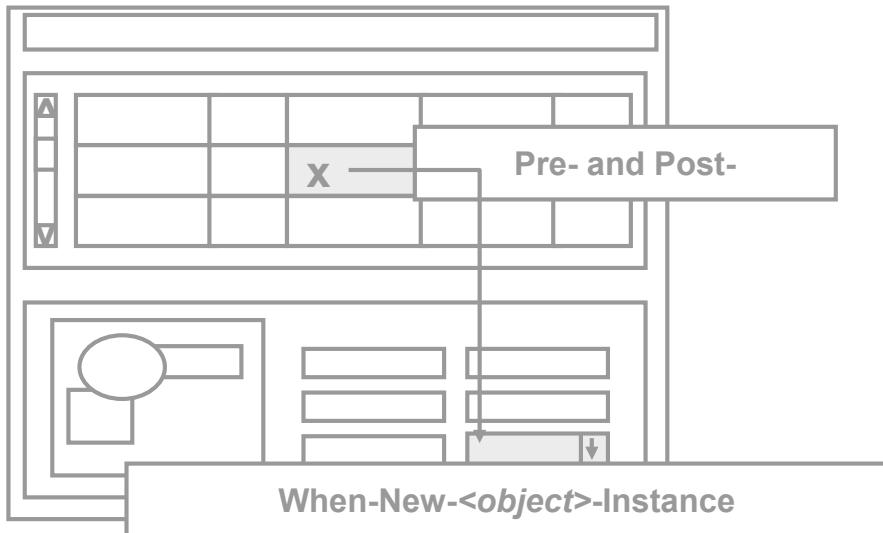
The Mouse Navigate property is valid for the following items:

- Push button
- Check box
- List item
- Radio group
- Hierarchical tree item
- Bean Area Item

**Note:** The default setting for the Mouse Navigate property is Yes.

Setting	Use to Ensure That:
Yes	Forms navigates to the new item. (This causes the relevant navigational and validation triggers to fire.)
No	Forms does not navigate to the new item or validate the current item when the user activates the new item with the mouse.

# Writing Navigation Triggers



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Writing Navigation Triggers

The navigation triggers can be subdivided into two general groups:

- Pre- and Post- navigation triggers
- When-New-<object>-Instance triggers

### When Do Pre- and Post-Navigation Triggers Fire?

The Pre- and Post- navigation triggers fire during navigation, just before entry to or just after exit from the object specified as part of the trigger name.

#### Example

The Pre-Text-Item trigger fires just before entering a text item.

### When Do Navigation Triggers Not Fire?

The Pre- and Post-navigation triggers do not fire if they belong to a unit that is lower in the hierarchy than the current validation unit. For instance, if the validation unit is Record, Pre- and Post-Text-Item triggers do not fire.

# Navigation Triggers

Pre- and Post-	When-New-<object>-Instance
Fire during navigation	Fire after navigation
Do not fire if validation unit is higher than trigger object	Fire even when validation unit is higher than trigger object
Allow unrestricted built-ins	Allow restricted and unrestricted built-ins
Handle failure by returning to initial object	Are not affected by failure

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Navigation Triggers

### When Do When-New-<object>-Instance Triggers Fire?

The When-New-<object>-Instance triggers fire immediately after navigation to the object specified as part of the trigger name.

#### Example

The When-New-Item-Instance trigger fires immediately after navigation to a new instance of an item.

#### What Happens when a Navigation Trigger Fails?

If a Pre- or Post-navigation trigger fails, the input focus returns to its initial location (where it was prior to the trigger firing). To the user, it appears that the input focus has not moved at all.

**Note:** Be sure that Pre- and Post-navigation triggers display a message on failure. Failure of a navigation trigger can cause a fatal error to your form. For example, failure of Pre-Form, Pre-Block, Pre-Record, or Pre-Text-Item on entry to the form will cancel execution of the form.

## When-New-<object>-Instance Triggers

- **When-New-Form-Instance**
- **When-New-Block-Instance**
- **When-New-Record-Instance**
- **When-New-Item-Instance**

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Using the When-New-<object>-Instance Triggers

If you include complex navigation paths through your application, you may want to check or set initial conditions when the input focus arrives in a particular block, record, or item. Use the following triggers to do this:

Trigger	Fires
When-New-Form-Instance	Whenever a form is run, after successful navigation into the form
When-New-Block-Instance	After successful navigation into a block
When-New-Record-Instance	After successful navigation into the record
When-New-Item-Instance	After successful navigation to a new instance of the item

## SET\_<object>\_PROPERTY: Examples

```
SET [FORM] PROPERTY(FIRST_NAVIGATION_BLOCK,  
'ORDER_ITEMS');
```

```
SET [BLOCK] PROPERTY('ORDERS', ORDER_BY,  
'CUSTOMER_ID');
```

```
SET [RECORD] PROPERTY(3, 'ORDER_ITEMS', STATUS,  
QUERY_STATUS);
```

```
SET [ITEM] PROPERTY('CONTROL.stock_button',  
ICON_NAME, 'stock');
```

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Initializing Forms Builder Objects

Use the When-New-<object>-Instance triggers, along with the SET\_<object>\_PROPERTY built-in subprograms to initialize Forms Builder objects. These triggers are particularly useful if you conditionally require a default setting.

#### Example

The following example of a When-New-Block-Instance trigger conditionally sets the DELETE ALLOWED property to FALSE.

```
IF GET_APPLICATION_PROPERTY(username) = 'SCOTT' THEN  
SET_BLOCK_PROPERTY('ORDER_ITEMS', DELETE_ALLOWED, PROPERTY_FALSE);  
END IF;
```

#### Example

Perform a query of all orders, when the ORDERS form is run, by including the following code in your When-New-Form-Instance trigger:

```
EXECUTE_QUERY;
```

## Initializing Forms Builder Objects (continued)

### Example

Register the Color Picker JavaBean into the Control.Colorpicker bean area item when the CUSTOMERS form is run by including the following code in your When-New-Form-Instance trigger:

```
FBean.Register_Bean('control.colorpicker',1,  
'oracle.forms.demos.beans.ColorPicker');
```

At run time, Forms looks for the Java class contained on the middle tier or in the archive files with the path specified in the code. If you open colorpicker.jar in WinZip, you find that the path to ColorPicker.class is oracle\forms\demos\beans.

Oracle Internal & OAI Use Only

## Pre- and Post-Triggers

- Pre/Post-Form
- Pre/Post-Block
- Pre/Post-Record
- Pre/Post-Text-Item



Copyright © 2006, Oracle. All rights reserved.

### Using the Pre- and Post-Triggers

Define Pre- and Post-Text-Item triggers at item level, Pre- and Post-Block at block level, and Pre- and Post-Form at form level. Pre- and Post-Text-Item triggers fire only for text items.

## Using the Pre- and Post-Triggers (continued)

Trigger Type	Use to
Pre-Form	<ul style="list-style-type: none"><li>Validate<ul style="list-style-type: none"><li>User</li><li>Time of day</li></ul></li><li>Initialize control blocks</li><li>Call another form to display messages</li></ul>
Post-Form	<ul style="list-style-type: none"><li>Perform housekeeping, such as erasing global variables</li><li>Display messages to user before exit</li></ul>
Pre-Block	<ul style="list-style-type: none"><li>Authorize access to the block</li><li>Set global variables</li></ul>
Post-Block	<ul style="list-style-type: none"><li>Validate the last record that had input focus</li><li>Test a condition and prevent the user from leaving the block</li></ul>
Pre-Record	<ul style="list-style-type: none"><li>Set global variables</li></ul>
Post-Record	<ul style="list-style-type: none"><li>Clear global variables</li><li>Set a visual attribute for an item as the user scrolls through a set of records</li><li>Perform cross-field validation</li></ul>
Pre-Text-Item	<ul style="list-style-type: none"><li>Derive a complex default value</li><li>Record the previous value of a text item</li></ul>
Post-Text-Item	<ul style="list-style-type: none"><li>Calculate or change item values</li></ul>

Oracle Internal & OAI Use Only

## Post-Block Trigger: Example

**Disabling the Stock button when leaving the ORDER\_ITEMS block:**

```
SET_ITEM_PROPERTY('CONTROL.stock_button',  
enabled, property_false);
```

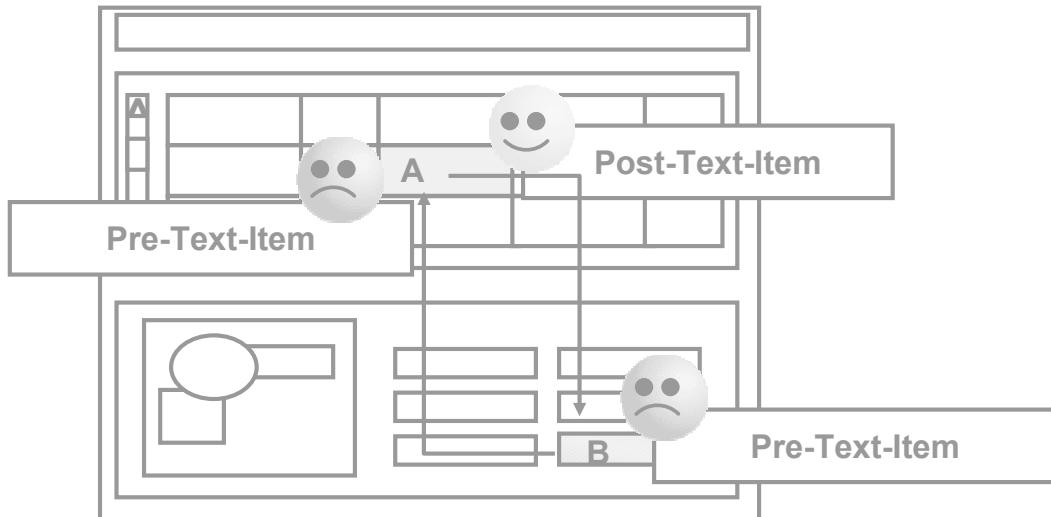


Copyright © 2006, Oracle. All rights reserved.

### Post-Block Trigger: Example

The example on the slide shows the use of a Post-Block trigger to disable a toolbar button that is only valid for the current block.

# Navigation Trap



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Navigation Trap

You have seen that the Pre- and Post- navigation triggers fire during navigation, and when they fail, the internal cursor attempts to return to the current item (`SYSTEM.CURSOR_ITEM`).

The diagram in the slide illustrates the navigation trap. This can occur when a Pre- navigation trigger (on the item to which the user is navigating) fails and attempts to return the logical cursor to its initial item. However, if the initial item has a Pre-Text-Item trigger that also fails, then the cursor has nowhere to go, and a fatal error occurs.

**Note:** Be sure to code against navigation trigger failure.

# Using Navigation Built-ins in Triggers

GO\_FORM  
GO\_BLOCK  
GO\_ITEM  
GO\_RECORD  
NEXT\_BLOCK  
NEXT\_ITEM  
NEXT\_KEY  
NEXT\_RECORD

NEXT\_SET  
UP  
DOWN  
PREVIOUS\_BLOCK  
PREVIOUS\_ITEM  
PREVIOUS\_RECORD  
SCROLL\_UP  
SCROLL\_DOWN

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Using Navigation Built-ins in Triggers

You can initiate navigation programmatically by calling the built-in subprograms, such as GO\_ITEM and PREVIOUS\_BLOCK from triggers.

Built-ins for Navigation	Function
GO_FORM	Navigates to an open form in a multiple form application
GO_BLOCK/ITEM/RECORD	Navigates to the indicated block, item, or record
NEXT_BLOCK/ITEM/KEY	Navigates to the next enterable block, item, or primary key item
NEXT/PREVIOUS_RECORD	Navigates to the first enterable item in the next or previous record
NEXT_SET	Fetches another set of records from the database and navigates to the first record that the fetch retrieves
UP, DOWN	Navigates to the instance of the current item in the previous/next record
PREVIOUS_BLOCK/ITEM	Navigates to the previous enterable block or item
SCROLL_UP/DOWN	Scrolls the block so that the records above the top visible one or below the bottom visible one are displayed

# Using Navigation Built-ins in Triggers

- When-New-Item-Instance

```
IF CHECKBOX_CHECKED('ORDERS.order_mode') --Online  
THEN  
    ORDERS.order_status := 4; --Credit order  
    GO_ITEM('ORDERS.order_status');   
END IF;
```

- Pre-Text-Item

```
IF CHECKBOX_CHECKED('ORDERS.order_mode') --Online  
THEN  
    ORDERS.order_status := 4; --Credit order  
    GO_ITEM('ORDERS.order_status');   
END IF;
```

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Using Navigation Built-ins in Triggers (continued)

### Calling Built-ins from Navigational Triggers

You are not allowed to use a restricted built-in from within a trigger that fires during the navigation process (the Pre- and Post- triggers). This is because restricted built-ins perform some sort of navigation, and therefore, cannot be called until Forms navigation is complete.

You can call restricted built-ins from triggers such as When-New-Item-Instance because that trigger fires after Forms has moved input focus to the new item.

## Summary

In this lesson, you should have learned that:

- External navigation is visible to the user, whereas internal navigation occurs behind the scenes
- You can control navigation with properties of the form, block, or item in one of the following ways:
  - Setting in Navigation category of the Property Palette
  - Using `SET_[FORM | BLOCK | ITEM]_PROPERTY`

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Summary

In this lesson, you learned the different methods of forcing visible navigation and also the invisible events.

- You can control navigation through the following properties:
  - Form module properties
  - Data block properties
  - Item properties
- Internal navigation events also occur.

## Summary

In this lesson, you should have learned that:

- Navigation triggers are:
  - Those that fire during navigation (watch out for the navigation trap):  
[Pre | Post] - [Form | Block | Item]
  - Those that fire after navigation:  
When-New- [Form | Block | Record | Item] -Instance
- You can use navigation built-ins in triggers  
(except for triggers that fire during navigation):
  - GO\_[FORM | BLOCK | RECORD | ITEM]
  - NEXT\_[BLOCK | RECORD | ITEM | KEY | SET]
  - UP
  - DOWN
  - PREVIOUS\_[BLOCK | RECORD | ITEM]
  - SCROLL\_[UP | DOWN]

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Summary (continued)

- Navigation triggers:
  - When-New-<object>-Instance
  - Pre- and Post-
- Avoid the navigation trap.
- Navigation built-ins are available.

## Practice 20: Overview

This practice covers the following topics:

- Registering the bean area's JavaBean at form startup
- Setting properties on a Pluggable Java Component at form startup
- Executing a query at form startup
- Populating product images when cursor arrives on each record of the ORDER\_ITEMS block

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Practice 20: Overview

In this practice, you provide a trigger to automatically perform a query, register a JavaBean, and set properties on a PJC at form startup. Also, you use When-New-<object>-Instance triggers to populate the Product\_Image item as the operator navigates between records in the ORDGXX form.

- Executing a query at form startup
- Populating product images when the cursor arrives on each record of the ORDER\_ITEMS block

**Note:** For solutions to this practice, see Practice 20 in Appendix A, “Practice Solutions.”

## Practice 20

1. When the ORDGXX 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.
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.
3. Define the same trigger type and code on the ORDERS block.
4. Is there another trigger where you might also want to place this code?
5. Save and compile the form. Click Run Form to run the form and test the changes.
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.
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.
8. Click Run Form to run the form again and test the changes.
9. In the CUSTGXX 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.
10. Save, compile, and click Run Form to run the 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.

Oracle Internal & OAI Use Only

# 21

## Transaction Processing

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

Oracle Internal & OAI Use Only

# Objectives

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

- Explain the process used by Forms to apply changes to the database
- Describe the commit sequence of events
- Supplement transaction processing
- Allocate sequence numbers to records as they are applied to tables
- Implement array DML



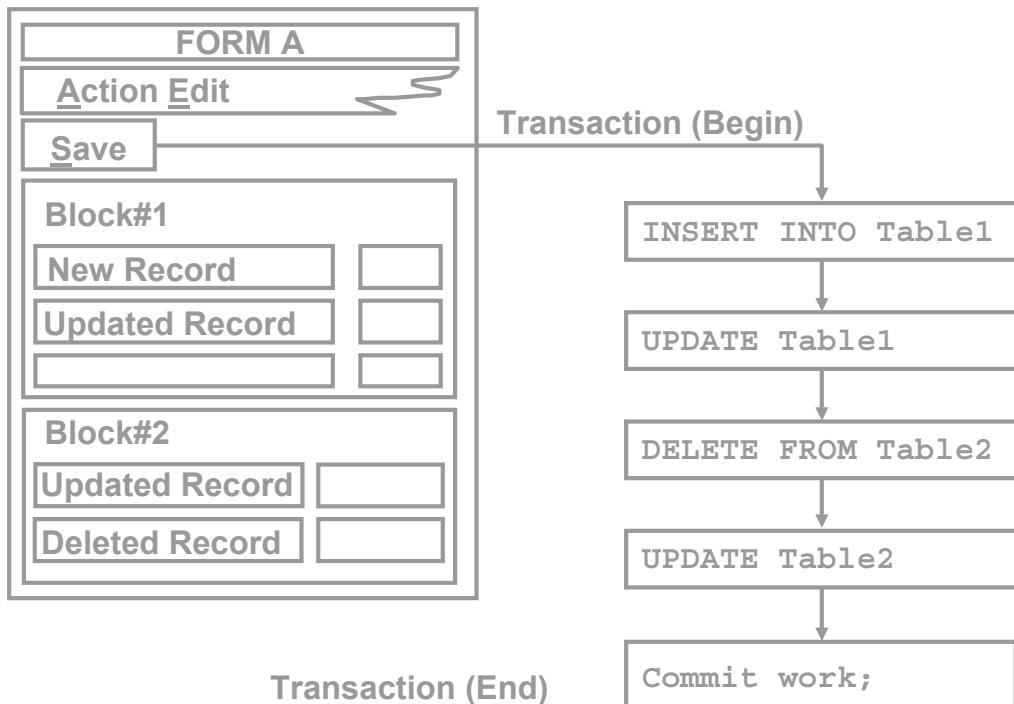
Copyright © 2006, Oracle. All rights reserved.

## Introduction

### Overview

While applying a user's changes to the database, Forms Builder enables you to make triggers fire in order to alter or add to the default behavior. This lesson shows you how to build triggers that can perform additional tasks during this stage of a transaction.

# Transaction Processing: Overview



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Transaction Processing: Overview

When Forms is asked to save the changes made in a form by the user, a process takes place involving events in the current database transaction. This process includes:

- **Default Forms transaction processing:** Applies the user's changes to the base tables
- **Firing transactional triggers:** Are needed to perform additional or modified actions in the saving process defined by the designer

When all of these actions are successfully completed, Forms commits the transaction, making the changes permanent.

# Transaction Processing: Overview

**Transaction processing includes two phases:**

- **Post:**
  - Writes record changes to base tables
  - Fires transactional triggers
- **Commit: Performs database commit**

**Errors result in:**

- **Rollback of the database changes**
- **Error message**

**ORACLE®**

Copyright © 2006, Oracle. All rights reserved.

## Transaction Processing: Overview (continued)

The transaction process occurs as a result of either of the following actions:

- The user clicks Save or selects Action > Save from the menu, or clicks Save on the default Form toolbar.
- The COMMIT\_FORM built-in procedure is called from a trigger.

In either case, the process involves two phases, posting and committing:

**Post:** Posting writes the user's changes to the base tables, using implicit INSERT, UPDATE, and DELETE statements generated by Forms. The changes are applied in block sequence order as they appear in the Object Navigator at design time. For each block, deletes are performed first, followed by inserts and updates. Transactional triggers fire during this cycle if defined by the designer.

The built-in procedure POST alone can invoke this posting process.

**Commit:** This performs the database commit, making the applied changes permanent and releasing locks.

## **Transaction Processing: Overview (continued)**

Other events related to transactions include rollbacks, savepoints, and locking.

### **Rollbacks**

Forms will roll back applied changes to a savepoint if an error occurs in its default processing, or when a transactional trigger fails.

By default, the user is informed of the error through a message, and a failing insert or update results in the record being redisplayed. The user can then attempt to correct the error before trying to save again.

### **Savepoints**

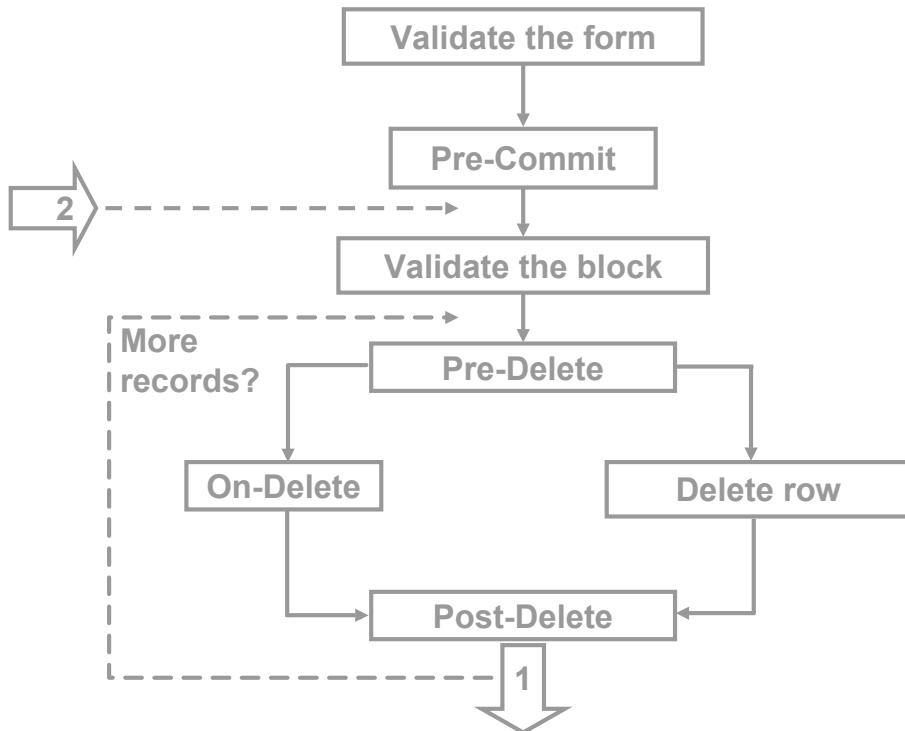
Forms issues savepoints in a transaction automatically, and will roll back to the latest savepoint if certain events occur. Generally, these savepoints are for Forms internal use, but certain built-ins, such as the `EXIT_FORM` built-in procedure, can request a rollback to the latest savepoint by using the `TO_SAVEPOINT` option.

### **Locking**

When you update or delete base table records in a form application, database locks are automatically applied. Locks also apply during the posting phase of a transaction, and for DML statements that you explicitly use in your code.

**Note:** The SQL statements `COMMIT`, `ROLLBACK`, and `SAVEPOINT` cannot be called from a trigger directly. If encountered in a Forms program unit, Forms treats `COMMIT` as the `COMMIT_FORM` built-in, and `ROLLBACK` as the `CLEAR_FORM` built-in.

## Commit Sequence of Events



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Commit Sequence of Events

The commit sequence of events (when the Array DML size is 1) is as follows:

1. Validate the form.
2. Process savepoint.
3. Fire the Pre-Commit trigger.
4. Validate the block (for all blocks in sequential order).
5. Perform the DML:

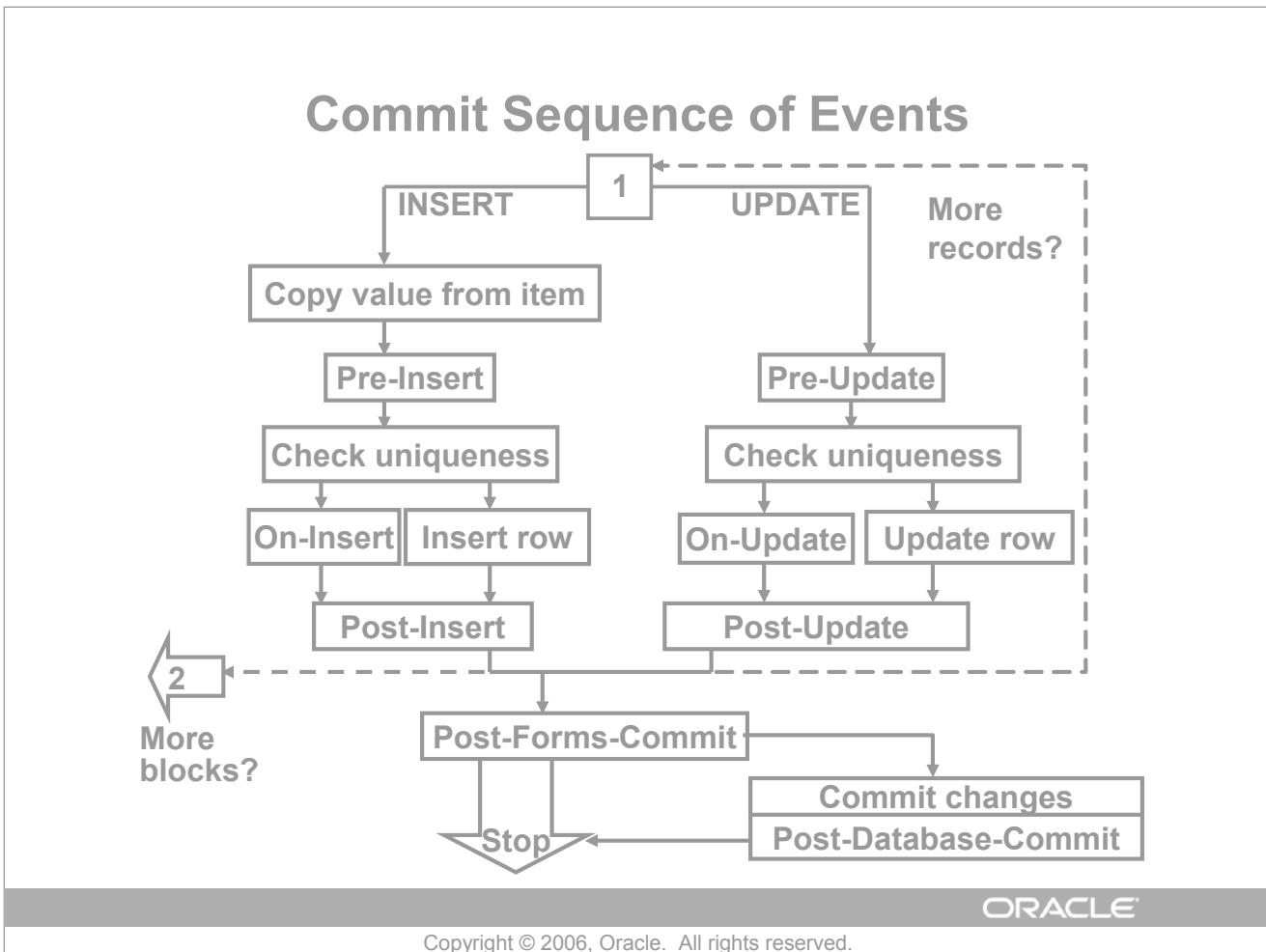
For all deleted records of the block (in reverse order of deletion):

- Fire the Pre-Delete trigger.
- Delete the row from the base table or fire the On-Delete trigger.
- Fire the Post-Delete trigger.

For all inserted or updated records of the block in sequential order:

If it is an inserted record:

- Copy Value From Item
- Fire the Pre-Insert trigger
- Check the record uniqueness
- Insert the row into the base table or fire the On-Insert trigger
- Fire the Post-Insert trigger



### Commit Sequence of Events (continued)

If it is an updated record:

- Fire the Pre-Update trigger
- Check the record uniqueness
- Update the row in the base table or fire the On-Update trigger
- Fire the Post-Update trigger

6. Fire the Post-Forms-Commit trigger.

If the current operation is COMMIT, then:

7. Issue a SQL-COMMIT statement
8. Fire the Post-Database-Commit trigger

Oracle Internal & OA Use Only

# Characteristics of Commit Triggers

- **Pre-Commit:** Fires once if form changes are made or uncommitted changes are posted
- **Pre- and Post-DML**
- **On-DML:** Fires per record, replacing default DML on row  
**Use `DELETE_RECORD`, `INSERT_RECORD`, `UPDATE_RECORD` built-ins**

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Characteristics of Commit Triggers

You have already seen when commit triggers fire during the normal flow of commit processing. The table on the next page gives more detailed information regarding the conditions under which these triggers fire.

It is usually unnecessary to code commit triggers, and the potential for coding errors is high. Because of this, use commit triggers only if your application requires special processing at commit time.

One valid use of commit triggers is to distribute DML statements to underlying tables when you are performing DML on a block based on a join view. However, using a database instead-of trigger may eliminate the need to define specialized Forms commit triggers for this purpose.

**Note:** If a commit trigger—except for the Post-Database-Commit trigger—fails, the transaction is rolled back to the savepoint that was set at the beginning of the current commit processing. This also means that uncommitted posts issued before the savepoint are not rolled back.

# Characteristics of Commit Triggers

- **Post-Forms-Commit:** Fires once even if no changes are made
- **Post-Database-Commit:** Fires once even if no changes are made

**Note:** A commit-trigger failure causes a rollback to the savepoint.



Copyright © 2006, Oracle. All rights reserved.

## Characteristics of Commit Triggers (continued)

Trigger	Characteristic
Pre-Commit	Fires once during commit processing, before base table blocks are processed; fires if there are changes to base table items in the form or if changes have been posted but not yet committed (always fires in case of uncommitted posts, even if there are no changes to post)
Pre- and Post-DML	Fire for each record that is marked for insert, update, or delete, just before or after the row is inserted, updated, or deleted in the database
On-DML	Fires for each record marked for insert, update, or delete when Forms would typically issue its INSERT, UPDATE, or DELETE statement, replacing the default DML statements (Include a call to INSERT_RECORD, UPDATE_RECORD, or DELETE_RECORD built-in to perform default processing for these triggers.)
Post-Forms-Commit	Fires once during commit processing, after base table blocks are processed but before the SQL-COMMIT statement is issued; even fires if there are no changes to post or commit
Post-Database-Commit	Fires once during commit processing, after the SQL-COMMIT statement is issued; even fires if there are no changes to post or commit (This is also true for the SQL-COMMIT statement.)

# Common Uses for Commit Triggers

Pre-Commit	Check user authorization; set up special locking
Pre-Delete	Journaling; implement foreign-key delete rule
Pre-Insert	Generate sequence numbers; journaling; automatically generated columns; check constraints
Pre-Update	Journaling; implement foreign-key update rule; auto-generated columns; check constraints

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Common Uses for Commit Triggers

After you know when a commit trigger fires, you should be able to choose the right commit trigger for the functionality that you want. To help you with this, the most common uses for commit triggers are mentioned in the table on the next page.

Where possible, implement functionality such as writing to a journal table, automatically supplying column values, and checking constraints in the server.

**Note:** Locking is also needed for transaction processing. You can use the On-Lock trigger if you want to amend the default locking of Forms.

Use DML statements in commit triggers only; otherwise, the DML statements are not included in the administration kept by Forms concerning commit processing. This may lead to unexpected and unwanted results.

## Common Uses for Commit Triggers

On-Insert/Update/Delete	Replace default block DML statements
Post-Forms-Commit	Check complex multirow constraints
Post-Database-Commit	Test commit success; test uncommitted posts

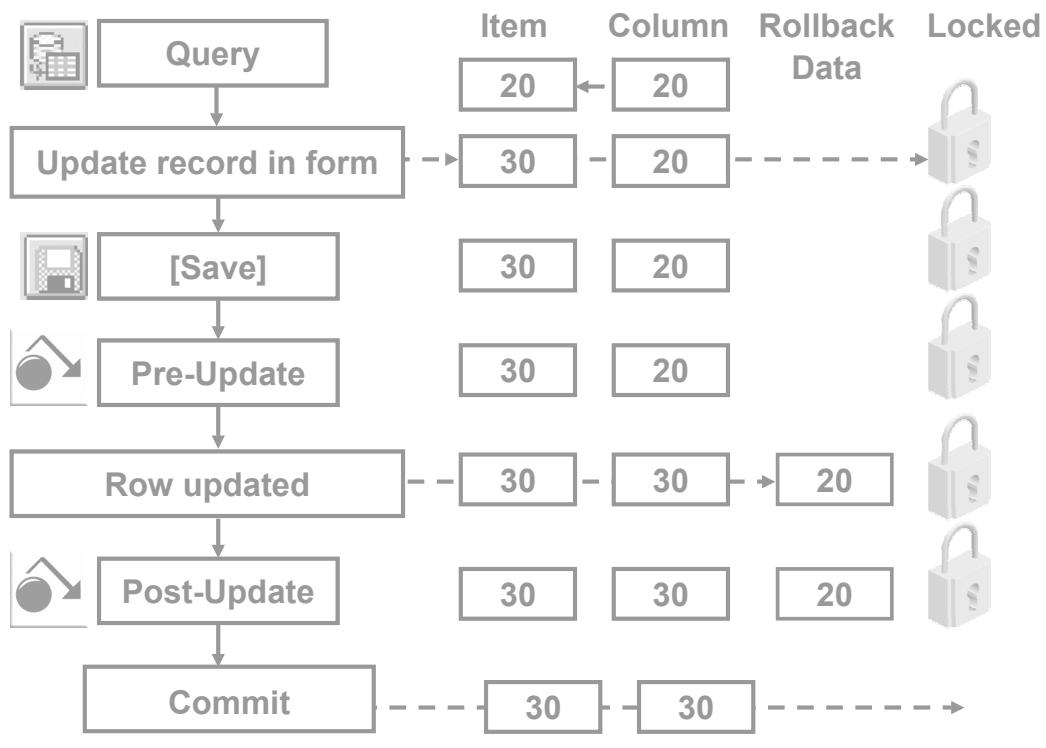
ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Common Uses for Commit Triggers (continued)

Trigger	Common Use
Pre-Commit	Checks user authorization; sets up special locking requirements
Pre-Delete	Writes to journal table; implements restricted or cascade delete
Pre-Insert	Writes to journal table; fills automatically generated columns; generates sequence numbers; checks constraints
Pre-Update	Writes to journal table; fills automatically generated columns; checks constraints; implements restricted or cascade update
Post-Delete, Post-Insert, Post-Update	Seldom used
On-Delete, On-Insert, On-Update	Replaces default block DML statements; for example, to implement a pseudo delete or to update a join view
Post-Forms-Commit	Checks complex multi-row constraints
Post-Database-Commit	Determines if commit was successful; determines if there are posted uncommitted changes

## Life of an Update



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Life of an Update

To help you decide where certain trigger actions can be performed, consider an update operation as an example.

#### Example

The price of a product is being updated in a form. After the user queries the record, the following events occur:

1. The user updates the Price item. This is now different from the corresponding database column. By default, the row is locked on the base table.
2. The user saves the change, initiating the transaction process.
3. The Pre-Update trigger fires (if present). At this stage, the item and column are still different, because the update has not been applied to the base table. The trigger could compare the two values (for example, to make sure the new price is not lower than the existing one).
4. Forms applies the user's change to the database row. The item and column are now the same.

## **Life of an Update (continued)**

### **Example (continued)**

5. The Post-Update trigger fires (if present). It is too late to compare the item against the column, because the update has already been applied. However, the Oracle database retains the old column value as rollback data, so that a failure of this trigger reinstates the original value.
6. Forms issues the database commit, thus discarding the rollback data, releasing the lock, and making the changes permanent. The user receives the message Transaction Completed....

Oracle Internal & OAI Use Only

# Delete Validation

- Pre-Delete trigger
- Final checks before row deletion

```
DECLARE
    CURSOR C1 IS
        SELECT 'anything' FROM ORDERS
        WHERE customer_id = :CUSTOMERS.customer_id;
BEGIN
    OPEN C1;
    FETCH C1 INTO :GLOBAL.dummy;
    IF C1%FOUND THEN
        CLOSE C1;
        MESSAGE('There are orders for this
customer!');
        RAISE form_trigger_failure;
    ELSE
        CLOSE C1;
    END IF;
END;
```

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Delete Validation

Master-detail blocks that are linked by a relation with the nonisolated deletion rule automatically prevent master records from being deleted in the form if matching detail rows exist.

You may, however, want to implement a similar check, as follows, when a deletion is applied to the database:

- A final check to ensure that no dependent detail rows have been inserted by another user since the master record was marked for deletion in the form (In an Oracle database, this is usually performed by a constraint or a database trigger.)
- A final check against form data, or checks that involve actions within the application

**Note:** If you select the “Enforce data integrity” check box in the Data Block Wizard, then Forms Builder automatically creates the related triggers to implement constraints.

## Delete Validation (continued)

### Example

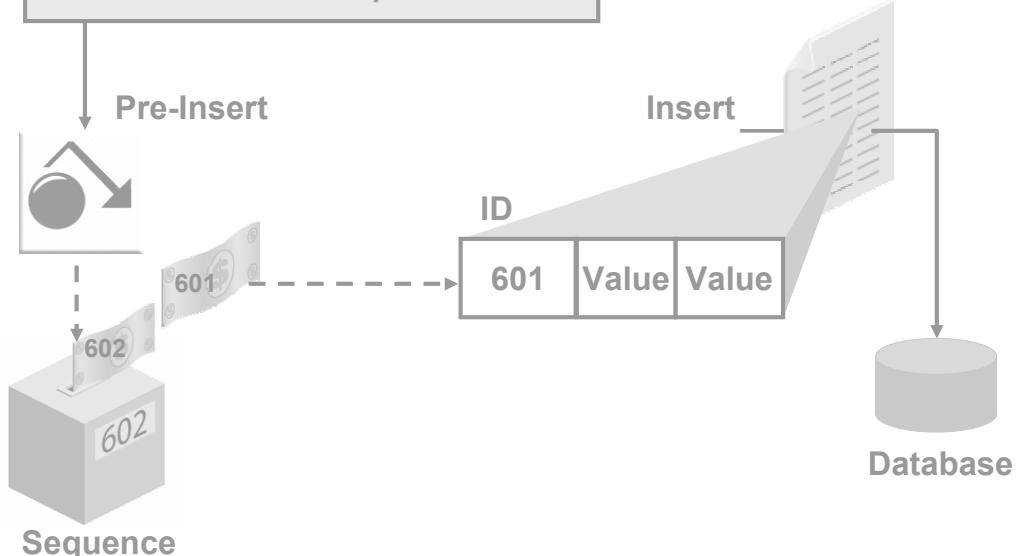
This Pre-Delete trigger on the CUSTOMER block of the CUSTOMERS form prevents deletion of rows if there are existing orders for the customer.

```
DECLARE
    CURSOR C1 IS
        SELECT 'anything' FROM ORDERS
        WHERE customer_id = :CUSTOMERS.customer_id;
BEGIN
    OPEN C1;
    FETCH C1 INTO :GLOBAL.dummy;
    IF C1%FOUND THEN
        CLOSE C1;
        MESSAGE('There are orders for this customer!');
        RAISE form_trigger_failure;
    ELSE
        CLOSE C1;
    END IF;
END;
```

Oracle Internal & OAI Use Only

# Assigning Sequence Numbers

```
SELECT    ORDERS_SEQ.nextval
INTO      :ORDERS.order_id
FROM      SYS.dual;
```



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Assigning Sequence Numbers to Records

You will recall that you can assign default values for items from an Oracle sequence, to automatically provide unique keys for records on their creation. However, if the user does not complete a record, the assigned sequence number is “wasted.”

An alternative method is to assign unique keys to records from a Pre-Insert trigger, just before their insertion in the base table, by which time the user has completed the record and issued the Save.

Assigning unique keys in the posting phase can:

- Reduce gaps in the assigned numbers
- Reduce data traffic on record creation, especially if records are discarded before saving

## Assigning Sequence Numbers to Records (continued)

### Example

This Pre-Insert trigger on the ORDERS block assigns an Order ID from the ORDERS\_SEQ sequence, which will be written to the ORDER\_ID column when the row is subsequently inserted.

```
SELECT ORDERS_SEQ.nextval  
  INTO :ORDERS.order_id  
  FROM SYS.dual;
```

**Note:** The Insert Allowed and Keyboard Navigable properties on :ORDERS.order\_id should be No, so that the user does not enter an ID manually.

You can also assign sequence numbers from a table. If you use this method, then two transactional triggers are usually involved:

- Use Pre-Insert to select the next available number from the sequence table (locking the row to prevent other users from selecting the same value) and increment the value by the required amount.
- Use Post-Insert to update the sequence table, recording the new upper value for the sequence.

Oracle Internal & OAI Use Only

# Keeping an Audit Trail

- Write changes to nonbase tables.
- Gather statistics on applied changes.

Post-Insert example:

```
:GLOBAL.insert_tot :=  
    TO_CHAR(TO_NUMBER(:GLOBAL.insert_tot)+1);
```

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Keeping an Audit Trail

You may want to use the Post-<event> transactional triggers to record audit information about the changes applied to base tables. In some cases, this may involve duplicating inserts or updates in backup history tables, or recording statistics each time a DML operation occurs.

If the base table changes are committed at the end of the transaction, the audit information will also be committed.

### Example

This Post-Update trigger writes the current record ID to the UPDATE\_AUDIT table, along with a time stamp and the user who performed the update.

```
INSERT INTO update_audit (id, timestamp, who_did_it)  
VALUES (:ORDERS.order_id, SYSDATE, USER);
```

### Example

This Post-Insert trigger adds to a running total of Inserts for the transaction, which is recorded in the INSERT\_TOT global variable.

```
:GLOBAL.insert_tot :=  
    TO_CHAR(TO_NUMBER(:GLOBAL.insert_tot)+1);
```

# Testing the Results of Trigger DML

- **SQL%FOUND**
- **SQL%NOTFOUND**
- **SQL%ROWCOUNT**

```
UPDATE ORDERS
      SET order_date = SYSDATE
        WHERE order_id = :ORDERS.order_id;
      IF SQL%NOTFOUND THEN
        MESSAGE('Record not found in database');
        RAISE form_trigger_failure;
      END IF;
```



Copyright © 2006, Oracle. All rights reserved.

## Testing the Results of Trigger DML

When you perform DML in transactional triggers, you may need to test the results.

Unlike SELECT statements, DML statements do not raise exceptions when zero or multiple rows are processed. PL/SQL provides some useful attributes for obtaining results from the implicit cursor used to process the latest SQL statement (in this case, DML).

### Obtaining Cursor Information in PL/SQL

PL/SQL Cursor Attribute	Values
SQL%FOUND	TRUE: Indicates > 0 rows processed FALSE: Indicates 0 rows processed
SQL%NOTFOUND	TRUE: Indicates 0 rows processed FALSE: Indicates > 0 rows processed
SQL%ROWCOUNT	Integer indicating the number of rows processed

## Testing the Results of Trigger DML (continued)

### Obtaining Cursor Information in PL/SQL (continued)

#### Example

This When-Button-Pressed trigger records the date of posting as the date ordered for the current Order record. If a row is not found by the UPDATE statement, an error is reported.

```
UPDATE ORDERS
    SET order_date = SYSDATE
    WHERE order_id = :ORDERS.order_id;
IF SQL%NOTFOUND THEN
    MESSAGE('Record not found in database');
    RAISE form_trigger_failure;
END IF;
```

**Note:** Triggers containing base table DML can adversely affect the usual behavior of your form, because DML statements can cause some of the rows in the database to lock.

Oracle Internal & OAI Use Only

# DML Statements Issued During Commit Processing

```
INSERT INTO base_table  (base_column, base_column, ...)  
VALUES                  (:base_item, :base_item, ...)
```

```
UPDATE    base_table  
SET       base_column = :base_item, base_column =  
          :base_item, ...  
WHERE     ROWID = :ROWID
```

```
DELETE    FROM base_table  
WHERE     ROWID = :ROWID
```



Copyright © 2006, Oracle. All rights reserved.

## DML Statements Issued During Commit Processing

If you have not altered default commit processing, Forms issues DML statements at commit time for each database record that is inserted, updated, or deleted.

```
INSERT INTO base_table  (base_column, base_column, ...)  
VALUES (:base_item, :base_item, ...)  
  
UPDATE base_table  
SET   base_column = :base_item, base_column = :base_item, ...  
      WHERE ROWID = :ROWID  
  
DELETE    FROM base_table  
      WHERE ROWID = :ROWID
```

# DML Statements Issued During Commit Processing

## Rules

- DML statements may fire database triggers.
- Forms uses and retrieves ROWID.
- The Update Changed Columns Only and Enforce Column Security properties affect UPDATE statements.
- Locking statements are not issued.

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## DML Statements Issued During Commit Processing (continued)

### Rules

- These DML statements may fire associated database triggers.
- Forms uses the ROWID construct only when the Key mode block property is set to Unique (or Automatic, the default). Otherwise, the primary key is used to construct the WHERE clause.
- If Forms successfully inserts a row in the database, it also retrieves the ROWID for that row.
- If the Update Changed Columns Only block property is set to Yes, only base columns with changed values are included in the UPDATE statement.
- If the Enforce Column Security block property is set to Yes, all base columns for which the current user has no update privileges are excluded from the UPDATE statement.

Locking statements are not issued by Forms during default commit processing; they are issued as soon as a user updates or deletes a record in the form. If you set the Locking mode block property to delayed, Forms waits to lock the corresponding row until commit time.

# Overriding Default Transaction Processing

Additional transactional triggers:

Trigger	Do-the-Right-Thing Built-in
On-Check-Unique	CHECK_RECORD_UNIQUENESS
On-Column-Security	ENFORCE_COLUMN_SECURITY
On-Commit	COMMIT_FORM
On-Rollback	ISSUE_ROLLBACK
On-Savepoint	ISSUE_SAVEPOINT
On-Sequence-Number	GENERATE_SEQUENCE_NUMBER

Note: These triggers are meant to be used when connecting to data sources other than Oracle.

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Overriding Default Transaction Processing

You have already seen that some commit triggers can be used to replace the default DML statements that Forms issues during commit processing. You can use several other triggers to override the default transaction processing of Forms.

### Transactional Triggers

All triggers that are related to accessing a data source are called *transactional triggers*. Commit triggers form a subset of these triggers. Other examples include triggers that fire during logon and logout or during queries performed on the data source.

# Overriding Default Transaction Processing

Transactional triggers for logging on and off:

Trigger	Do-the-Right-Thing Built-in
Pre-Logon	-
Pre-Logout	-
On-Logon	LOGON
On-Logout	LOGOUT
Post-Logon	-
Post-Logout	-



Copyright © 2006, Oracle. All rights reserved.

## Overriding Default Transaction Processing (continued)

### Transactional Triggers for Logging On and Off

Trigger	Do-the-Right-Thing Built-In
Pre-Logon	-
Pre-Logout	-
On-Logon	LOGON
On-Logout	LOGOUT
Post-Logon	-
Post-Logout	-

### Uses of Transactional Triggers

- Transactional triggers, except for the commit triggers, are primarily intended to access certain data sources other than Oracle.
- The logon and logoff transactional triggers can also be used with Oracle databases to change connections at run time.

# Running Against Data Sources Other than Oracle

- **Two ways to run against data sources other than Oracle:**
  - **Connecting with Open Gateway:**  
**Cursor and Savepoint mode form module properties**  
**Key mode and Locking mode block properties**
  - **Using transactional triggers:**  
**Call 3GL programs**  
**Database data block property**

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Running Against Data Sources Other than Oracle

### Two Ways to Run Against Data Sources Other than Oracle

- Use Oracle Transparent Gateway products.
- Write the appropriate set of transactional triggers.

### Connecting with Open Gateway

When you connect to a data source other than Oracle with an Open Gateway product, you should be aware of these transactional properties:

- Cursor mode form module property
- Savepoint mode form module property
- Key mode block property
- Locking mode block property

You can set these properties to specify how Forms should interact with your data source. The specific settings depend on the capabilities of the data source.

### Using Transactional Triggers

If no Open Gateway drivers exist for your data source, you must define transactional triggers. From these triggers, you must call 3GL programs that implement the access to the data source.

## Running Against Data Sources Other than Oracle (continued)

### Database Data Block Property

This block property identifies a block as a transactional control block; that is, a control block that should be treated as a base-table block. Setting this property to Yes ensures that transactional triggers will fire for the block, even though it is not a base-table block. If you set this property to Yes, you must define all On-Event transactional triggers, otherwise you will get an error during form generation.

Oracle Internal & OAI Use Only

# Getting and Setting the Commit Status

- **Commit status:** Determines how record will be processed
- **SYSTEM.RECORD\_STATUS:**
  - NEW
  - INSERT (also caused by control items)
  - QUERY
  - CHANGED
- **SYSTEM.BLOCK\_STATUS:**
  - NEW (may contain records with status INSERT)
  - QUERY (also possible for control block)
  - CHANGED (block will be committed)
- **SYSTEM.FORM\_STATUS:** NEW, QUERY, CHANGED

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Getting and Setting the Commit Status

If you want to process a record in your form, it is often useful to know if the record is in the database or if it has been changed, and so on. You can use system variables and built-ins to obtain this information.

### What Is the Commit Status of a Record?

The commit status of a record of a base table block determines how the record will be processed during the next commit process. For example, the record can be inserted, updated, or not processed at all.

## Getting and Setting the Commit Status (continued)

### The Four Values of `SYSTEM.RECORD_STATUS`

Value	Description
NEW	Indicates that the record has been created, but that none of its items have been changed yet (The record may have been populated by default values.)
INSERT	Indicates that one or more of the items in a newly created record have been changed (The record will be processed as an insert during the next commit process if its block has the CHANGED status; see below. Note that when you change a control item of a NEW record, the record status also becomes INSERT.)
QUERY	Indicates that the record corresponds to a row in the database, but that none of its base table items have been changed
CHANGED	Indicates that one or more base table items in a database record have been changed (The record will be processed as an update (or delete) during the next commit process.)

### The Three Values of `SYSTEM.BLOCK_STATUS`

Value	Description
NEW	Indicates that all records of the block have the status NEW (Note that a base table block with the status NEW may also contain records with the status INSERT caused by changing control items.)
QUERY	Indicates that all records of the block have the status QUERY if the block is a base table block (A control block has the status QUERY if it contains at least one record with the status INSERT.)
CHANGED	Indicates that the block contains at least one record with the status INSERT or CHANGED if the block is a base table block (The block will be processed during the next commit process. Note that a control block cannot have the status CHANGED.)

### The Three Values of `SYSTEM.FORM_STATUS`

Value	Description
NEW	Indicates that all blocks of the form have the status NEW
QUERY	Indicates that at least one block of the form has the status QUERY and all other blocks have the status NEW
CHANGED	Indicates that at least one block of the form has the status CHANGED

# Getting and Setting the Commit Status

- **System variables versus built-ins for commit status**
- **Built-ins for getting and setting commit status:**
  - GET\_BLOCK\_PROPERTY
  - GET\_RECORD\_PROPERTY
  - SET\_RECORD\_PROPERTY

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Using Built-ins to Get the Commit Status

The system variables SYSTEM.RECORD\_STATUS and SYSTEM.BLOCK\_STATUS apply to the record and block where the cursor is located. You can use built-ins to obtain the status of other blocks and records.

Built-in	Description
GET_BLOCK_PROPERTY	Use the STATUS property to obtain the block status of the specified block.
GET_RECORD_PROPERTY	Use the STATUS property to obtain the record status of the specified record in the specified block.
SET_RECORD_PROPERTY	Set the STATUS property of the specified record in the specified block to one of the following constants: <ul style="list-style-type: none"><li>• NEW_STATUS</li><li>• INSERT_STATUS</li><li>• QUERY_STATUS</li><li>• CHANGED_STATUS</li></ul>

# Getting and Setting the Commit Status

- **Example:** If the third record of block ORDERS is a changed database record, set the status back to QUERY.
- **Warnings:**
  - Do not confuse commit status with validation status.
  - The commit status is updated during validation.

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Using Built-ins to Get the Commit Status (continued)

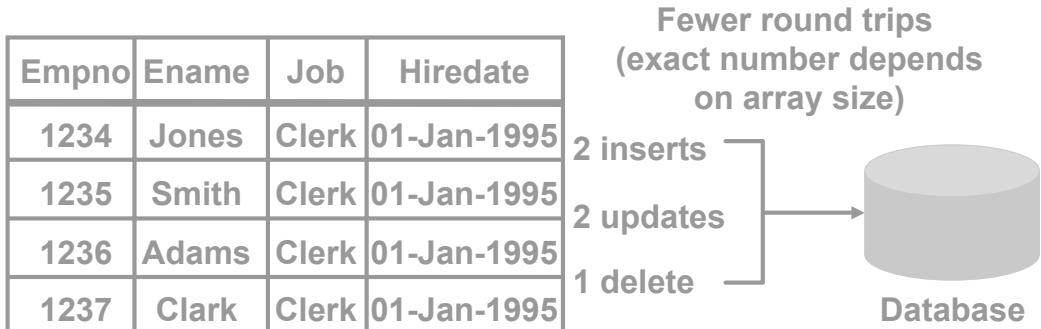
### Example

If the third record of the ORDERS block is a changed database record, set the status back to QUERY:

```
BEGIN
    IF GET_RECORD_PROPERTY(3, 'ORDERS',status) = 'CHANGED'
    THEN
        SET_RECORD_PROPERTY(3, 'ORDERS', status,
                            query_status);
    END IF;
END;
```

# Array DML

- **Performs array inserts, updates, and deletes**
- **Vastly reduces network traffic**



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Array Processing

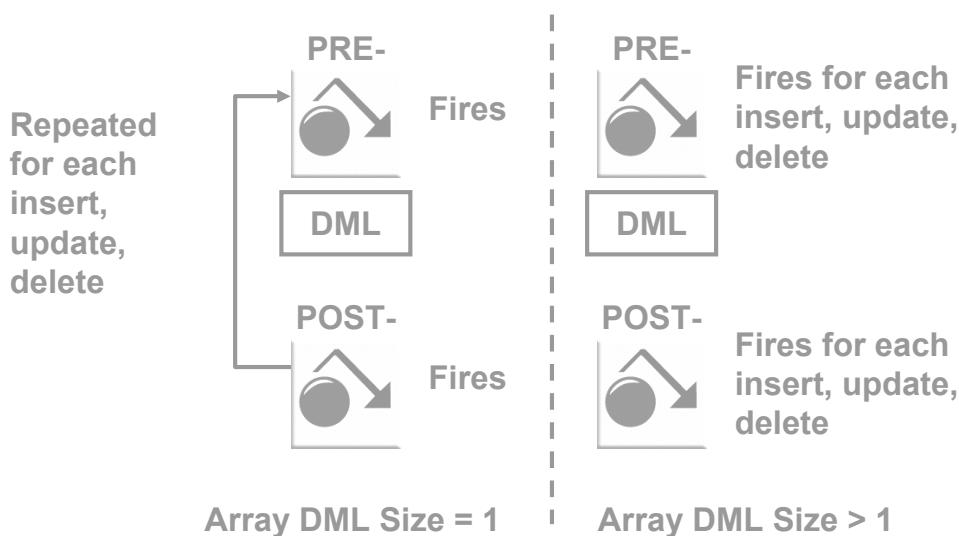
### Overview

Array processing is an option in Forms Builder that alters the way records are processed. The default behavior of Forms is to process records one at a time. By enabling array processing, you can process groups of records at a time, reducing network traffic and thereby increasing performance. This is especially important in Web applications. With array processing, a structure (an array) containing multiple records is sent to or returned from the server for processing.

Forms Builder supports both array fetch processing and array DML processing. For both querying and DML operations, you can determine the array size to optimize performance for your needs. This lesson focuses on array DML processing.

Array processing is available for query and DML operations for blocks based on tables, views, procedures, and subqueries; it is not supported for blocks based on transactional triggers.

# Effect of Array DML on Transactional Triggers



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Array Processing (continued)

### Effect of Array DML on Transactional Triggers

With DML Array Size set to 1, the Pre-Insert, Pre-Update, and Pre-Delete triggers fire for each new, changed, and deleted record; the DML is issued, and the Post- trigger for that record fires.

With DML Array Size set to greater than 1, the appropriate Pre- triggers fire for all of the new, changed, and deleted rows; all of the DML statements are issued, and all of the Post- triggers fire.

If you change 100 rows and DML Array Size is 20, you get 100 Pre- triggers, 5 arrays of 20 DML statements, and 100 Post- triggers.

# Implementing Array DML

1. Enable the Array Processing option.
2. Specify a DML Array Size of greater than 1.
3. Specify block primary keys.

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## How to Implement Array DML

1. To set preferences:
  - Select Edit > Preferences.
  - Click the Runtime tab.
  - Select the Array Processing check box.
2. To set properties:
  - In the Object Navigator, select the Data Blocks node.
  - Double-click the Data Blocks icon to display the Property Palette.
  - Under the Advanced Database category, set the DML Array Size property to a number that represents the number of records in the array for array processing. You can also set this property programmatically.

**Note:** When the DML Array Size property is greater than 1, you must specify the primary key. Key mode can still be unique.

The Oracle server uses the ROWID to identify the row, except after an array insert. If you update a record in the same session that you inserted it, the server locks the record by using the primary key.

## Summary

In this lesson, you should have learned that:

- To apply changes to the database, Forms issues post and commit
- The commit sequence of events is:
  1. Validate the form.
  2. Process savepoint.
  3. Fire Pre-Commit.
  4. Validate the block (performed for all blocks in sequential order).

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Summary

This lesson showed you how to build triggers that can perform additional tasks during the save stage of a current database transaction.

- Transactions are processed in two phases:
  - Post: Applies form changes to the base tables and fires transactional triggers
  - Commit: Commits the database transaction
- Flow of commit processing

## Summary

In this lesson, you should have learned that:

5. You can perform the DML:

Delete records: Fire Pre-Delete, delete row or fire

On-Delete, fire Post-Delete trigger

Insert records: Copy Value From Item, fire

Pre-Insert, check record uniqueness, insert row or  
fire On-Insert, fire Post-Insert

Update records: Fire Pre-Update, check record  
uniqueness, update row or fire On-Update, fire  
Post-Update

6. Fire Post-Forms-Commit trigger

If the current operation is COMMIT, then:

7. Issue a SQL-COMMIT statement

8. Fire the Post-Database-Commit trigger

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

Oracle Internal & OAI Use Only

# Summary

In this lesson, you should have learned that:

- You can supplement transaction processing with triggers:
  - Pre-Commit: Fires once if form changes are made or uncommitted changes are posted
  - [Pre | Post] – [Update | Insert | Delete]
  - On- [Update | Insert | Delete]:  
Fires per record, replacing default DML on row  
Perform default functions with built-ins:  
[UPDATE | INSERT | DELETE] \_ RECORD

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Summary (continued)

- DML statements issued during commit processing:
  - Based on base-table items
  - UPDATE and DELETE statements use ROWID by default.
- Characteristics of commit triggers:
  - The Pre-Commit, Post-Forms-Commit, and Post-Database-Commit triggers fire once per commit process, but consider uncommitted changes or posts.
  - The Pre-, On-, and Post-Insert, Update, and Delete triggers fire once per processed record.

## Summary

In this lesson, you should have learned that:

- You use the Pre-Insert trigger to allocate sequence numbers to records as they are applied to tables
- You check or change commit status:
  - GET\_BLOCK\_PROPERTY, [GET | SET]\_RECORD\_STATUS
  - :SYSTEM. [FORM | BLOCK | RECORD]\_STATUS
- You use transactional triggers to override or augment default commit processing
- You reduce network roundtrips by setting DML Array Size block property to implement Array DML

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Summary (continued)

- Common uses for commit triggers: Check authorization, set up special locking requirements, generate sequence numbers, check complex constraints, replace default DML statements issued by Forms.
- Overriding default transaction processing:
  - Transactional On-<Event> triggers and “Do-the-Right-Thing” built-ins
  - Data sources other than Oracle use Transparent Gateway or transactional triggers
- Getting and setting the commit status:
  - System variables
  - Built-ins
- Array DML

## Practice 21: Overview

This practice covers the following topics:

- Automatically populating order IDs by using a sequence
- Automatically populating item IDs by adding the current highest order ID
- Customizing the commit messages in the CUSTOMERS form
- Customizing the login screen in the CUSTOMERS form

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Practice 21: Overview

In this practice, you add transactional triggers to the ORDGXX form to automatically provide sequence numbers to records at save time. You also customize commit messages and the login screen in the CUSTGXX form.

- Automatically populating order IDs by using a sequence
- Automatically populating item IDs by adding the current highest order ID
- Customizing the commit messages in the CUSTOMERS form
- Customizing the login screen in the CUSTOMERS form

**Note:** For solutions to this practice, see Practice 21 in Appendix A, “Practice Solutions.”

## Practice 21

1. In the ORDGXX 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.
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.
3. Save, compile, and run the form to test.
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.
5. Save and compile the form. Click Run Form to run the form and test the changes.
6. Open the CUSTGXX form module. Create three global variables called GLOBAL.INSERT, GLOBAL.UPDATE, and GLOBAL.DELETE. These variables indicate the number of inserts, updates, and deletes, respectively. You need to write Post-Insert, Post-Update, and Post-Delete triggers to initialize and increment the value of each global variable. You may import the code from pr21\_6a.txt, pr21\_6b.txt, and pr21\_6c.txt.  
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.
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 message line to be displayed. This procedure uses the three global variables to display a customized commit message and then erases the global variables.  
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.
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.
9. Click Run Form to run the form and test the changes.



# Writing Flexible Code

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

Oracle Internal & OAI Use Only

# Objectives

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

- **Describe flexible code**
- **State the advantages of using system variables**
- **Identify the built-in subprograms that assist flexible coding**
- **Write code to reference objects:**
  - By internal ID
  - Indirectly

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Introduction

### Overview

In this lesson, you learn about the Forms Builder features that enable you to write code in a flexible, reusable way.

# What Is Flexible Code?

## Flexible code:

- Is reusable
- Is generic
- Avoids hard-coded object names
- Makes maintenance easier
- Increases productivity

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## What Is Flexible Code?

*Flexible code* is code that you can use again. It is often generic code that you can use in any form module in an application. It typically includes the use of system variables instead of hard-coded object names.

## Why Write Flexible Code?

Writing flexible code gives you the following advantages:

- It is easier for you and others to maintain.
- It increases productivity.

# Using System Variables for Current Context

- **Input focus:**
  - **SYSTEM.CURSOR\_BLOCK**
  - **SYSTEM.CURSOR\_RECORD**
  - **SYSTEM.CURSOR\_ITEM**
  - **SYSTEM.CURSOR\_VALUE**

```
IF :SYSTEM.CURSOR_BLOCK = 'ORDERS' THEN
    GO_BLOCK('ORDER_ITEMS');
ELSIF :SYSTEM.CURSOR_BLOCK = 'ORDER_ITEMS' THEN
    GO_BLOCK('INVENTORIES');
ELSIF :SYSTEM.CURSOR_BLOCK = 'INVENTORIES' THEN
    GO_BLOCK('ORDERS');
END IF;
```

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Using System Variables for Current Context

In this lesson, you use the system variables that provide the current status of the record, the block, and the form, as well as system variables that return the current input focus location.

### System Variables for Locating Current Input Focus

System Variable	Function
CURSOR_BLOCK	The block that has the input focus
CURSOR_RECORD	The record that has the input focus
CURSOR_ITEM	The item and block that has the input focus
CURSOR_VALUE	The value of the item with the input focus

### Example

The example in the above shows code that could be put in a When-Button-Pressed trigger to enable users to navigate to another block in the form. It tests the current block name, then navigates depending on the result.

**Note:** Be sure to set the button's Mouse Navigate property to No; otherwise, the :SYSTEM.CURSOR\_BLOCK will always be the block on which the button is located.

# Using System Variables for Current Context

- **Trigger focus:**
  - SYSTEM.TRIGGER\_BLOCK
  - SYSTEM.TRIGGER\_RECORD
  - SYSTEM.TRIGGER\_ITEM

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Using System Variables for Current Context (continued)

### System Variables for Locating Trigger Focus

System Variable	Function
TRIGGER_BLOCK	The block that the input focus was in when the trigger initially fired
TRIGGER_RECORD	The number of the record that Forms is processing
TRIGGER_ITEM	The block and item that the input focus was in when the trigger initially fired

### Uses for Trigger Focus Variables

The variables for locating trigger focus are useful for navigating back to the initial block, record, and item after the trigger code completes. For example, the trigger code may navigate to other blocks, records, or items to perform actions upon them, but after the trigger fires, you may want the cursor to be in the same item instance that it was in originally. Because the navigation in the trigger occurs behind the scenes, the user will not even be aware of it.

**Note:** The best way to learn about system variables is to look at their values when a form is running. You can examine the system variables by using the Debugger.

# System Status Variables

## When-Button-Pressed

```
ENTER;  
IF :SYSTEM.BLOCK_STATUS = 'CHANGED' THEN  
    COMMIT_FORM;  
END IF;  
CLEAR_BLOCK;
```

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## System Variables for Determining the Current Status of the Form

You can use these system status variables presented in the previous lesson to write the code that performs one action for one particular status and a different action for another:

- SYSTEM.RECORD\_STATUS
- SYSTEM.BLOCK\_STATUS
- SYSTEM.FORM\_STATUS

The example in the slide performs a commit before clearing a block if there are changes to commit within that block.

## **GET\_<object>\_PROPERTY Built-Ins**

- **GET\_APPLICATION\_PROPERTY**
- **GET\_FORM\_PROPERTY**
- **GET\_BLOCK\_PROPERTY**
- **GET\_RELATION\_PROPERTY**
- **GET\_RECORD\_PROPERTY**
- **GET\_ITEM\_PROPERTY**
- **GET\_ITEM\_INSTANCE\_PROPERTY**

**ORACLE®**

Copyright © 2006, Oracle. All rights reserved.

### **Using Built-in Subprograms for Flexible Coding**

Some of Forms Builder built-in subprograms provide the same type of run-time status information that built-in system variables provide.

#### **GET\_APPLICATION\_PROPERTY**

The GET\_APPLICATION\_PROPERTY built-in returns information about the current Forms application.

#### **Example**

The following example captures the username and the operating system information:

```
:GLOBAL.username := GET_APPLICATION_PROPERTY(USERNAME) ;
:GLOBAL.o_sys :=
    GET_APPLICATION_PROPERTY(OPERATING_SYSTEM) ;
```

**Note:** The GET\_APPLICATION\_PROPERTY built-in returns information about the Forms application running on the middle tier. If you require information about the client machine, you can use a JavaBean.

## **GET\_<object>\_PROPERTY Built-Ins**

- **GET\_LOV\_PROPERTY**
- **GET\_RADIO\_BUTTON\_PROPERTY**
- **GET\_MENU\_ITEM\_PROPERTY**
- **GET\_CANVAS\_PROPERTY**
- **GET\_TAB\_PAGE\_PROPERTY**
- **GET\_VIEW\_PROPERTY**
- **GET\_WINDOW\_PROPERTY**

**ORACLE®**

Copyright © 2006, Oracle. All rights reserved.

### **Using Built-in Subprograms for Flexible Coding (continued)**

#### **GET\_BLOCK\_PROPERTY**

The GET\_BLOCK\_PROPERTY built-in returns information about a specified block.

##### **Example**

To determine the current record that is visible at the first (top) line of a block:

```
...GET_BLOCK_PROPERTY('blockname',top_record)...
```

#### **GET\_ITEM\_PROPERTY**

The GET\_ITEM\_PROPERTY built-in returns information about a specified item.

##### **Example**

To determine the canvas that the item with the input focus displays on, use:

```
DECLARE
  cv_name varchar2(30);
BEGIN
  cv_name :=
    GET_ITEM_PROPERTY(:SYSTEM.CURSOR_ITEM,item_canvas);
  ...

```

## **SET\_<object>\_PROPERTY Built-Ins**

- **SET\_APPLICATION\_PROPERTY**
- **SET\_FORM\_PROPERTY**
- **SET\_BLOCK\_PROPERTY**
- **SET\_RELATION\_PROPERTY**
- **SET\_RECORD\_PROPERTY**
- **SET\_ITEM\_PROPERTY**
- **SET\_ITEM\_INSTANCE\_PROPERTY**

**ORACLE®**

Copyright © 2006, Oracle. All rights reserved.

### **SET\_<object>\_PROPERTY Built-Ins**

#### **SET\_ITEM\_INSTANCE\_PROPERTY**

The **SET\_ITEM\_INSTANCE\_PROPERTY** built-in modifies the specified instance of an item in a block by changing the specified item property.

##### **Example**

The following example sets the visual attribute to **VA\_CURR** for the current record of the current item:

```
SET_ITEM_INSTANCE_PROPERTY (:SYSTEM.CURSOR_ITEM,  
                           VISUAL_ATTRIBUTE, CURRENT_RECORD, 'VA_CURR');
```

#### **SET\_MENU\_ITEM\_PROPERTY**

The **SET\_MENU\_ITEM\_PROPERTY** built-in modifies the given properties of a menu item.

##### **Example**

To enable the save menu item in a file menu, use:

```
SET_MENU_ITEM_PROPERTY ('FILE.SAVE', ENABLED, PROPERTY_TRUE);
```

## **SET\_<object>\_PROPERTY Built-Ins**

- **SET\_LOV\_PROPERTY**
- **SET\_RADIO\_BUTTON\_PROPERTY**
- **SET\_MENU\_ITEM\_PROPERTY**
- **SET\_CANVAS\_PROPERTY**
- **SET\_TAB\_PAGE\_PROPERTY**
- **SET\_VIEW\_PROPERTY**
- **SET\_WINDOW\_PROPERTY**

**ORACLE®**

Copyright © 2006, Oracle. All rights reserved.

### **SET\_<object>\_PROPERTY Built-Ins (continued)**

#### **SET\_TAB\_PAGE\_PROPERTY**

The SET\_TAB\_PAGE\_PROPERTY built-in sets the tab page properties of the specified tab canvas page.

#### **Example**

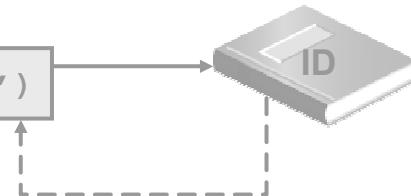
To enable tab\_page\_1, if it is already disabled, use:

```
DECLARE
    tbpg_id  TAB_PAGE;
BEGIN
    tbpg_id := FIND_TAB_PAGE('tab_page_1');
    IF GET_TAB_PAGE_PROPERTY(tbpg_id, enabled) = 'FALSE' THEN
        SET_TAB_PAGE_PROPERTY(tbpg_id, enabled,property_true);
    END IF;
END;
```

# Referencing Objects by Internal ID

## Finding the object ID:

```
lov_id := FIND_LOV('my_lov')
```



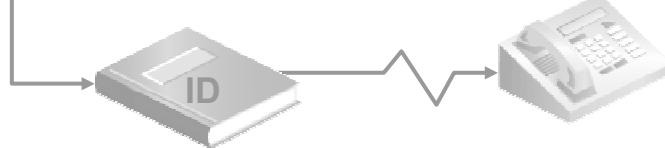
## Referencing an object by ID:

```
...SHOW_LOV(lov_id)
```



## Referencing an object by name:

```
...SHOW_LOV('my_lov')
```



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Referencing Objects by Internal ID

Forms Builder assigns an ID to each object that you create. An object ID is an internal value that is never displayed. You can get the ID of an object by calling the built-in FIND\_ subprogram appropriate for the object. The FIND\_ subprograms require a fully qualified object name as a parameter. For instance, when referring to an item, use *BLOCKNAME.ITEMNAME*.

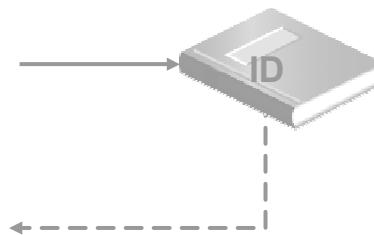
The return values of the FIND\_ subprograms (the object IDs) are of a specific type. The types for object IDs are predefined in Forms Builder. There is a different type for each object.

### Three Reasons for Using Object IDs

- Improving performance (Forms looks up the object only once when you initially call the FIND\_ subprogram to get the ID. When you refer to an object by name in a trigger, Forms must look up the object ID each time.)
- Writing more generic code
- Testing whether an object exists (using the ID\_NULL function and FIND\_object)

## FIND\_ Built-Ins

- FIND\_ALERT
- FIND\_BLOCK
- FIND\_CANVAS
- FIND\_EDITOR
- FIND\_FORM
- FIND\_ITEM
- FIND\_LOV
- FIND\_RELATION
- FIND\_VIEW
- FIND\_WINDOW



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Forms Builder FIND\_ Built-Ins

The following table lists some of the FIND\_ subprograms, along with the object classes that use them and the return types they produce:

Object Class	Subprogram	Return Type
Alert	FIND_ALERT	ALERT
Block	FIND_BLOCK	BLOCK
Canvas	FIND_CANVAS	CANVAS
Editor	FIND_EDITOR	EDITOR
Form	FIND_FORM	FORMMODULE
Item	FIND_ITEM	ITEM
LOV	FIND_LOV	LOV
Relation	FIND_RELATION	RELATION
View	FIND_VIEW	VIEWPORT
Window	FIND_WINDOW	WINDOW

# Using Object IDs

- **Declare a PL/SQL variable of the same data type.**
- **Use the variable for any later reference to the object.**
- **Use the variable within the current PL/SQL block only.**

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Declaring Variables for Object IDs

To use an object ID, you must first assign it to a variable. You must declare a variable of the same type as the object ID.

The following example uses the `FIND_ITEM` built-in to assign the ID of the item that currently has input focus to the `id_var` variable.

After you assign an object ID to a variable in a trigger or PL/SQL program unit, you can use that variable to reference the object, rather than referring to the object by name:

```
DECLARE
    id_var item;
BEGIN
    id_var := FIND_ITEM(:SYSTEM.CURSOR_ITEM);
    .
    .
    .
END;
```

# Using Object IDs

## Example:

```
DECLARE
    item_var item;
BEGIN
    item_var := FIND_ITEM(:SYSTEM.CURSOR_ITEM);
    SET_ITEM_PROPERTY(item_var,position,30,55);
    SET_ITEM_PROPERTY(item_var,prompt_text,'Current');
END;
```

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Declaring Variables for Object IDs (continued)

The following two examples show that you can pass either an item name or an item ID to the SET\_ITEM\_PROPERTY built-in subprogram. The following calls are logically equivalent:

```
SET_ITEM_PROPERTY('ORDERS.order_id',position,50,35);
SET_ITEM_PROPERTY(id_var,position,50,35);
```

You can use either object IDs or object names in the same argument list, provided that each individual argument refers to a distinct object.

You cannot, however, use an object ID and an object name to form a fully qualified object\_name(blockname.itemname). The following call is illegal:

```
GO_ITEM(block_id.'item_name');
```

**Note:** Use the FIND\_ built-in subprograms only when referring to an object more than once in the same trigger or PL/SQL program unit.

# Increasing the Scope of Object IDs

- A PL/SQL variable has limited scope.
- An .id extension:
  - Broadens the scope
  - Converts to a numeric format
  - Enables assignment to a global variable
  - Converts back to the object data type

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Using Object IDs Outside the Initial PL/SQL Block

You have seen how object IDs are referenced within the trigger or program unit by means of PL/SQL variables. You can reference these PL/SQL variables only in the current PL/SQL block; however, you can increase the scope of an object ID.

To reference an object ID outside the initial PL/SQL block, you need to convert the ID to a numeric format using an .id extension for your declared PL/SQL variable, then assign it to a global variable.

### Example

The following example of trigger code assigns the object ID to a local PL/SQL variable (`item_var`) initially, then to a global variable (`:GLOBAL.item`):

```
DECLARE
    item_var item;
BEGIN
    item_var := FIND_ITEM(:SYSTEM.CURSOR_ITEM);
    :GLOBAL.item := item_var.id;
END;
```

## Using Object IDs Outside the Initial PL/SQL Block (continued)

You can pass the global variable around within the application. To be able to reuse the object ID, you need to convert it back to its original data type.

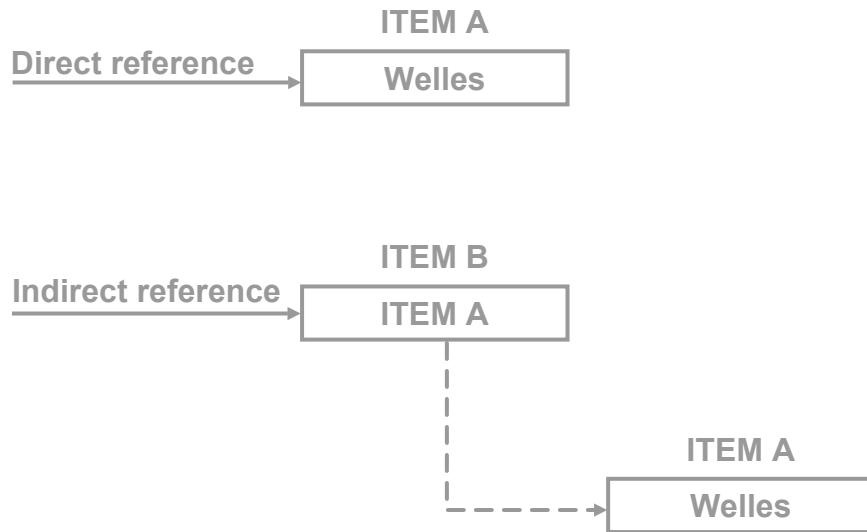
### Example

The following example shows the conversion of the global variable back to its original PL/SQL variable data type:

```
DECLARE
    item_var item;
BEGIN
    item_var.id := TO_NUMBER(:GLOBAL.item);
    GO_ITEM(item_var);
END;
```

Oracle Internal & OAI Use Only

## Referencing Items Indirectly



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Referencing Items Indirectly

By referencing items indirectly, you can write more generic, reusable code. Using variables instead of actual item names, you can write a PL/SQL program unit to use any item whose name is assigned to the indicated variable.

You can reference items indirectly with the NAME\_IN and COPY built-in subprograms.

**Note:** Use indirect referencing when you create procedures and functions in a library module, because direct references cannot be resolved.

## Referencing Items Indirectly

- **The NAME\_IN function returns:**
  - The contents of a variable
  - Character string
- **Use conversion functions for NUMBER and DATE.**

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Referencing Items Indirectly (continued)

#### Using the NAME\_IN Built-in Function

The NAME\_IN function returns the contents of an indicated variable. The following statements are equivalent. The first one uses a direct reference to customer.name, whereas the second uses an indirect reference:

```
IF :CUSTOMERS.cust_last_name = 'Welles'...
```

In a library, you could avoid this direct reference by using:

```
IF NAME_IN('CUSTOMERS.cust_last_name') = 'Welles'...
```

The return value of NAME\_IN is always a character string. To use NAME\_IN for a date or number item, convert the string to the desired data type with the appropriate conversion function. For instance:

```
date_var := TO_DATE(NAME_IN('ORDERS.order_date'));
```

# Referencing Items Indirectly

The **COPY** procedure enables:

- Direct copy:

```
COPY('Welles','CUSTOMERS.cust_last_name');
```

- Indirect copy:

```
COPY('Welles',NAME_IN('global.customer_name_item'));
```

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Referencing Items Indirectly (continued)

### Using the **COPY** Built-in Procedure

The **COPY** built-in assigns an indicated value to an indicated variable or item. Unlike the standard PL/SQL assignment statement, using the **COPY** built-in enables you to indirectly reference the item whose value is being set. The first example in the slide shows copying using a direct reference to the form item.

### Using **COPY** with **NAME\_IN**

Use the **COPY** built-in subprogram with the **NAME\_IN** built-in to indirectly assign a value to an item whose name is stored in a global variable, as in the second example in the slide.

## Summary

In this lesson, you should have learned that:

- Flexible code is reusable, generic code that you can use in any form module in an application
- With system variables, you can:
  - Perform actions conditionally based on current location (`SYSTEM.CURSOR_[RECORD | ITEM | BLOCK]`)
  - Use the value of an item without knowing its name (`SYSTEM.CURSOR_VALUE`)
  - Navigate to the initial location after a trigger completes: (`SYSTEM.TRIGGER_[RECORD | ITEM | BLOCK]`)
  - Perform actions conditionally based on commit status: `SYSTEM.[RECORD | BLOCK | FORM]_STATUS`

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Summary

Use the following to write flexible code:

- System variables:
  - To avoid hard-coding object names
  - To return information about the current state of the form

## Summary

In this lesson, you should have learned that:

- The [GET | SET]\_<object>\_PROPERTY built-ins are useful in flexible coding
- Code that references objects is more efficient and generic:
  - By internal ID: Use FIND\_<object> built-ins
  - Indirectly: Use COPY and NAME\_IN built-ins

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Summary (continued)

- GET\_<object>\_PROPERTY built-ins, to return current property values for Forms Builder objects
- Object IDs, to improve performance
- Indirect referencing, to allow form module variables to be referenced in library and menu modules

## Practice 22: Overview

This practice covers the following topics:

- Populating product images only when the image item is displayed
- Modifying the When-Button-Pressed trigger of Image\_Button in order to use object IDs instead of object names
- Writing generic code to print out the names of the blocks in a form

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Practice 22: Overview

In this practice, you use properties and variables in the ORDGXX form to provide flexible use of its code. You also make use of object IDs.

- Populating product images only when the image item is displayed
- Modifying the When-Button-Pressed trigger of Image\_Button in order to use object IDs instead of object names
- Writing generic code to print out the names of blocks in a form and using the same code in two different forms

**Note:** For solutions to this practice, see Practice 22 in Appendix A, “Practice Solutions.”

## Practice 22

1. In the ORDGXX 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.
2. Alter the When-Button-Pressed trigger on 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.
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.
4. Save, compile, and run the form to test these features.
5. The trigger code above is generic, so it will work with any form. Open the CUSTGXX 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 ORDGXX form to the Blocks\_Button of the CUSTGXX form. Run the CUSTGXX form to test the button.



# Sharing Objects and Code

23

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

Oracle Internal & OAI Use Only

# Objectives

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

- Describe the various methods for reusing objects and code
- Inherit properties from property classes
- Group related objects for reuse
- Explain the inheritance symbols in the Property Palette
- Reuse objects from an object library
- Reuse PL/SQL code

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Introduction

### Overview

Forms Builder includes some features specifically for object and code reuse. In this lesson, you learn how to share objects between form modules using the Object Library. You also learn how to share code using the PL/SQL Library.

# Benefits of Reusable Objects and Code

- **Increases productivity**
- **Decreases maintenance**
- **Increases modularity**
- **Maintains standards**
- **Improves application performance**

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Benefits of Reusable Objects and Code

When you are developing applications, you should share and reuse objects and code wherever possible in order to:

- **Increase productivity:** You can develop applications much more effectively and efficiently if you are not trying to “start over” each time you write a piece of code. By sharing and reusing frequently used objects and code, you can cut down development time and increase productivity.
- **Decrease maintenance:** By creating applications that use or call the same object or piece of code several times, you can decrease maintenance time.
- **Increase modularity:** Sharing and reusing code increases the modularity of your applications.
- **Maintain standards:** You can maintain standards by reusing objects and code. If you create an object once and copy it again and again, you do not run the risk of introducing minor changes. In fact, you can create a set of standard objects and some pieces of standard code and use them as a starting point for all of your new form modules.

## Benefits of Reusable Objects and Code (continued)

- **Improve application performance:** When Forms Services communicates the user interface to the Forms Client, it sends metadata about the items on the form. This metadata includes values of properties that differ from the default. After an item is defined, the metadata about the next item includes only those properties that differ from the previous item. This is referred to as message diffing. Promoting similarities among items by using the methods of object reuse presented in this lesson improves the efficiency of message diffing and thus decreases network traffic and increases performance.

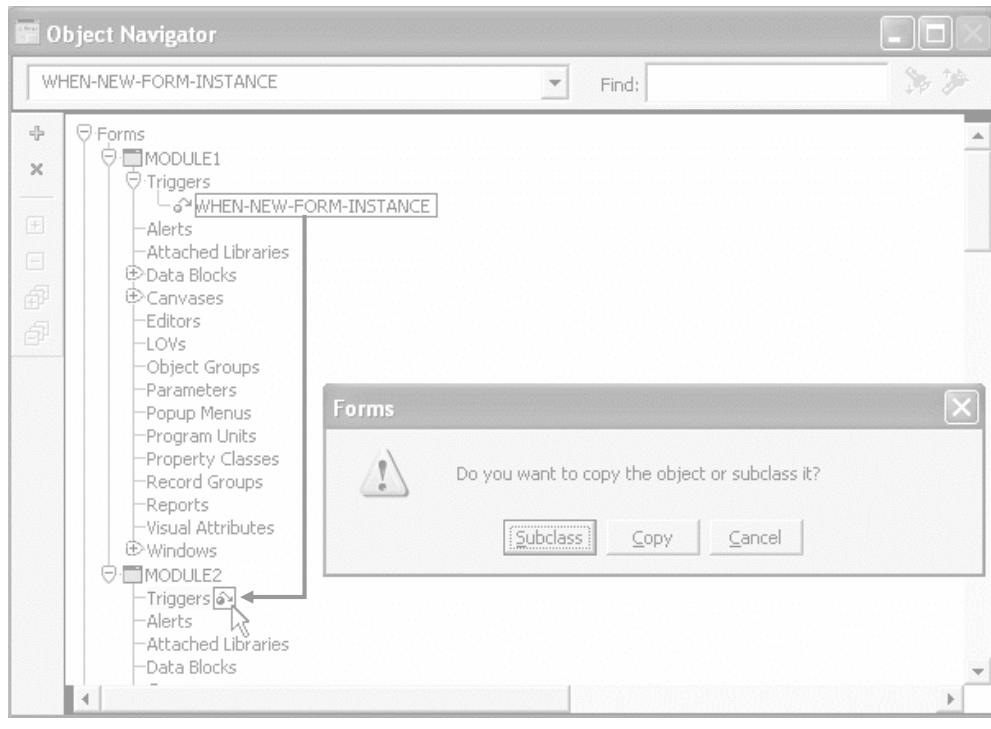
## Promoting Similarities Among Objects

One of the easiest ways a developer can increase the efficiency of network performance through message diffing is by using consistent standards for all objects within an application. Items of different types should at least have common values for common or shared properties.

To maximize reuse, the developer should apply the following guidelines in the order shown:

- **Accept default properties as much as possible:** If the properties are not overwritten for each object, then the value for common properties will be the same regardless of the object type, except for position and size.
- **Use SmartClasses to describe an object:** If, because of design standards, the use of default properties is not a viable option, then the subclassing of objects from a set of SmartClasses ensures that the development standards are being met. It also forces a high degree of property sharing across widgets. Items of the same type will then have (unless overridden) the same properties and, therefore, will be able to share properties more effectively. You will learn about SmartClasses in this lesson.
- **Use sets of visual attributes:** If SmartClasses are not being used to enforce standards and common properties, then use sets of partial visual attributes to enforce a common set of properties across objects of different types (for example, font, font size, foreground color, background color, and so on). These sets of visual attributes can be defined as Property Classes, as explained in the following slides.

# Copying and Subclassing Objects and Code



Copyright © 2006, Oracle. All rights reserved.

ORACLE®

## Copying and Subclassing Objects and Code

You can copy or subclass objects:

- Between modules, by dragging objects between the modules in the Object Navigator
- Within a single module by selecting the object in the Object Navigator, pressing [Ctrl], and dragging it to create the new object

When you drag objects, a dialog box appears that asks whether you want to copy or subclass the object.

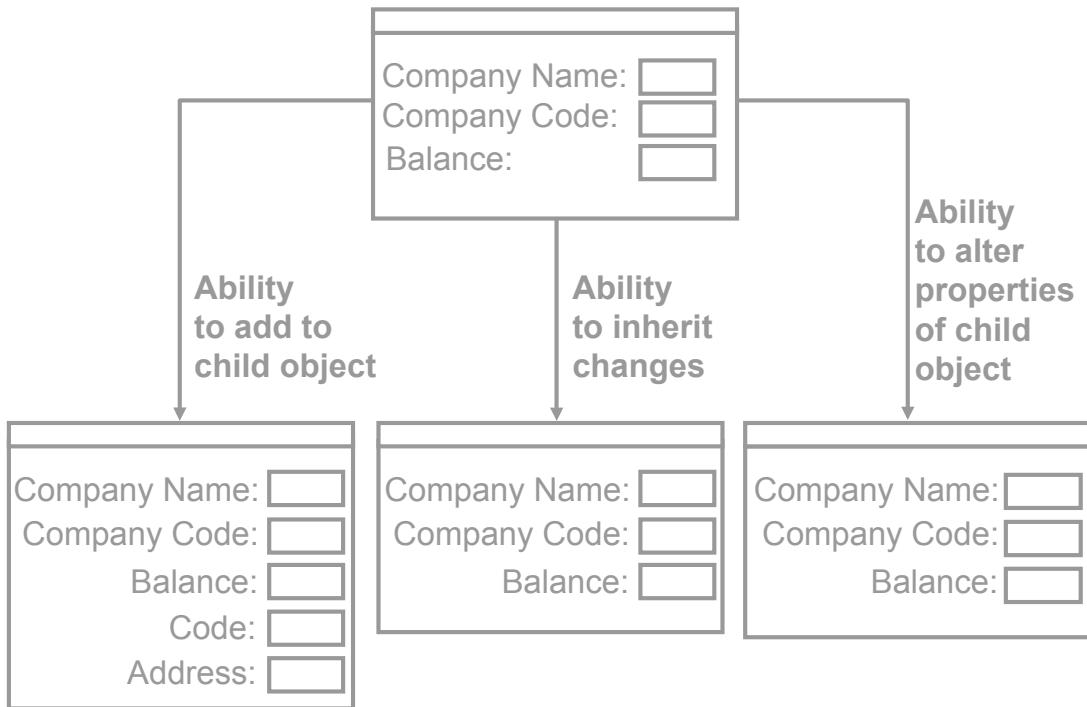
### Copying an Object

Copying an object creates a separate, unique version of that object in the target module. Any objects owned by the copied object are also copied.

### Points to Remember

- Use copying to export the definition of an object to another module.
- Changes made to a copied object in the source module do not affect the copied object in the target module.

# Subclassing



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Subclassing an Object

With subclassing, you can make an exact copy, and then alter some of the properties of the object if desired. If you change the parent class, the changes also apply to those properties of the subclassed object that you have not altered. However, any properties that you override remain overridden. This provides a powerful object inheritance model.

When you subclass a data block, you can:

- Change the structure of the parent, automatically propagating the changes to the child
- Add or change properties to the child to override the inheritance

When you subclass a data block, you cannot:

- Delete items from the child
- Change the order of items in the child
- Add items to the child unless you add them to the end

**Note:** Subclassing is an object-oriented term that refers to the following capabilities:

- Inheriting the characteristics of a base class (Inheritance)
- Overriding properties of the base class (Specialization)

## **Subclassing an Object (continued)**

### **Ability to Add to an Object**

You can create an exact copy of an object, and you can add to the subclassed object. For example, you can add additional items to the end of a subclassed block.

### **Ability to Alter Properties**

With subclassing, you can make an exact copy and then alter the properties of some objects. If you change the parent class, the changes also apply to the properties of the subclassed object that you have not altered. However, any properties that you override remain overridden.

### **Ability to Inherit Changes**

When you change the properties of a parent object, all child objects inherit those properties if they are not already overridden.

The child inherits changes:

- Immediately, if the parent and child objects are in the same form
- When you reload the form containing a child object

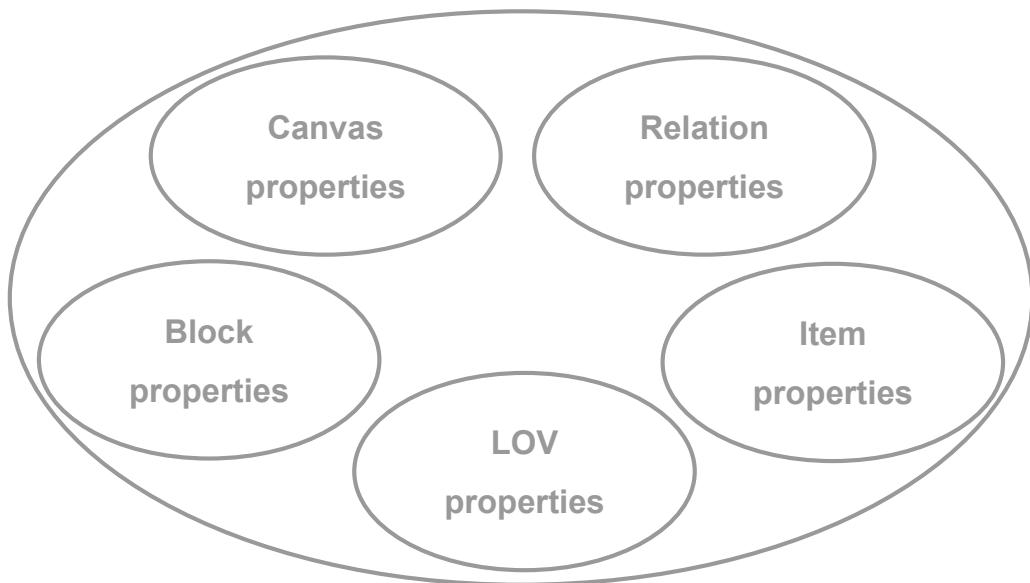
### **Ability to Reinherit**

If you make changes to the child object to override properties of the parent object, you can click the Inherit button to reinherit the property from the parent object.

**Property Palette icons:** Enable you to identify inherited or overridden properties

<b>Property Palette Icon</b>	<b>Meaning</b>
Circle	The value for the property is the default.
Square	The value for the property was changed from the default.
Arrow	The value for the property was inherited.
Arrow with red X	The value for the property was inherited but overridden (variant property).

# What Are Property Classes?



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Property Class

### What Is a Property Class?

A *property class* is a named object that contains a list of properties and their settings.

### Why Use Property Classes?

Use property classes to:

- Increase productivity by setting standard or frequently used values for common properties and associating them with several Forms Builder objects. You can use property classes to define standard properties not just for one particular object, but for several at a time. This results in increased productivity, because it eliminates the time spent on setting identical properties for several objects.
- Improve network performance by increasing the efficiency of message diffing.

# Creating a Property Class

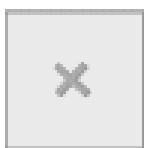
Add Property



Inherit Property



Delete Property



Property Class



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Creating a Property Class

When you create a property class, you have all the properties from every Forms Builder object available. You choose the properties and their values to include in the property class. You can create a property class in two ways:

- Using the Create button in the Object Navigator
- Using the Create Property Class button

### How to Create a Property Class from the Object Navigator

1. Select the Property Class node.
2. Click Create. A new property class entry displays.
3. In the Property Palette, use the Add Property button to add the required properties and their values.

### How to Create a Property Class from the Property Palette

1. In the Object Navigator, double-click the icon of the object whose properties you want to copy into a property class.
2. In the Property Palette, select the properties you want to copy into a property class.
3. Click the Property Class button. An information alert is displayed.
4. Use the Object Navigator to locate the property class and change its name.

## **Creating a Property Class (continued)**

### **Adding a Property**

After you create a property class, you can add a property by clicking the Add Property button and selecting a property from the list. Set the value for that property using the Property Palette.

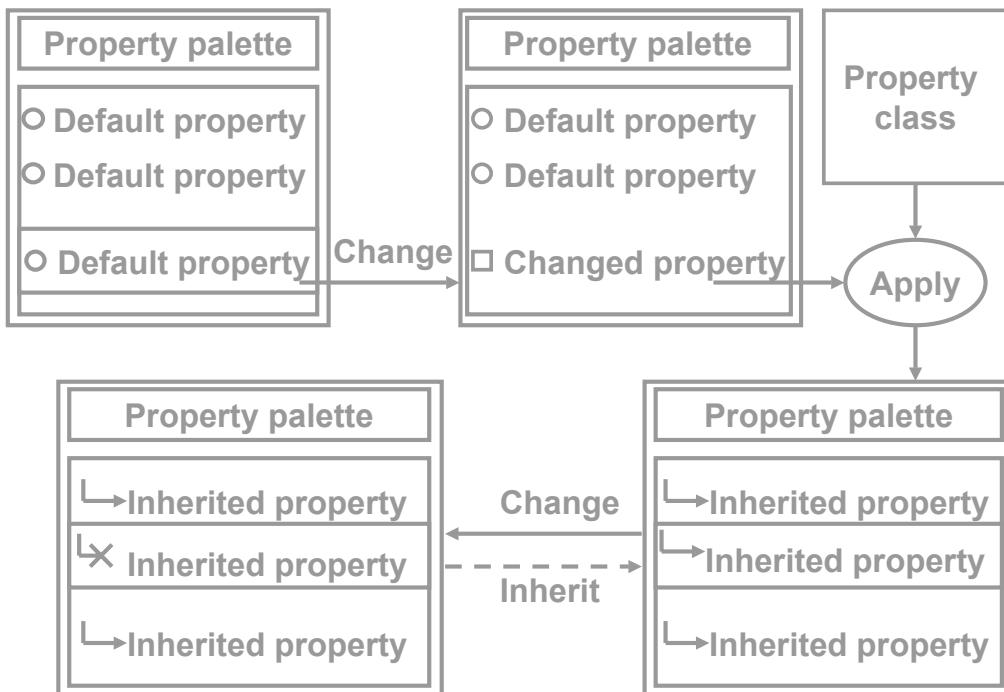
You can also use the Copy Properties button and the Paste Properties button to add several properties at a time to a property class.

### **Deleting a Property**

You can remove properties from a property class using the Delete Property button.

Oracle Internal & OAI Use Only

## Inheriting from a Property Class



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Inheriting from a Property Class

After you create a property class and add properties, you can use the property class. To apply the properties from a property class to an object, use the Subclass Information property in the Property Palette.

#### What Is an Inherited Property?

An *inherited property* is one that takes its value from the property class that you associated with the object. An inherited property is displayed with an arrow to the left of the property name.

#### What Is a Variant Property?

A *variant property* is one whose value has been modified from the inherited value that comes from the property class associated with the object. You can override the setting of any inherited property to make that property variant. Variant properties are displayed with a red cross over an arrow.

# Inheriting from a Property Class

- Set the Subclass Information property.
- Convert an inherited property to a variant property.
- Convert a variant property to an inherited property.
- Convert a changed property to a default property.

Inherited property	<input type="button" value="Background Color"/> r100g0b50	...
Variant property	<input type="button" value="Background Color"/> r100g100b50	...
Default property	<input type="button" value="Fill Pattern"/> <Unspecified>	...
Changed property	<input type="button" value="Fill Pattern"/> v45waves	...

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Inheriting from a Property Class (continued)

### How to Inherit Property Values from a Property Class

1. In the Object Navigator, double-click the object to which you want to apply the properties from the property class.
2. Click the Subclass Information property in the Property Palette.
3. Select the property class whose properties you want to use. The object takes on the values of that property class. Inherited properties are displayed with an arrow symbol.

### Converting an Inherited Property to a Variant Property

To convert an inherited property to a variant property, simply enter a new value over the inherited one.

### Converting a Variant Property to an Inherited Property

To convert a variant property to an inherited property, click the Inherit button in the Property Palette.

### Converting a Changed Property to a Default Property

You can also use the Inherit button to revert a changed property to its default.

# What Are Object Groups?

## Object groups:

- Are logical containers
- Enable you to:
  - Group related objects
  - Copy multiple objects in one operation

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## What Are Object Groups?

An *object group* is a logical container for a set of Forms Builder objects.

## Why Use Object Groups?

You define an object group when you want to:

- Package related objects for copying or subclassing in another module
- Bundle numerous objects into higher-level building blocks that you can use again in another application

## Example

Your application can include an appointment scheduler that you want to make available to other applications. You can package the various objects in an object group and copy the entire bundle in one operation.

# Creating and Using Object Groups

- **Blocks include:**
  - Items
  - Item-level triggers
  - Block-level triggers
  - Relations
- **Object groups cannot include other object groups.**
- **Deleting an object group does not affect the objects.**
- **Deleting an object affects the object group.**

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Creating and Using Object Groups

### How to Create an Object Group

1. Click the Object Group node in the Object Navigator.
2. Click the Create button. A new object group entry is displayed.
3. Rename the new object group.
4. Select the form module and, from the menu, select View > Expand All.
5. Control-click all the objects of one type that you want to include in the object group.
6. Drag the selected objects into the new object group entry. The objects are displayed as object group children.
7. Repeat steps 5 and 6 for different object types.

The objects in the object group are still displayed in their usual position in the Object Navigator, as well as within the object group. The objects in the object group are not duplicates, but pointers to the source objects.

## **Creating and Using Object Groups (continued)**

### **Things to Consider when Using Object Groups**

- Including a block in an object group also includes its items, the item-level triggers, the block-level triggers, and the relations. You cannot use any of these objects in an object group without the block.
- It is not possible to include another object group.
- Deleting an object from a module automatically deletes the object from the object group.
- Deleting an object group from a module does not delete the objects it contains from the module.

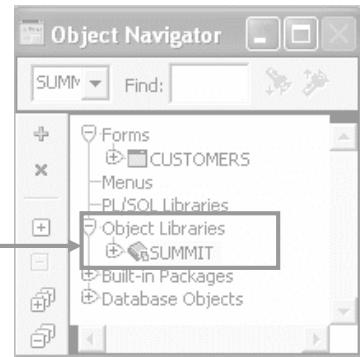
### **Subclass Information Dialog Box**

The Subclass Information property of a form object shows a dialog box that provides information about the origins of the object. You can see whether an object is local to the form document or foreign to it. If the object is foreign to the current form, then the Module field shows the module from which the object originates. The original object name is shown in the Object Name field.

# What Are Object Libraries?

## An Object library:

- Is a convenient container of objects for reuse
- Simplifies reuse in complex environments
- Supports corporate, project, and personal standards
- Simplifies the sharing of reusable components
- Is separate from the form module



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## What Are Object Libraries?

*Object libraries* are convenient containers of objects for reuse. They simplify reuse in complex environments, and they support corporate, project, and personal standards.

An object library can contain simple objects, property classes, object groups, and program units, but they are protected against change in the library. Objects can be used as standards (classes) for other objects.

Object libraries simplify the sharing of reusable components. Reusing components enables you to:

- Apply standards to simple objects, such as buttons and items, for consistent look and feel. This also improves network performance by promoting similarities among objects, thus increasing the efficiency of message diffing.
- Reuse complex objects such as a Navigator

In combination with SmartClasses, which are discussed later, object libraries support both of these requirements.

## **What Are Object Libraries? (continued)**

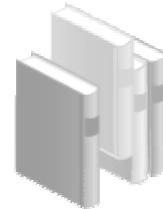
### **Why Object Libraries Instead of Object Groups?**

- Object libraries are external to the form, so are easily shared among form modules.
- Object libraries can contain individual items (for example, iconic buttons). The smallest unit accepted in an object group is a block.
- Object libraries accept PL/SQL program units.
- If you change an object in an object library, all forms that contain the subclassed object reflect the change.

Oracle Internal & OAI Use Only

## Benefits of the Object Library

- Simplifies the sharing and reuse of objects
- Provides control and enforcement of standards
- Promotes increased network performance
- Eliminates the need to maintain multiple referenced forms



ORACLE®

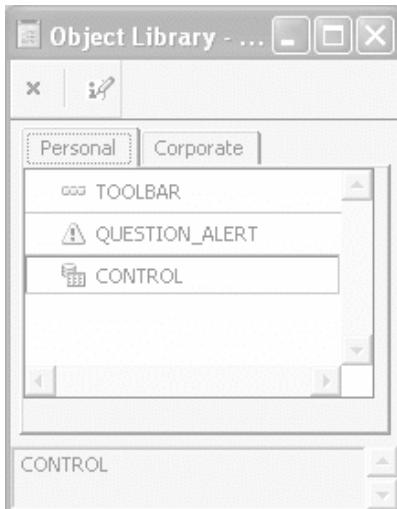
Copyright © 2006, Oracle. All rights reserved.

### Benefits of the Object Library

There are several advantages to using object libraries to develop applications:

- Simplifies the sharing and reuse of objects
- Provides control and enforcement of standards
- Increases the efficiency of message diffing by promoting similarity of objects, thus increasing the performance of the application
- Eliminates the need to maintain multiple referenced forms

# Working with Object Libraries



## Object libraries:

- Appear in the Navigator if they are open
- Are used with a simple tabbed interface
- Are populated by dragging Form objects to tab page
- Are saved to .olb files

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Working with Object Libraries

Object libraries appear in the Navigator if they are open. You can create, open, and close object libraries like other modules. Forms Builder automatically opens all object libraries that were open when you last closed Forms Builder.

It is easy to use object libraries with a simple tabbed interface. Using the Edit menu, you can add or remove tab pages that help you to create your own groups. You can save object libraries to a file system as .olb files.

**Note:** You cannot modify objects inside the object library itself. To make changes, drag the object into a form, change it, and drag it back to the object library.

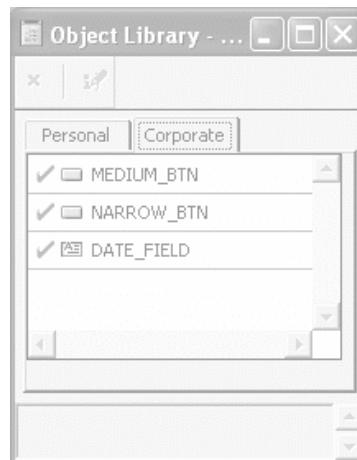
### How to Populate an Object Library

1. Select Tools > Object Library to display the object library.
2. Drag objects from the Object Navigator into the object library.
3. You can edit the descriptive comment by clicking the Edit button in the object library interface.

# What Is a SmartClass?

- **A SmartClass:**
  - Is an object in an object library that is frequently used as a class
  - Can be applied easily and rapidly to existing objects
  - Is the preferred method to promote similarity among objects for performance
- You can have many SmartClasses of a given object type.
- You can define SmartClasses in multiple object libraries.

To create:  
Edit > Smartclass



Check mark  
indicates  
a SmartClass.

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## What Are SmartClasses?

A SmartClass is a special member of an object library. It can be used to easily subclass existing objects in a form using the SmartClass option from the shortcut menu. To use object library members, which are not SmartClasses for existing objects, you have to use the Subclass Information dialog box available in the Property Palette of the form object that you are modifying.

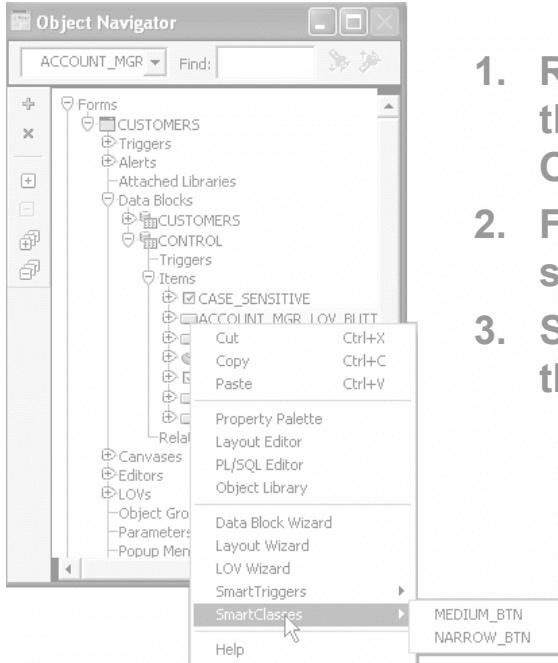
### Marking Objects as SmartClasses

If you frequently use certain objects as standards, such as standard buttons, date items, and alerts, then you can mark them as SmartClasses by selecting each object in the object library and choosing Edit > SmartClass.

You can mark many different objects, spread across multiple object libraries, as SmartClasses. Other than accepting default values for all object properties, using Smart Classes is the preferred method to promote similarities between objects for efficiency of message diffing, resulting in better performance of applications.

You can also have many SmartClasses of a given object type (for example, Wide\_button, Narrow\_button, and Small\_iconic\_button).

# Working with SmartClasses



1. Right-click an object in the Layout Editor or the Object Navigator.
2. From the pop-up menu, select SmartClasses.
3. Select a class from the list.

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Working with SmartClasses

To apply a SmartClass to a Forms object, perform the following steps:

1. Right-click an object in the Layout Editor or Object Navigator.
2. From the pop-up menu, select SmartClasses. The SmartClasses pop-up menu lists all the SmartClasses from all open object libraries that have the same type as the object, and, for items, also have the same item type (for example, push button, text item).
3. Select a class for the object; it then becomes the parent class of the object. You can see its details in the Subclass Information dialog box in the object's Property Palette, just like any other subclassed object.

This mechanism makes it very easy to apply classes to existing objects.

# Reusing PL/SQL

- **Triggers:**
  - Copy and paste text
  - Copy and paste within a module
  - Copy to or subclass from another module
  - Move to an object library
- **PL/SQL program units:**
  - Copy and paste text
  - Copy and paste within a module
  - Copy to or subclass in another module
  - Create a library module
  - Move to an object library

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Reusing PL/SQL

### PL/SQL in Triggers

You can reuse PL/SQL in your triggers by:

- Copying and pasting, using the Edit menu
- Copying to another area of the current form module, using Copy and Paste on the shortcut menu
- Copying to or subclassing from another form module, using the drag-and-drop feature in the Object Navigator
- Dragging the trigger to an object library

### PL/SQL Program Units

Although triggers are the primary way to add programmatic control to a Forms Builder application, using PL/SQL program units to supplement triggers, you can reuse code without having to retype it.

With Forms Builder, you can create PL/SQL program units to hold commonly used code. These PL/SQL program units can use parameters, which decrease the need to hard-code object names within the procedure body.

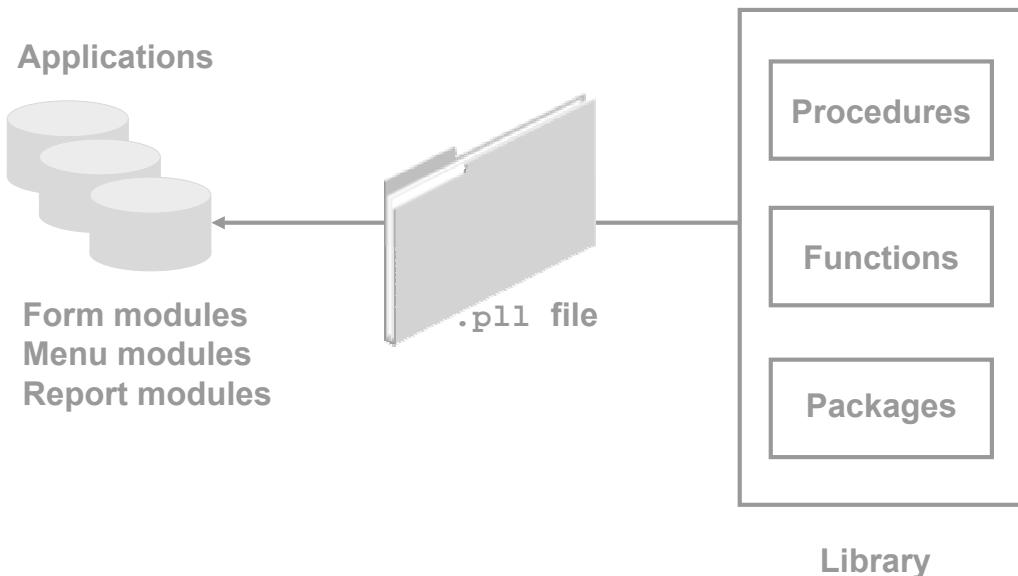
## **Reusing PL/SQL (continued)**

You can reuse PL/SQL program units by:

- Copying and pasting, using the Edit menu
- Copying or subclassing to another form module, using the drag-and-drop feature in the Object Navigator
- Creating a library module
- Dragging the program unit to an object library

Oracle Internal & OAI Use Only

# What Are PL/SQL Libraries?



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## What Are PL/SQL Libraries?

A *library* is a collection of PL/SQL program units that can include procedures, functions, and packages. A single library can contain many program units that can be shared among the Oracle Forms Developer modules and applications that need to use them.

A library:

- Is produced as a separate module and stored in a .p11 file
- Provides a convenient means of storing client-side code and sharing it among applications
- Provides a way for many form, menu, or report modules to use a single copy of program units
- Supports dynamic loading of program units

## Scoping of Objects

Because libraries are compiled independently of the Forms modules that use them, bind variables in forms, menus, reports, and displays are outside the scope of the library. This means that you cannot directly refer to variables that are local to another module because the compiler does not know about them when you compile the library program units.

# Writing Code for Libraries

- A library is a separate module, holding procedures, functions, and packages.
- Direct references to bind variables are not allowed.
- Use subprogram parameters for passing bind variables.
- Use functions, where appropriate, to return values.

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Writing Code for PL/SQL Libraries

There are two ways to avoid direct references to bind variables:

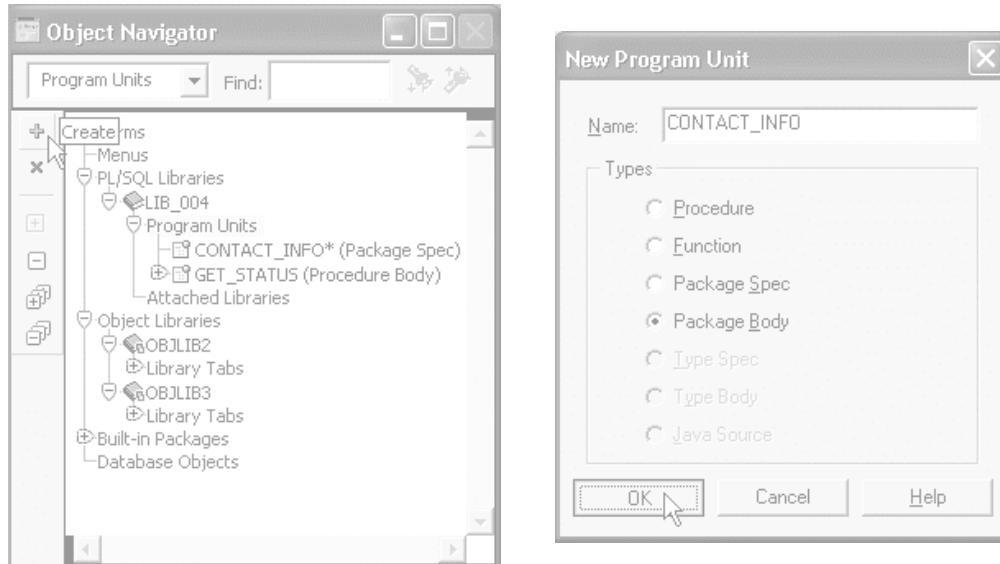
- You can refer to global variables and system variables in forms indirectly as quoted strings by using certain built-in subprograms.
- Write program units with IN and IN OUT parameters that are designed to accept references to bind variables. You can then pass the names of bind variables as parameters when calling the library program units from your Forms applications.

### Example

Consider the second method listed above in the following library subprogram:

```
FUNCTION locate_emp(bind_value IN NUMBER) RETURN VARCHAR2 IS
    v_ename VARCHAR2(15);
BEGIN
    SELECT last_name INTO v_ename FROM employees
    WHERE employee_id = bind_value;
    RETURN(v_ename);
END;
```

# Creating Library Program Units



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Creating Library Program Units

### Creating a Library

You must first create libraries in the builder before you add program units. To do this, you can do one of the following:

- Select File > New > PL/SQL Library from the menu; an entry for the new library then appears in the Object Navigator.
- Select the PL/SQL Libraries node in the Object Navigator, and then click Create on the tool bar.

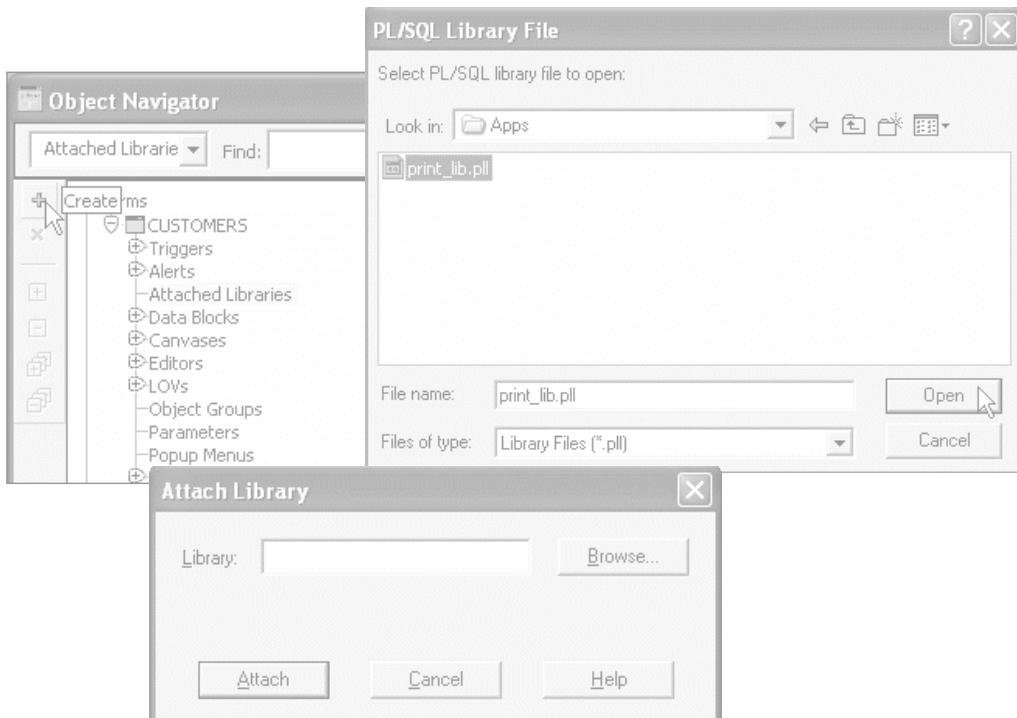
### Adding Program Units

There is a Program Units node within the library's hierarchy in the Object Navigator. From this node, you can create procedures, functions, package bodies, and specifications in the same way as in other modules.

### How to Save the Library

1. With the context set on the library, click Save.
2. Enter the name by which the library is to be saved.

# Attach Library Dialog Box



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Attaching Libraries

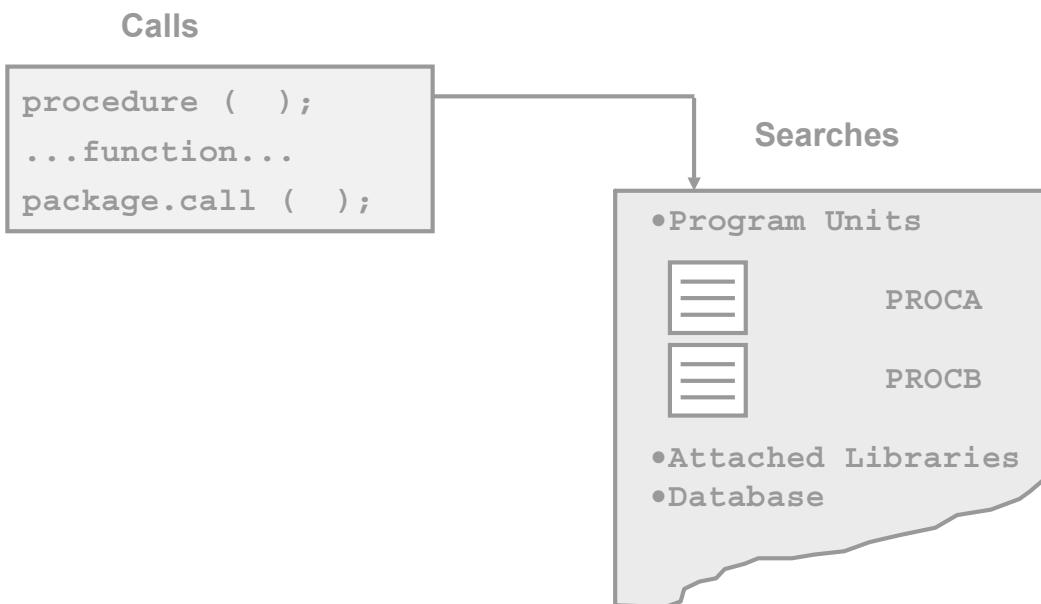
Before you can refer to a library program unit from a form, menu, report, or graphics, you must attach the library to the modules.

To attach a library to a module, perform the following steps:

1. Open the module that needs to have the library attached. This may be a form, menu, or another library module.
2. Expand the module and select the Attached Libraries node in the Object Navigator.
3. Click Create to display the Attach Library dialog box.
4. In the Attach Library dialog box, specify the name of the library.
5. Click Attach.
6. You are asked if you want to remove the path to the library. You should click Yes and include the path to the library in the FORMS\_PATH design-time and run-time environment variables to avoid a hard-coded path, thus facilitating deployment.
7. Save the module to which you have attached the library. This permanently records the library attachment in the definition of this module.

To detach a library later, in the Object Navigator, delete the library entry from the list of Attached Libraries for that module. That module will then no longer be able to reference the library program units, either in the Builder or at run time.

# Calls and Searches



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Referencing Attached Library Program Units

You refer to library program units in the same way as those that are defined locally, or stored in the database. Remember that objects declared in a package must be referenced with the package name as a prefix, whether or not they are part of a library.

Program units are searched for first in the calling module, and then in the libraries that are attached to the calling module.

### Example

Assume that the program units `report_totals`, `how_many_people`, and `pack5.del_emps` are defined in an attached library:

```
report_totals(:sub1);      --library procedure
v_sum := how_many_people; --library function
pack5.del_emps;           --library package procedure
```

## **Referencing Attached Library Program Units (continued)**

### **When Several Libraries Are Attached**

You can attach several libraries to the same Oracle Forms Developer module. References are resolved by searching through libraries in the order in which they occur in the attachment list.

If two program units of the same name and type occur in different libraries in the attachment list, then the one in the “higher” library will be executed because it is located first.

### **Creating .PLX Files**

The library .PLX file is a platform-specific executable that contains no source.

When you are ready to deploy your application, you will probably want to generate a version of your library that contains only the compiled p-code, without any source. You can generate a .PLX file from Forms Builder.

# Summary

In this lesson, you should have learned that:

- You can reuse objects or code in the following ways:
  - Property classes
  - Object groups
  - Copying and subclassing
  - Object libraries and SmartClasses
- To inherit properties from a property class, you set an item's Subclass Information property
- You can create an object group in one module to make it easy to reuse related objects in other modules

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Summary

Forms provides a variety of methods for reusing objects and code. This lesson described how to use these methods.

Reasons to share objects and code:

- Increased productivity
- Increased modularity
- Decreased maintenance
- Maintaining standards
- Increased performance

Methods of sharing objects and code:

- Property classes
- Object groups
- Copying
- Subclassing
- Creating a library module
- Using object libraries and SmartClasses

## Summary

In this lesson, you should have learned that:

- Inheritance symbols in the Property Palette show whether the value is changed, inherited, overridden, or the default
- You can drag objects from an object library or mark them as SmartClasses for even easier reuse
- You can reuse PL/SQL code by:
  - Copying and pasting in the PL/SQL Editor
  - Copying or subclassing
  - Defining program units to call the same code at multiple places within a module
  - Creating PL/SQL library to call the same code from multiple forms

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

Oracle Internal & OAI Use Only

## Practice 23: Overview

This practice covers the following topics:

- Creating an object group and using this object group in a new form module
- Using property classes
- Creating an object library and using this object library in a new form module
- Modifying an object in the object library and observing the effect on subclassed objects
- Setting and using SmartClasses
- Creating a PL/SQL program unit to be called from multiple triggers

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Practice 23: Overview

In this practice, you use an object group and an object library to copy Forms Builder objects from one form to another. You will also create a property class and use it to set multiple properties for several objects. You set SmartClasses in the object library and use these classes in the form module.

- Creating an object group and using this object group in a new form module
- Using property classes
- Creating an object library and using this object library in a new form module
- Modifying an object in the object library to see how the modification affects subclassed objects
- Setting and using SmartClasses
- Creating a PL/SQL program unit to be called from multiple triggers

**Note:** For solutions to this practice, see Practice 23 in Appendix A, “Practice Solutions.”

## Practice 23

1. In the ORDGXX form, create an object group, called Stock\_Objects, consisting of the INVENTORIES block, CV\_INVENTORY canvas, and WIN\_INVENTORY window.
2. Save the form.
3. Create a new form module and copy the Stock\_Objects object group into it.
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

5. Apply ClassA to CV\_INVENTORY and the Quantity\_on\_Hand item.
6. Save the form module as STOCKXX.fmb, compile, and run the form and note the error.
7. Correct the error. Save, compile, and run the form again.
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.

9. Create a new form, and create a data block based on the DEPARTMENTS table, including all columns except DN. Use the Form layout style.

Drag the Toolbar canvas, CONTROL block, and Question\_Alert from the object library to 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.

### **Practice 23 (continued)**

10. Save this form as DEPTGXX, compile, and run the form to test it.
11. Try to delete items on the Null canvas. What happens and why?
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.
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.
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.

Oracle Internal & OAI Use Only

# Using WebUtil to Interact with the Client

24

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

Oracle Internal & OAI Use Only

# Objectives

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

- **Describe the benefits of the WebUtil utility**
- **Integrate WebUtil into a form**
- **Use WebUtil to interact with a client machine**

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Introduction

### Overview

Forms built-in subprograms are called on the middle-tier application server. However, there are times when you want to be able to interact with the client machine. You can do so with JavaBeans and PJC's, but there is a utility called WebUtil that includes much prewritten functionality for client interaction. This lesson shows you how to use WebUtil to interface with the client machine.

# WebUtil: Overview

**WebUtil is a utility that:**

- **Enables you to provide client-side functionality on Win32 clients**



- **Consists of:**
  - Java classes
  - Forms objects
  - PL/SQL library

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## WebUtil: Overview

Forms built-ins typically are executed on the application server machine. Some Forms built-ins interact with the machine to create or read a file, read an image file, or execute operating system commands. Although, in some cases, it is desirable to execute such built-ins on the application server machine, there is often a need to perform such functionality on the client. To do this, you can use a JavaBean or PJC, but that requires that you either write or locate prewritten components and integrate each into Forms applications.

WebUtil is a utility that enables you to easily provide client-side functionality. It consists of a set of Java classes, Forms objects, and a PL/SQL API that enables you to execute the many Java functions of WebUtil without knowing Java.

For further information about WebUtil, see the WebUtil page on OTN at:  
<http://otn.oracle.com/products/forms/htdocs/webutil/webutil.htm>

**Note:** The client machine must be a Windows 32-bit platform, although the middle-tier server on which WebUtil is installed can be any platform on which Forms Services is supported.

# Benefits of the WebUtil Utility

## Why use WebUtil?

- **Developer has to code only in PL/SQL (no Java knowledge required).**
- **WebUtil is a free download (part of Forms 10g in a patch set).**
- **It is easy to integrate into a Forms application.**
- **It is extensible.**
- **WebUtil provides:**
  - Client/server parity APIs
  - Client/server added value functions
  - Public functions
  - Utility functions
  - Internal functions

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Benefits of the WebUtil Utility

### Why Use WebUtil?

Any Forms developer can use WebUtil to carry out complex tasks on client browser machines by simply coding PL/SQL.

It is very easy to integrate WebUtil into your Forms applications using its object group and PL/SQL library, and you can easily extend it by adding your own custom functionality while leveraging its basic structure.

You can use WebUtil to perform a multitude of tasks, enabling you to:

- Read and write text files on the client machine
- Transfer files between the client, application server, and database
- Read client-side variables
- Manipulate client-side files
- Integrate with C code on the client
- Obtain information about the client
- Use a file selection dialog box on the client
- Run operating system commands on the client machine and call back into Forms
- Integrate with the browser, such as displaying messages to the browser window
- Perform OLE automation, such as using Word and Excel, on the client
- Read and write client-side images

## Benefits of the WebUtil Utility (continued)

### What Functionality Is Available with WebUtil?

There is a wealth of functionality available in the utility, including the following:

- Client/server parity APIs that enable you to retrieve a file name from the client, read or write an image to or from the client, get information about the client machine, or perform HOST and TEXT\_IO commands and OLE automation on the client (Without the WebUtil functions, these built-ins execute on the application server machine.)
- Client/server added value functions (ported from d2kwutil, a client/server package):

CREATE\_REGISTRY\_KEY and DELETE\_REGISTRY\_KEY  
GET\_COMPUTER\_NAME  
GET\_NET\_CONNECTION  
GET\_TEMP\_DIRECTORY, GET\_WINDOWS\_DIRECTORY, and  
GET\_WORKING\_DIRECTORY  
GET\_WINDOWS\_USERNAME  
READ\_INI\_FILE and WRITE\_INI\_FILE  
READ\_REGISTRY, WRITE\_REGISTRY, and WRITE\_REGISTRYEX

**Note:** Some of these may duplicate other WebUtil functions, but are provided as an easy way to migrate code that uses d2kwutil.

- Public functions: The core of WebUtil is a set of packages, each of which provides an API to implement certain functionality.
  - WebUtil\_ClientInfo package: Contains the following functions to obtain information about the client machine:

Function	Returns:
GET_DATE_TIME	Date and time on client machine
GET_FILE_SEPARATOR	Character used on client as file separator (“\” on Windows)
GET_HOST_NAME	Name of client machine
GET_IP_ADDRESS	IP address of client (string)
GET_JAVA_VERSION	JVM version that is running the Forms applet
GET_LANGUAGE	Language code of the client machine, such as de for German
GET_OPERATING_SYSTEM	Name of OS running the browser

## Benefits of the WebUtil Utility (continued)

Function	Returns:
GET_PATH_SEPARATOR	Character used on client to separate directory locations on paths (“;” on Windows)
GET_SYSTEM_PROPERTY	Any Java system property
GET_TIME_ZONE	Time zone of client machine
GET_USER_NAME	Name of user logged in to the client

- WebUtil\_C\_API package: Contains the following functions to call into C libraries on the client:

Function	Purpose:
IsSupported	Returns True if the client is a valid platform
RegisterFunction	Returns a handle to a specified C library function
DeregisterFunction	Deregisters function and frees client resources
Create_Parameter_List	Creates a parameter list to pass to C function
Destroy_Parameter_List	Deletes a parameter list and frees its resources
Add_Parameter	Adds a parameter to the parameter list
Get_Parameter_Number Get_Parameter_Ptr Get_Parameter_String	Typed functions to return parameter values
Rebind_Parameter	Changes parameter values of the existing parameter list to reuse it
Invoke_*	Typed functions to execute a registered C function
Parameter_List_Count	Returns the number of parameter lists that have been created
Function_Count	Returns the number of functions that have been registered
Id_Null	Checks to see whether the various types used in the package are null

## Benefits of the WebUtil Utility (continued)

- WebUtil\_File package: Contains a new type called FILE\_LIST, which is a PL/SQL table used to pass back multiple file names; also contains the following APIs to manipulate files and directories on the client and to display file selection dialog boxes:

Function	Purpose
COPY_FILE RENAME_FILE DELETE_FILE	Copy, rename, or delete a file and return a Boolean value to indicate success
CREATE_DIRECTORY	Creates the named directory if it does not exist; returns a Boolean value to indicate success
DIRECTORY_ROOT_LIST	Returns a FILE_LIST containing the directory roots on the client system (the drives on Windows)
DIRECTORY_FILTERED_LIST	Returns a list of files in a directory that you filter using wildcard characters (* and ?)
FILE_EXISTS	Returns a Boolean value indicating the existence of the specified file on the client
FILE_IS_DIRECTORY	Returns a Boolean value indicating whether the specified file is a directory
FILE_IS_HIDDEN	Returns a Boolean value indicating whether the specified file has its hidden attribute set
FILE_IS_READABLE FILE_IS_WRITABLE	Returns a Boolean value indicating whether the file can be read from or written to
DIRECTORY_SELECTION_DIALOG	Displays a directory selection dialog box and returns the selected directory
FILE_SELECTION_DIALOG	Enables definition of File Save or File Open dialog box with configurable file filter and returns the selected file
FILE_MULTI_SELECTION_DIALOG	Enables definition of File Save or File Open dialog box with configurable file filter and returns the selected files in a FILE_LIST
GET_FILE_SEPARATOR	Returns character used on the client machine as a file separator ("\ on Windows)
GET_PATH_SEPARATOR	Returns character used on client machine to separate directory locations on paths ("," on Windows)

## Benefits of the WebUtil Utility (continued)

- `WebUtil_File_Transfer` package: The `WebUtil_File_Transfer` package contains APIs to transfer files to and from the client browser machine and to display a progress bar as the transfer occurs. The following APIs are included in the `WebUtil_File_Transfer` package:

Function	Purpose:
<code>URL_TO_CLIENT</code> <code>URL_TO_CLIENT_WITH_PROGRESS</code>	Transfers a file from a URL to the client machine (and displays a progress bar)
<code>CLIENT_TO_DB</code> <code>CLIENT_TO_DB_WITH_PROGRESS</code>	Uploads a file from the client to a database BLOB column (and displays a progress bar)
<code>DB_TO_CLIENT</code> <code>DB_TO_CLIENT_WITH_PROGRESS</code>	Downloads file from a BLOB column in the database to the client machine (and displays a progress bar)
<code>CLIENT_TO_AS</code> <code>CLIENT_TO_AS_WITH_PROGRESS</code>	Uploads a file from the client to the application server (with a progress bar)
<code>AS_TO_CLIENT</code> <code>AS_TO_CLIENT_WITH_PROGRESS</code>	Transfers a file from the application server to the client (with a progress bar)
<code>IN_PROGRESS</code>	Returns True if an asynchronous update is in progress
<code>ASYNCHRONOUS_UPLOAD_SUCCESS</code>	Returns a Boolean value indicating whether an asynchronous upload succeeded
<code>GET_WORK_AREA</code>	Returns a work area directory on the application server that is private to the user
<code>IS_AS_READABLE</code> <code>IS_AS_WRITABLE</code>	Returns True if the rules defined in the WebUtil configuration allow the specified file to be read from or written to

- `WebUtil_Session` package: Provides a way to react to an interruption of the Forms session by defining a URL to which the user is redirected if the session ends or crashes. It contains the following:

Function	Purpose
<code>ENABLE_REDIRECT_ON_TIMEOUT</code>	Enables the time-out monitor and the specification of a redirection URL
<code>DISABLE_REDIRECT_ON_TIMEOUT</code>	Switches off the monitor; if you do not call this function before <code>EXIT_FORM</code> , the redirection occurs even if the exit is normal

## Benefits of the WebUtil Utility (continued)

- **WebUtil\_Host package:** Contains routines to execute commands on the client machine. Includes two types: **PROCESS\_ID** to hold a reference to a client-side process, and **OUTPUT\_ARRAY**, a PL/SQL table which holds the **VARCHAR2** output from a client-side command. The **WebUtil\_Host** package also contains the following functions:

Function	Purpose
HOST	Runs the specified command in a <b>BLOCKING</b> mode on the client and optionally returns the return code
BLOCKING	Runs the specified command in a <b>BLOCKING</b> mode on the client and optionally returns the process ID
NONBLOCKING	Runs the specified command in a <b>NON-BLOCKING</b> mode on the client and optionally returns the process ID
NONBLOCKING_WITH_CALLBACK	Runs the specified command in a <b>NON-BLOCKING</b> mode on the client and executes a supplied trigger when the process is complete
TERMINATE_PROCESS	Kills the specified process on the client
GET_RETURN_CODE	Returns the return code of a specified process as an integer
GET_STANDARD_OUTPUT	Returns an <b>OUTPUT_ARRAY</b> containing output that was sent to standard output by the specified client process
GET_STANDARD_ERROR	Returns an <b>OUTPUT_ARRAY</b> containing output that was sent to standard error by the specified client process
RELEASE_PROCESS	Frees resources of specified client process
GET_CALLBACK_PROCESS	Returns the process ID of the finished process when a callback trigger executes following a command called from <b>NonBlocking_With_Callback</b>
ID_NULL	Tests if a process ID is null
EQUALS	Tests whether two process IDs represent the same process

## Benefits of the WebUtil Utility (continued)

- WebUtil\_Core package: Contains mostly private functions, but you can call the following functions:

Function	Purpose
IsError	Checks whether the last WebUtil call succeeded
ErrorCode	Returns the last WebUtil error code
ErrorText	Returns the text of the last WebUtil error

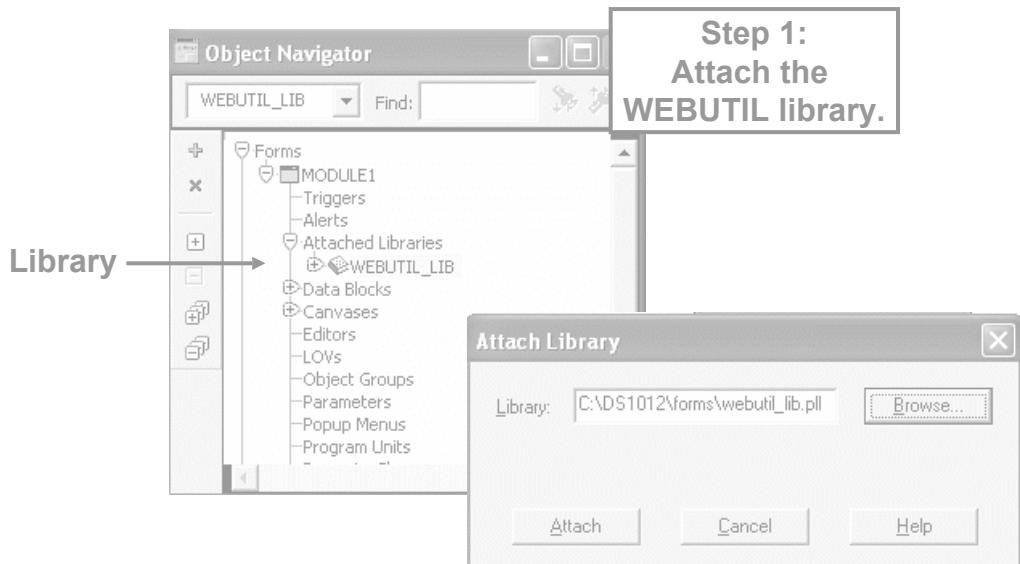
- Utility functions: The following functions are not related to client integration, but they can be useful:

Function	Purpose:
DelimStr	Provides interaction with delimited strings
Show_WebUtil_Information	Calls the hidden WebUtil window to show the version of all WebUtil components
WebUtil_Util	Provides the BoolToStr() function for converting Boolean to text

- Internal APIs that should not be called directly

For more information about WebUtil, including a sample form that showcases its functionality, see OTN at <http://otn.oracle.com/products/forms/htdocs/webutil/webutil.htm>.

# Integrating WebUtil into a Form



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

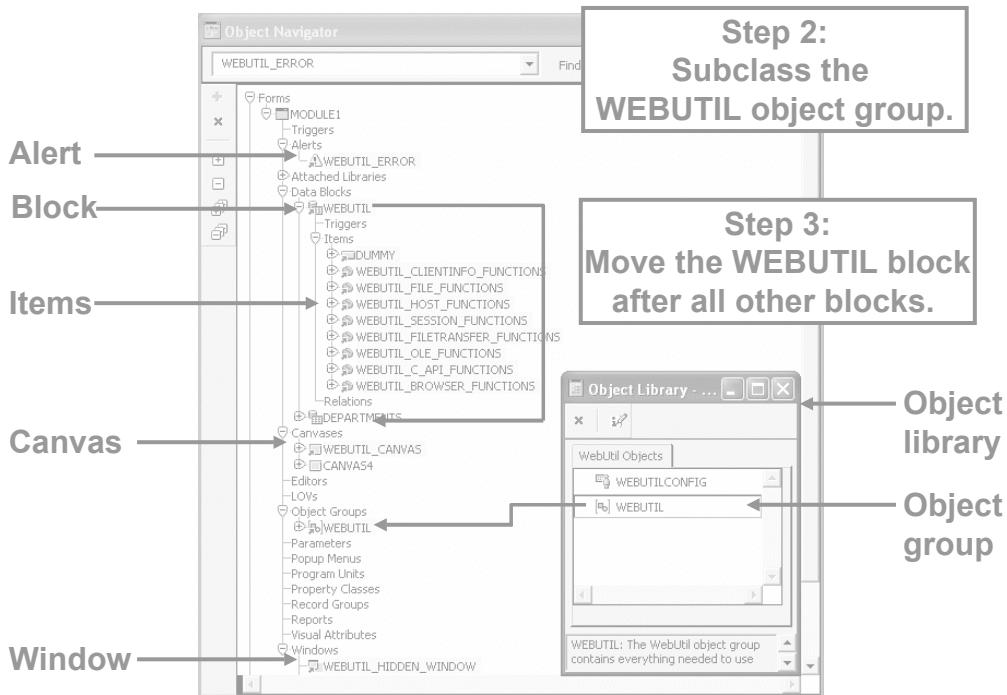
## Integrating WebUtil into a Form

### Step 1: Attaching the WebUtil Library

To use the functions of WebUtil in a Forms application, you must first attach the `webutil.PLL` library to any module that will use the WebUtil PL/SQL API.

**Note:** To avoid problems due to Bug 671456, you should rename the `webutil.PLL` library if there are any other files or subdirectories named `webutil` in the `FORMS_PATH`. Otherwise, you will receive an FRM-40039 error (cannot attach library) at run time.

# Integrating WebUtil into a Form



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Integrating WebUtil into a Form (continued)

### Step 2: Subclassing WebUtil Forms Objects

Part of the WebUtil utility is a set of Forms objects contained in `webutil.olb`. This object library contains an object group called `WebUtil` that you can subclass into your form.

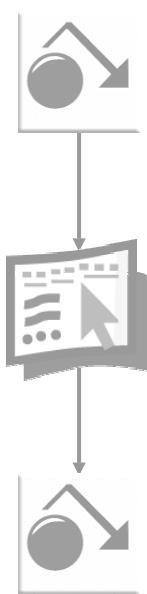
### Step 3: You must ensure that **WEBUTIL** is the last block in the Object Navigator.

If you subclass the `WebUtil` object group into an empty form, you can see that it contains the following objects:

- A generic alert to display WebUtil error messages
- A data block with items, including a button and several bean area items to implement the JavaBeans (the bean area items are hidden because there is no visual component)
- A canvas to contain the items
- A window to display the canvas

**Note:** If you receive an `FRM-92101` error when running the form, be sure that you have followed the above steps exactly. See Note 330497.1 in MetaLink.

# When to Use the WebUtil Functionality



Pre-Form  
When-New-Form-Instance  
When-New-Block-Instance  
(first block)

Form starts  
JavaBeans are instantiated

Any trigger after form starts  
and while form is running



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## When to Use the WebUtil Functionality

After the WebUtil library has been attached to your form, you can start to add calls to the various PL/SQL APIs defined by the utility. However, there is an important restriction in the use of WebUtil functions—WebUtil can communicate with the client only after the Form has instantiated the WebUtil JavaBeans.

This means that you cannot call WebUtil functions before the user interface is rendered, so you should not use the WebUtil functionality in triggers such as Pre-Form, When-New-Form-Instance, and When-New-Block-Instance for the first block in the form. In the case of the When-New-Form-Instance trigger, it is possible, however, to call WebUtil functions after a call to the SYNCHRONIZE built-in has been issued because this ensures that the user interface is rendered.

Further, you cannot call WebUtil functions after the user interface has been destroyed. For example, you should not use a WebUtil call in a Post-Form trigger.

# Interacting with the Client

Forms Built-Ins/Packages	WebUtil Equivalents
HOST	CLIENT_HOST
GET_FILE_NAME	CLIENT_GET_FILE_NAME
READ_IMAGE_FILE	CLIENT_IMAGE.READ
WRITE_IMAGE_FILE	(WRITE)_IMAGE_FILE
OLE2	CLIENT_OLE2
TEXT_IO	CLIENT_TEXT_IO
TOOL_ENV	CLIENT_TOOL_ENV

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Interacting with the Client

As previously mentioned, Forms built-ins work on the application server. For the most common Forms built-ins that you would want to use on the client, rather than on the application server, you can add a prefix to use the WebUtil equivalent.

These client/server parity APIs make it easy to provide similar functionality in applications that were written for client/server deployment by preceding those built-ins with “CLIENT\_” or “CLIENT\_IMAGE”. Although this makes it easy to upgrade such applications, other WebUtil commands may provide similar, but better, functionality. The client/server parity APIs include the following:

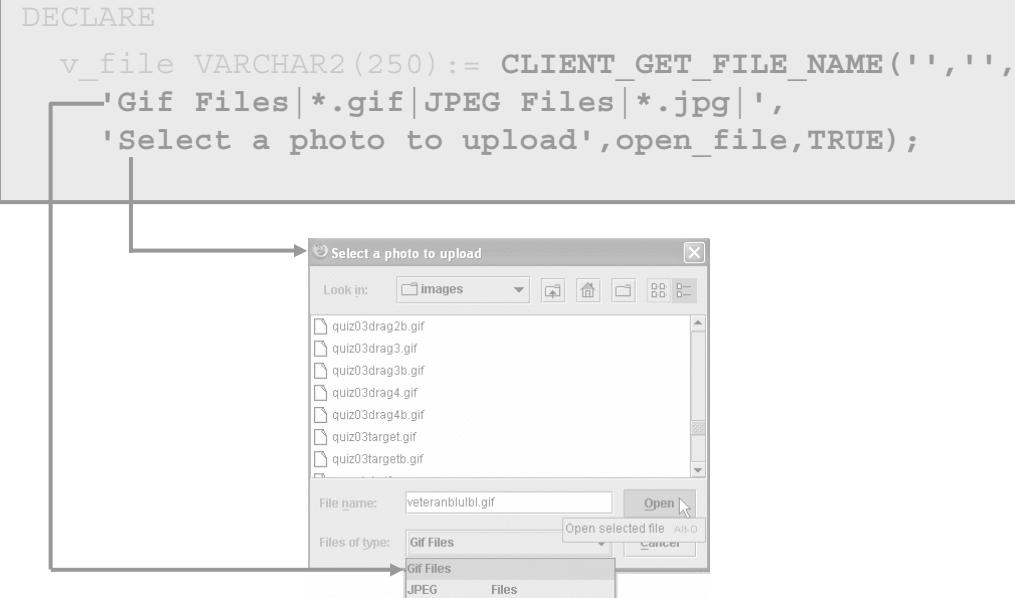
- CLIENT\_HOST
- CLIENT\_GET\_FILE\_NAME

You can use READ\_IMAGE\_FILE on the client by calling the WebUtil equivalent contained in a package: CLIENT\_IMAGE.READ\_IMAGE\_FILE.

In addition, there are certain Forms packages that you can use on the client with WebUtil:

- CLIENT\_OLE2
- CLIENT\_TEXT\_IO
- CLIENT\_TOOL\_ENV

## Example: Opening a File Dialog Box on the Client



Copyright © 2006, Oracle. All rights reserved.

### Example: Opening a File Dialog Box on the Client

To open a file dialog box on the client for selecting a file, you can use:

```
CLIENT_GET_FILE_NAME (DIRECTORY_NAME IN VARCHAR2,
FILE_NAME IN VARCHAR2, FILE_FILTER IN VARCHAR2,
MESSAGE IN VARCHAR2, DIALOG_TYPE IN NUMBER,
SELECT_FILE IN BOOLEAN) RETURN VARCHAR2;
```

The arguments for this WebUtil function are:

- **DIRECTORY\_NAME:** Specifies the name of the directory containing the file you want to open. If DIRECTORY\_NAME is NULL, subsequent invocations of the dialog box may open the last directory visited.
- **FILE\_NAME:** Specifies the name of the file you want to open
- **FILE\_FILTER:** Specifies that only particular files be shown. On Windows, the default is "All Files (\*.\*) | \*.\* |" if NULL.
- **MESSAGE:** Specifies the title of the file upload dialog box
- **DIALOG\_TYPE:** Specifies the intended dialog box to OPEN\_FILE or SAVE\_FILE
- **SELECT\_FILE:** Specifies whether the user is selecting files or directories. The default value is TRUE; if set to FALSE, the user must select a directory. If DIALOG\_TYPE is set to SAVE\_FILE, SELECT\_FILE is internally set to TRUE.

## Example: Reading an Image File into Forms from the Client

The screenshot shows a Windows-style file selection dialog box titled "Select a photo to upload". The "Look in:" dropdown is set to "images". The list of files includes: itmngrbm.jpg, keyboard.jpg, oldtimerblu.jpg, oldtimerbrn.jpg, reports\_logo\_space\_10.jpg, veteranblu.jpg, and veteranbm.jpg. An arrow points from the "veteranblu.jpg" entry in the list to a preview window on the right, which displays a grayscale portrait of a man. Below the dialog is a portion of Oracle PL/SQL code:

```
DECLARE
    v_file VARCHAR2(100);
    'Gif File';
    'Select a photo to upload',open_file,TRUE);
    it_image_id ITEM := FIND_ITEM
        ('employee_photos.photo');
BEGIN
    CLIENT_IMAGE.READ_IMAGE_FILE(v_file,'',it_image_id);
END;
```

ORACLE

Copyright © 2006, Oracle. All rights reserved.

### Example: Reading an Image File into Forms from the Client

You can use the `CLIENT_IMAGE` package to read or write image files. For example, the `CLIENT_IMAGE.READ_IMAGE_FILE` procedure reads an image from the client file system and displays it in the Forms image item:

```
CLIENT_IMAGE.READ_IMAGE_FILE (FILE_NAME VARCHAR2,
FILE_TYPE VARCHAR2, ITEM_ID ITEM or ITEM_NAME VARCHAR2);
```

The arguments for this WebUtil procedure are:

- **FILE\_NAME** Valid file name: The file name designation can include a full path statement appropriate to your operating system.
- **FILE\_TYPE** The valid image file type: BMP, CALS, GIF, JFIF, JPG, PICT, RAS, TIFF, or TPIC. (Note: File type is optional because Oracle Forms will attempt to deduce it from the source image file. To optimize performance, however, you should specify the file type.)
- **ITEM\_ID**: The unique ID that Oracle Forms assigns to the image item when it creates it
- **ITEM\_NAME**: The name you gave the image item when you created it

## Example: Writing Text Files on the Client

```
DECLARE
    v_dir VARCHAR2(250) := 'd:\temp';
    ft_tempfile CLIENT_TEXT_IO.FILE_TYPE;
begin
    ft_tempfile := CLIENT_TEXT_IO.FOPEN(v_dir ||
        '\tempdir.bat','w');
    CLIENT_TEXT_IO.PUT_LINE(ft_tempfile,'dir ' ||
        v_dir || '> '|| v_dir || '\mydir.txt');
    CLIENT_TEXT_IO.PUT_LINE(ft_tempfile,
        'notepad ' || v_dir || '\mydir.txt');
    CLIENT_TEXT_IO.PUT_LINE(ft_tempfile,'del ' ||
        v_dir || '\mydir.*');
    CLIENT_TEXT_IO.FCLOSE(ft_tempfile);
    CLIENT_HOST('cmd /c ' || v_dir || '\tempdir');
END;
```

- 1
- 2
- 3
- 4

ORACLE

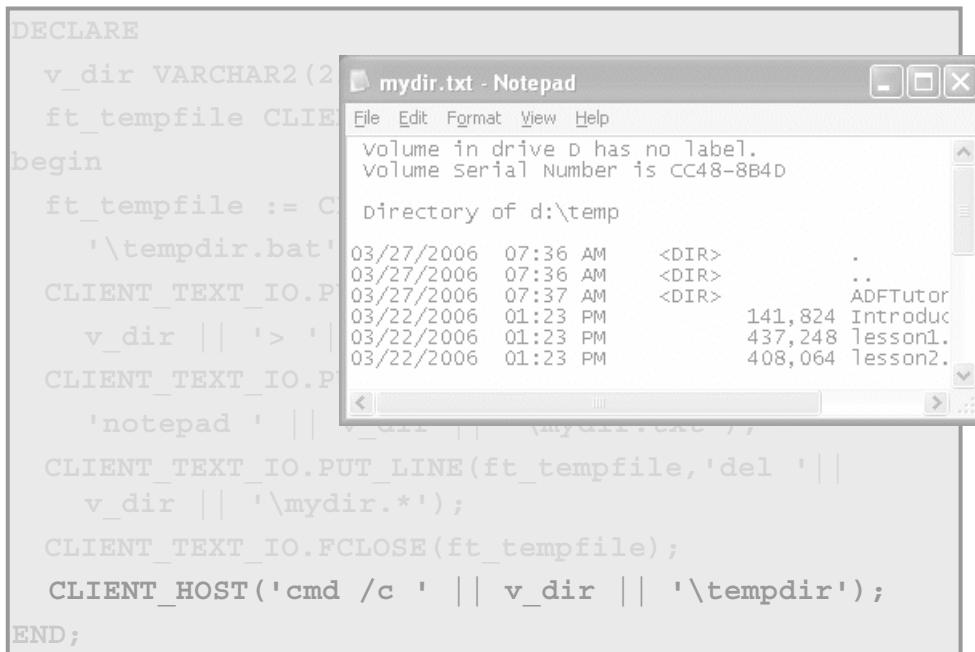
Copyright © 2006, Oracle. All rights reserved.

### Example: Writing Text Files on the Client

With CLIENT\_TEXT\_IO commands, you can create or read text files on the client containing any ASCII text. This example creates a batch file on the client to display a directory listing of the c :\temp directory. The code does the following:

1. Declares a variable to hold a handle to the file
2. Opens a file named tempdir.bat on the client for writing
3. Writes the following lines of text to the file:  
    dir d:\temp> d:\temp\mydir.txt  
    notepad d:\temp\mydir.txt  
    del d:\temp\mydir.\*
4. Closes the file

## Example: Executing Operating System Commands on the Client



The screenshot shows a PL/SQL block on the left and a Notepad window on the right. The PL/SQL block contains code to create a temporary file, write directory information to it, and then delete the file. The Notepad window shows the contents of the temporary file 'mydir.txt' which includes the volume label, serial number, and a list of files in the directory.

```
DECLARE
  v_dir VARCHAR2(255);
  ft_tempfile CLIENT_TEXT_IO.FILE_TYPE;
begin
  ft_tempfile := CLIENT_TEXT_IO.FOPEN('c:\temp\tempdir.bat', 'w');
  v_dir || '> ' || ft_tempfile;
  CLIENT_TEXT_IO.PUT_LINE(ft_tempfile, 'volume in drive D has no label.');
  CLIENT_TEXT_IO.PUT_LINE(ft_tempfile, 'volume Serial Number is CC48-8B4D');
  CLIENT_TEXT_IO.PUT_LINE(ft_tempfile, 'directory of d:\temp');
  CLIENT_TEXT_IO.PUT_LINE(ft_tempfile, '03/27/2006 07:36 AM <DIR> .');
  CLIENT_TEXT_IO.PUT_LINE(ft_tempfile, '03/27/2006 07:36 AM <DIR> ..');
  CLIENT_TEXT_IO.PUT_LINE(ft_tempfile, '03/27/2006 07:37 AM <DIR> ADFTutor');
  CLIENT_TEXT_IO.PUT_LINE(ft_tempfile, '03/22/2006 01:23 PM 141,824 Introduc');
  CLIENT_TEXT_IO.PUT_LINE(ft_tempfile, '03/22/2006 01:23 PM 437,248 lesson1.');
  CLIENT_TEXT_IO.PUT_LINE(ft_tempfile, '03/22/2006 01:23 PM 408,064 lesson2.');
  CLIENT_TEXT_IO.FCLOSE(ft_tempfile);
  CLIENT_HOST('cmd /c ' || v_dir || '\tempdir');
END;
```

ORACLE

Copyright © 2006, Oracle. All rights reserved.

### Example: Executing OS Commands on the Client

You can execute simple HOST commands on the client using the `CLIENT_HOST` command of WebUtil. The example shows running the batch file that was created in the previous example with `CLIENT_TEXT_IO`; `cmd /c` opens a command window and closes it after running the command. You must use `cmd /c` rather than running the command directly or it will not work. You can run any command that you would be able to execute from the Windows Start > Run menu.

Rather than creating the batch file with `CLIENT_TEXT_IO`, alternatively you can execute the commands it contains as follows:

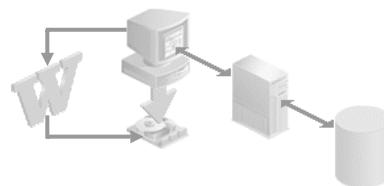
```
CLIENT_HOST('cmd /c dir c:\temp > c:\temp\mydir.txt');
CLIENT_HOST('cmd /c notepad c:\temp\mydir.txt');
CLIENT_HOST('cmd /c del c:\temp\mydir.*');
```

**Note:** You can obtain greater versatility in executing operating system commands by using the `WebUtil_Host` package. This enables you to execute commands synchronously or asynchronously and to call back into Forms from asynchronous commands when execution is complete.

## Example: Performing OLE Automation on the Client

You can use the following for OLE automation:

CLIENT_OLE2.OBJ_TYPE	CLIENT_OLE2.CREATE
CLIENT_OLE2.LIST_TYPE	_ARGLIST
CLIENT_OLE2.CREATE_OBJ	CLIENT_OLE2.ADD_ARG
CLIENT_OLE2.SET	CLIENT_OLE2.INVOKE
PROPERTY	CLIENT_OLE2.DESTROY
CLIENT_OLE2.GET_OBJ	_ARGLIST
PROPERTY	CLIENT_OLE2.RELEASE_OBJ
CLIENT_OLE2.INVOKE_OBJ	



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Example: Performing OLE Automation on the Client

You can use any OLE2 package on the client by preceding it with CLIENT\_. You can see the list of the OLE2 package procedures and functions in the Forms Builder Object Navigator under the Built-in Packages node.

You can see examples of client OLE on OTN at  
<http://otn.oracle.com/products/forms/htdocs/webutil/webutil.htm>. The following example takes data from a form to construct a Word document and save it to the client machine:

```
DECLARE
    app CLIENT_OLE2.OBJ_TYPE;
    docs CLIENT_OLE2.OBJ_TYPE;
    doc CLIENT_OLE2.OBJ_TYPE;
    selection CLIENT_OLE2.OBJ_TYPE;
    args CLIENT_OLE2.LIST_TYPE;
BEGIN
    -- create a new document
    app := CLIENT_OLE2.CREATE_OBJ('Word.Application');
    CLIENT_OLE2.SET_PROPERTY(app,'Visible',1);
```

## Example: Performing OLE Automation on the Client (continued)

```
docs := CLIENT_OLE2.GET_OBJ_PROPERTY(app, 'Documents');
doc := CLIENT_OLE2.INVOKE_OBJ(docs, 'add');

selection := CLIENT_OLE2.GET_OBJ_PROPERTY(app, 'Selection');
-- Skip 10 lines
args := CLIENT_OLE2.CREATE_ARGLIST;
CLIENT_OLE2.ADD_ARG(args,6);
FOR i IN 1..10 LOOP
    CLIENT_OLE2.INVOKE(selection, 'InsertBreak', args);
END LOOP;

-- insert data into new document

CLIENT_OLE2.SET_PROPERTY(selection, 'Text',
'RE: Order# '|| :orders.order_id);
FOR i in 1..2 LOOP
    CLIENT_OLE2.INVOKE(selection, 'EndKey');
    CLIENT_OLE2.INVOKE(selection, 'InsertBreak', args);
END LOOP;

CLIENT_OLE2.SET_PROPERTY(selection, 'Text',
'Dear '|| :customer_name|| ':');
FOR i in 1..2 LOOP
    CLIENT_OLE2.INVOKE(selection, 'EndKey');
    CLIENT_OLE2.INVOKE(selection, 'InsertBreak', args);
END LOOP;

CLIENT_OLE2.SET_PROPERTY(selection, 'Text', 'Thank you for your'|||
'order dated'|||
to_char(:orders.order_date, 'fmMonth DD, YYYY')|||
', in the amount of'|||
to_char(:control.total, '$99,999.99')|||
'. We will process your order immediately and want you to ' || 'know
that we appreciate your business');

FOR i in 1..2 LOOP
    CLIENT_OLE2.INVOKE(selection, 'EndKey');
    CLIENT_OLE2.INVOKE(selection, 'InsertBreak', args);
END LOOP;
```

## Example: Performing OLE Automation on the Client (continued)

```
CLIENT_OLE2.SET_PROPERTY(selection,'Text','Sincerely,');
FOR i in 1..5 LOOP
    CLIENT_OLE2.INVOKE(selection,'EndKey');
    CLIENT_OLE2.INVOKE(selection,'InsertBreak',args);
END LOOP;

IF :orders.sales_rep_id IS NOT NULL THEN
    CLIENT_OLE2.SET_PROPERTY(selection,'Text',
        :orders.sales_rep_name || ', Sales Representative');
    CLIENT_OLE2.INVOKE(selection,'EndKey');
    CLIENT_OLE2.INVOKE(selection,'InsertBreak',args);
END IF;

CLIENT_OLE2.SET_PROPERTY(selection,'Text','Summit Office Supply');

-- save document in temporary directory
CLIENT_OLE2.DESTROY_ARGLIST(args);
args := CLIENT_OLE2.CREATE_ARGLIST;
CLIENT_OLE2.ADD_ARG(args, 'letter_' ||
    :orders.order_id || '.doc');
CLIENT_OLE2.INVOKE(doc, 'SaveAs', args);
CLIENT_OLE2.DESTROY_ARGLIST(args);

-- close example.tmp
args := CLIENT_OLE2.CREATE_ARGLIST;
CLIENT_OLE2.ADD_ARG(args, 0);
CLIENT_OLE2.INVOKE(doc, 'Close', args);
CLIENT_OLE2.DESTROY_ARGLIST(args);
CLIENT_OLE2.RELEASE_OBJ(selection);
CLIENT_OLE2.RELEASE_OBJ(doc);
CLIENT_OLE2.RELEASE_OBJ(docs);

-- exit MSWord
CLIENT_OLE2.INVOKE(app,'Quit');
message('Letter created: letter_' ||
    :orders.order_id || '.doc');
END;
```

## Example: Obtaining Environment Information About the Client



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Example: Obtaining Environment Information About the Client

You can use the `CLIENT_TOOL_ENV.GETVAR` procedure from WebUtil to obtain information about registry variables on the client machine. You can obtain the values of any registry variables in the key `HKEY_LOCAL_MACHINE > SOFTWARE > ORACLE`.

**Note:** You can obtain a greater variety of information about the client with the `WebUtil_ClientInfo` package.

## Summary

**In this lesson, you should have learned that:**

- **WebUtil is a free extensible utility that enables you to interact with the client machine**
- **Although WebUtil uses Java classes, you code in PL/SQL**
- **You integrate WebUtil into a form by attaching its PL/SQL library and using an object group from its object library; then you can use its functions after the form has started and while it is running**
- **With WebUtil, you can do the following on the client machine: open a file dialog box, read and write image or text files, execute operating system commands, perform OLE automation, and obtain information about the client machine**

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Summary

WebUtil, included as part of Developer Suite 10g Patchset 1 and a free download from OTN before that, consists of a set of Java classes and a PL/SQL API. You can extend WebUtil by adding Java classes. The PL/SQL API enables you to do all coding within the form in PL/SQL.

After the middle tier has been configured for WebUtil, in order to use it in a form, you need only add an object group from WebUtil's object library and attach WebUtil's PL/SQL library. You should not use WebUtil functions in triggers that fire as the form is starting up or after its user interface has been destroyed.

WebUtil includes much functionality. Some of the most common commands enable you to:

- Open a file dialog box on the client (`CLIENT_GET_FILE_NAME`)
- Read or write an image file on the client (`CLIENT_IMAGE` package)
- Read or write a text file on the client (`CLIENT_TEXT_IO`)
- Execute operating system commands (`CLIENT_HOST` or the `WebUtil.HOST` package)
- Perform OLE automation on the client (`CLIENT_OLE2`)
- Obtain information about the client machine (`CLIENT_TOOL_ENV`)

## Practice 24: Overview

This practice covers the following topics:

- Integrating WebUtil with a form
- Using WebUtil functions to:
  - Open a file dialog box on the client
  - Read an image file from the client into the form
  - Obtain the value of a client environment variable
  - Create a file on the client
  - Open the file on the client with Notepad

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Practice 24: Overview

In this practice, you integrate WebUtil with a form, and then use WebUtil to perform various functions on the client machine.

**Note:** For solutions to this practice, see Practice 24 in Appendix A, “Practice Solutions.”

## Practice 24

1. In the ORDGXX form, integrate the WebUtil objects and library.
2. Save the form.
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 can import the code from `pr24_3.txt`.
4. Set the Forms Builder run-time preferences to use a WebUtil configuration that has been set up for you, `?config=webutil`, and 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 you exit the form.
5. 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`.
  - 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`.



# Introducing Multiple Form Applications

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

Oracle Internal & OAI Use Only

# Objectives

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

- Call one form from another form module
- Define multiple form functionality
- Share data among open forms



Copyright © 2006, Oracle. All rights reserved.

## Introduction

### Overview

Oracle Forms Developer applications rarely consist of a single form document. This lesson introduces you to the ways in which you can link two or more forms.

# Multiple Form Applications: Overview

- **Behavior:**
  - Flexible navigation between windows
  - Single or multiple database connections
  - Transactions may span forms, if required
  - Commits in order of opening forms, starting with current form
- **Links:**
  - Data is exchanged by global variables, parameter lists, global record groups, or PL/SQL variables in shared libraries
  - Code is shared as required, through libraries and the database

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Multiple Form Applications: Overview

At the beginning of the course, we discussed the ability to design Forms applications where blocks are distributed over more than one form, producing a modular structure. A modular structure indicates the following:

- Component forms are only loaded in memory if they are needed.
- One form can be called from another, providing flexible combinations, as required.

### How Does the Application Behave?

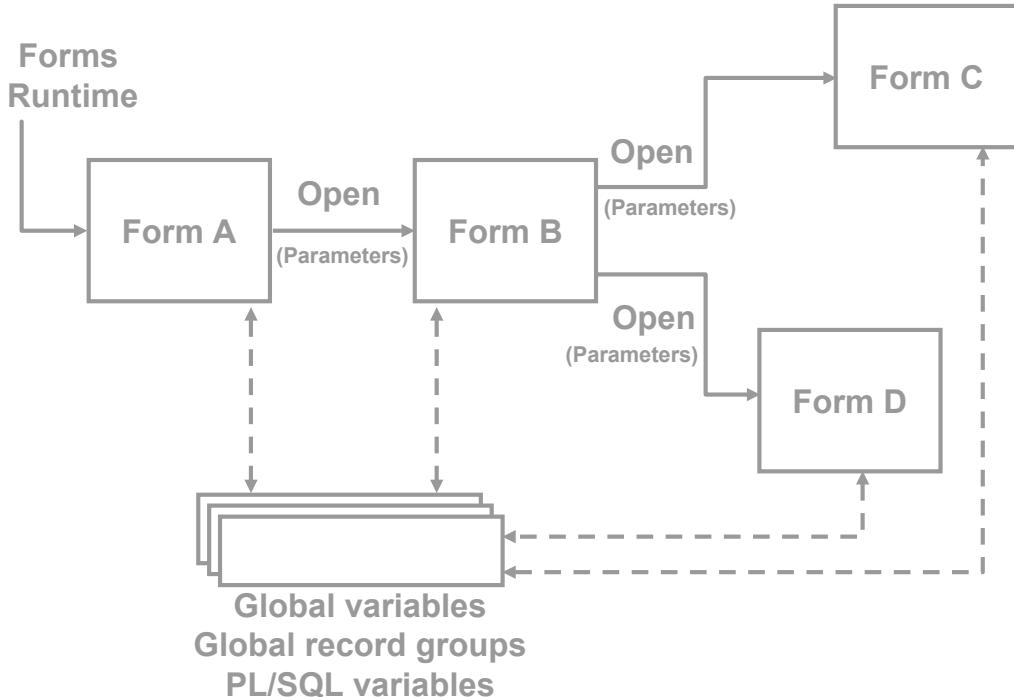
The first form module to run is specified before the Forms session begins in the HTML file that starts the application. Other form modules can be opened in the session by calling built-ins from triggers.

You can design forms to appear in separate windows, so the user can work with several forms concurrently in a session (when forms are invoked by the OPEN\_FORM built-in). Users can then navigate between visible blocks of different forms, much as they can in a single form.

You can design forms for a Forms Runtime session according to the following conditions:

- Forms share the same database session, or open their own separate sessions.

## Multiple Form Session



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Multiple Form Applications: Overview (continued)

- Database transactions are continued across forms, or ended before control is passed to another form. The commit sequence starts from the current form and follows the opening order of forms.
- Forms Builder provides the same menus across the application, or each form provides its own separate menus when it becomes the active form.

### What Links the Forms Together?

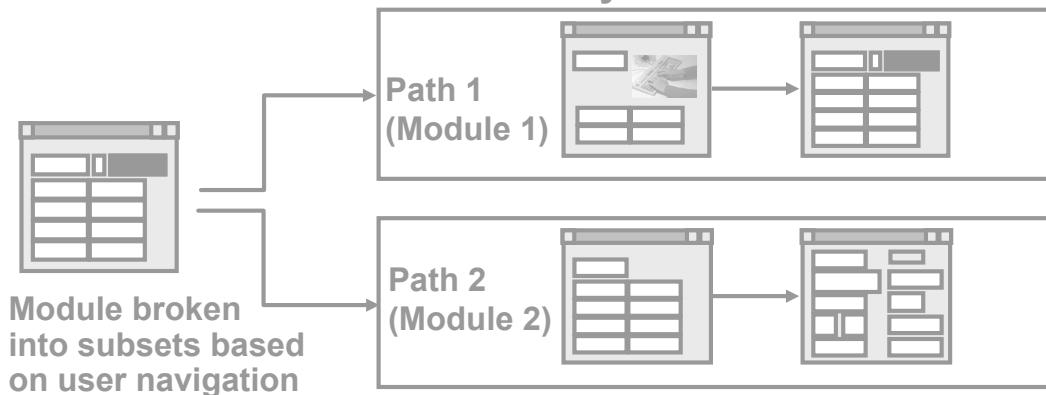
Each form runs within the same Forms Runtime session, and Forms remembers the form that invoked each additional form. This chain of control is used when you exit a form or commit transactions.

There are several methods to exchange data between forms, and code can also be shared.

# Benefits of Multiple Form Applications

Breaking your application into multiple forms offers the following advantages:

- Easier debugging
- Modularity
- Performance and scalability



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

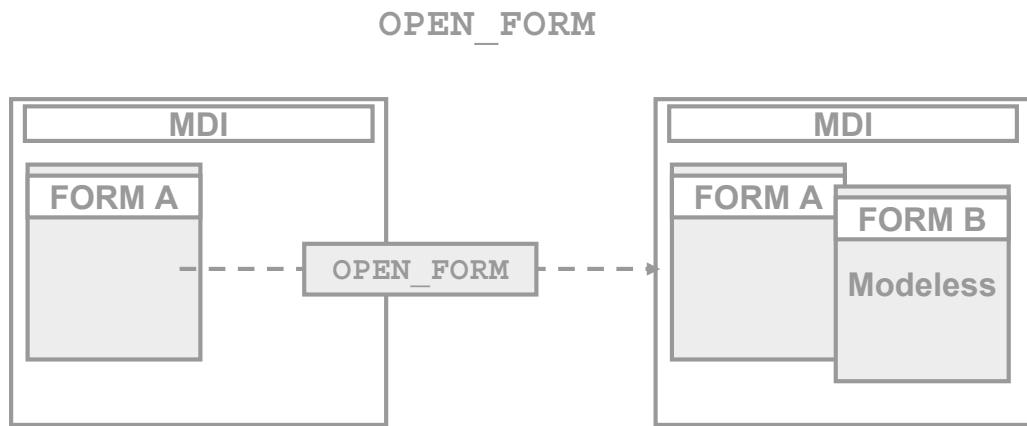
## Multiple Form Applications: Overview (continued)

### Benefits of Multiple Form Applications

Because you can use different windows and canvases, it may seem easier to put the entire application into one large form. However, separating your application into multiple forms offers the following benefits:

- **Debugging:** It is easier to debug a small form; when a single form is working perfectly, you have only to integrate it into your application.
- **Logic modularity:** You break the application into pieces based on the logical functions the module is designed to perform. For example, you would not want the functions of human resources management and accounts receivable combined in a single form, and within each of these major functions are other logical divisions.
- **Network performance and scalability:** Sufficient information must be downloaded to describe the entire form before the form appears on the user's screen. Large forms take longer to download the relevant information, and fewer users can be supported on the given hardware. Break large applications into smaller components based on the likelihood of user navigation, enabling logical areas to be loaded on demand rather than all at once. This approach enables the module to start faster and uses less memory on the application server.

# Starting Another Form Module



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## How to Start Another Form Module

When the first form in a Forms Runtime session has started, it can provide the user with facilities for starting additional forms. This can be performed by one of two methods:

- Calling a built-in procedure from a trigger in the form
- Calling a built-in procedure from a menu item in an attached menu

### Built-in Procedures for Starting Another Form

You can use the `OPEN_FORM` built-in to start another form module from one that is already active. This is a restricted procedure, and cannot be called in the Enter Query mode.

`OPEN_FORM` enables you to start another form in a modeless window, so the user can work in other running forms at the same time.

You can start another form using `OPEN_FORM` without passing control to it immediately, if required. This built-in also gives you the option to begin a separate database session for the new form.

## How to Start Another Form Module (continued)

### Syntax:

```
OPEN_FORM('form_name', activate_mode, session_mode,  
          data_mode, paramlist);
```

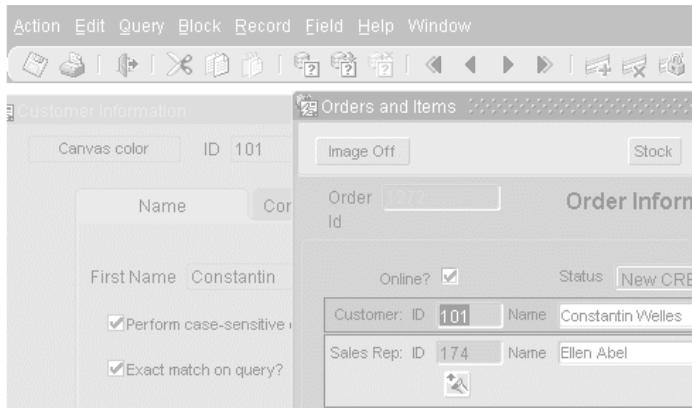
Parameter	Description
Form_Name	File name of the executable module (without the .FMX suffix)
Activate_Mode	Either ACTIVATE (the default), or NO_ACTIVATE
Session_Mode	Either NO_SESSION (the default), or SESSION
Data_Mode	Either NO_SHARE_LIBRARY_DATA (the default) or SHARE_LIBRARY_DATA (Use this parameter to enable Forms to share data among forms that have identical libraries attached.)
Paramlist	Either the name (within quotation marks) or internal ID of a parameter list

There are two other built-ins that can call another form. This course concentrates on OPEN\_FORM, which is considered the primary method to invoke multiple forms, rather than the CALL\_FORM or NEW\_FORM built-ins. For a discussion about these additional built-ins, see the *Oracle9i Forms Developer: Enhance Usability* online course. You can access the online library from the Oracle Education Web page at: <http://www.oracle.com/education>.

# Defining Multiple Form Functionality

## Summit application scenario:

- Run the CUSTOMERS and ORDERS forms in the same session, navigating freely between them.
- You can make changes in the same transaction across forms.
- All forms are visible together.



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Defining Multiple Form Functionality

### Using OPEN\_FORM to Provide Forms in Multiple Windows

You can use OPEN\_FORM to link form modules in an application and enable the user to work in them concurrently. Consider these requirements for the Summit application:

- The CUSTOMERS form must provide an option to start the ORDERS form in the same transaction, and orders for the current customer can be viewed, inserted, updated, and deleted.
- The user can see all open forms at the same time, and freely navigate between them to apply changes.
- Changes in all forms can be saved together.

Using OPEN\_FORM to open both forms in the same session satisfies the requirements. However, having both forms open in the same session may have an undesired effect: If changes are made in the opened form, but not in the calling form, when saving the changes, users may receive an error message indicating that no changes have been made. This error is produced by the calling form; changes in the opened form are saved successfully, but the error message may be confusing to users. To avoid this, you may decide to open the second form in a separate session, but then changes to each form would need to be saved separately.

# Defining Multiple Form Functionality

## Actions:

1. Define windows and positions for each form.
2. Plan shared data, such as global variables and their names.
3. Implement triggers to:
  - Open other forms
  - Initialize shared data from calling forms
  - Use shared data in opened forms

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Defining Multiple Form Functionality (continued)

To provide this kind of functionality, perform the following steps:

1. Create each of the form modules. Plan where the windows of each module will appear in relation to those of other modules.
2. Plan how to share data among forms, such as identifying names for global variables. You need one for each item of data that is to be accessible across all the forms in the application. Note that each form must reference a global variable by the same name.  
**Note:** You can also share data among forms using parameter lists, global record groups, or PL/SQL variables in shared libraries.
3. Plan and implement triggers to:
  - Open another form (You can do this from item interaction triggers, such as When-Button-Pressed, or from When-New-“object”-Instance triggers, or from a Key trigger that fires on a keystroke or equivalent menu selection.)
  - Initialize shared data in calling forms so that values such as unique keys are accessible to other forms when they open. This might need to be done in more than one trigger, if the reference value changes in the calling form.
  - Make use of shared data in opened forms. For example, a Pre-Query trigger can use the contents of the global variable as query criteria.

# Conditional Opening

## Example:

```
IF  ID_NULL(FIND_FORM('ORDERS'))  THEN
    OPEN_FORM('ORDERS');
ELSE
    GO_FORM('ORDERS');
END IF;
```



Copyright © 2006, Oracle. All rights reserved.

## Conditional Opening

The OPEN\_FORM built-in enables you to start up several instances of the same form. To prevent this, the application could perform appropriate tests, such as testing a flag (global variable) set by an opened form at startup, which the opened form could reset on exit. This method may be unreliable, however, because if the form exits with an error, the flag may not be properly reset. A better practice is to use the FIND\_FORM built-in.

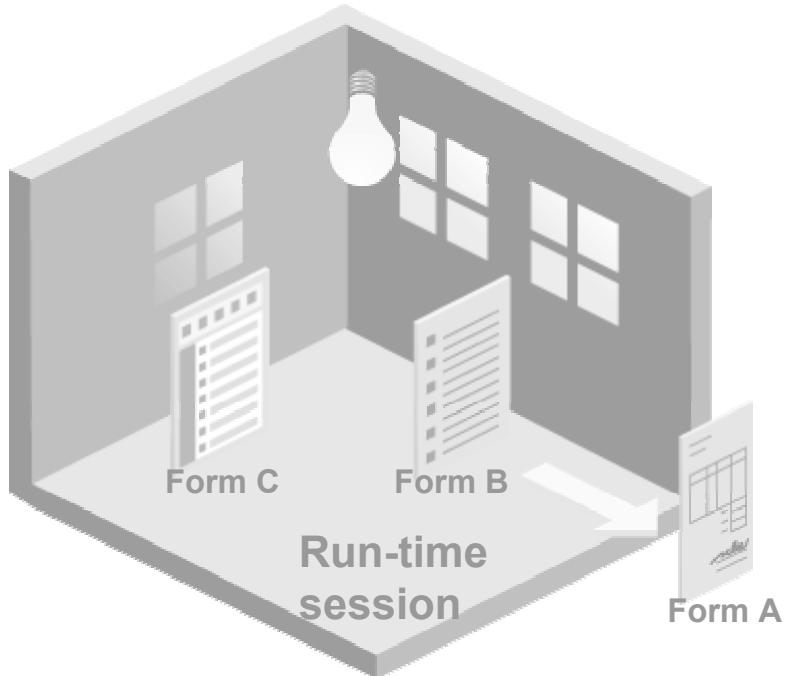
Here is a variation of the When-Button-Pressed trigger on Orders\_Button in the CUSTOMERS form. If the ORDERS form is already running, it simply uses GO\_FORM to pass control to it.

```
IF  ID_NULL(FIND_FORM('orders'))  THEN
    OPEN_FORM('orders');
ELSE
    GO_FORM('orders');
END IF;
```

**Note:** If the name of the form module and its file name are different:

- Use the file name for OPEN\_FORM: OPEN\_FORM('orderswk23');
- Use the form module name for GO\_FORM: GO\_FORM('orders');

# Closing the Session



“Will the last one out please turn off the lights?”

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Closing Forms and Forms Run-Time Sessions

A form may close down and pass control back to its calling form under the following conditions:

- The user presses Exit or selects Exit from the Action menu.
- The `EXIT_FORM` built-in is executed from a trigger.

If the closing form is the only form still running in the Forms run-time session, the session will end as a result. When a multiple form session involves the `OPEN_FORM` built-in, it is possible that the last form to close is not the one that began the session.

## Closing a Form with EXIT\_FORM

- The default functionality is the same as for the Exit key.
- The commit\_mode argument defines action on uncommitted changes.

```
ENTER;

IF    :SYSTEM.FORM_STATUS = 'CHANGED' THEN
    EXIT_FORM( DO_COMMIT );
ELSE
    EXIT_FORM( NO_COMMIT );
END IF;
```



Copyright © 2006, Oracle. All rights reserved.

### Closing a Form with EXIT\_FORM

When a form is closed, Forms checks to see whether there are any uncommitted changes. If there are, the user is prompted with the standard alert:

Do you want to save the changes you have made?

If you are closing a form with EXIT\_FORM, the default functionality is the same as described above. You can, however, make the decision to commit (save) or roll back through the EXIT\_FORM built-in, so the user is not asked. Typically, you might use this built-in from a Key-Exit or When-Button-Pressed trigger.

```
EXIT_FORM(commit_mode);
```

Parameter	Description
Commit_Mode	Defines what to do with uncommitted changes to the current form: <ul style="list-style-type: none"><li>• ASK_COMMIT (the default) gives the decision to the user.</li><li>• DO_COMMIT posts and commits changes across all forms for the current transaction.</li><li>• NO_COMMIT validates and rolls back uncommitted changes in the current form.</li><li>• NO_VALIDATE is the same as NO_COMMIT, but without validation.</li></ul>

# Other Useful Triggers

**Triggers to maintain referential links and synchronize data between forms:**

- **In the parent form:**
  - When-Validate-Item
  - When-New-Record-Instance
- **In opened forms: When-Create-Record**
- **In any form: When-Form-Navigate**



Copyright © 2006, Oracle. All rights reserved.

## Other Useful Triggers When Using OPEN\_FORM

One drawback of designing applications with multiple forms is that you do not have the functionality that is available within a single form to automatically synchronize data such as master and detail records. You must provide your own coding to ensure that related forms remain synchronized.

Because OPEN\_FORM enables the user to navigate among open forms, potentially changing and inserting records, you can use the triggers shown in the slide to help keep referential key values synchronized across forms.

### Example

In the parent form (CUSTOMERS), the following assignment to GLOBAL.CUSTOMERID can be performed in a When-Validate-Item trigger on :CUSTOMERS.Customer\_Id, so that the global variable is kept up-to-date with user changes.

```
:GLOBAL.customerid := :CUSTOMERS.customer_id;
```

The statement can also be issued from a When-New-Record-Instance trigger on the CUSTOMERS block, in case the user navigates to a different customer record.

## Other Useful Triggers When Using `OPEN_FORM` (continued)

### Example

In the opened form (ORDERS), the following code in a When-Create-Record trigger on the ORDERS block ensures that new records use the value of `GLOBAL.CUSTOMERID` as their default.

```
:ORDERS.customer_id := :GLOBAL.customerid;
```

When items are assigned from this trigger, the record status remains NEW, so that the user can leave the record without completing it.

### Example

You may have a query-only form that displays employee IDs and names, with a button to open another form that has all the columns from the EMPLOYEE table so that users can insert new records.

You can use a When-Form-Navigate trigger in the query-only form to reexecute the query, so that when the user navigates back to that form, the newly created records are included in the display.

# Sharing Data Among Modules

You can pass data between modules using:

- Global variables
- Parameter lists
- Global record groups
- PL/SQL package variables in shared libraries



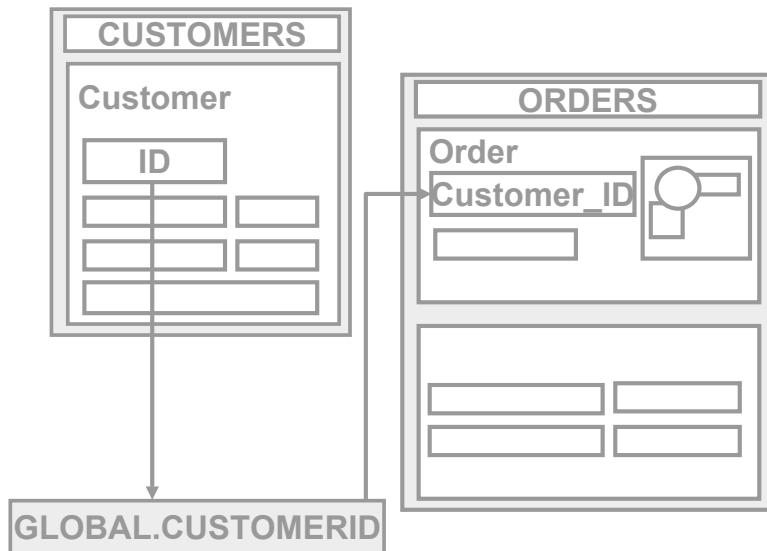
Copyright © 2006, Oracle. All rights reserved.

## Sharing Data Among Modules

Data can be exchanged between forms as follows:

- Through global variables, which span sessions
- Through parameter lists, for passing values between specific forms
- Through record groups created in one form with global scope
- Through PL/SQL variables in shared libraries

# Linking by Global Variables



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Defining Multiple Form Functionality

### Planning Global Variables and Their Names

You need a global variable for each item of data that is used across the application. Reminders:

- Global variables contain character data values, with a maximum of 255 characters.
- Each global variable is known by the same name to each form in the session.
- Global variables can be created by a PL/SQL assignment, or by the `DEFAULT_VALUE` built-in, which has no effect if the variable already exists.
- Attempting to read from a nonexistent global variable causes an error.

The scenario in the slide shows one global variable: `GLOBAL . CUSTOMERID` ensures that orders queried at the startup of the `ORDERS` form apply to the current customer.

# Global Variables: Opening Another Form

## Example:

```
:GLOBAL.customerid := :CUSTOMERS.customer_id;  
OPEN_FORM('ORDERS');
```

## Notes:

- Control passes immediately to the ORDERS form—no statements after OPEN\_FORM are processed.
- If the Activate\_Mode argument is set to NO\_ACTIVATE, you retain control in the current form.
- The transaction continues unless it was explicitly committed before.

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Opening Another Form

- When you default the Activate\_Mode argument in OPEN\_FORM, control is passed immediately to the specified form, and any remaining statements after OPEN\_FORM are not executed.
- If you set Activate\_Mode to NO\_ACTIVATE, control remains in the calling form, although the specified form starts up and the rest of the trigger is processed. Users can then navigate to the other form when they choose.
- If you want to end the current transaction before opening the next form, call the COMMIT\_FORM built-in before OPEN\_FORM. You can check to see if the value of :SYSTEM.FORM\_STATUS='CHANGED' to decide whether a commit is needed. Alternatively, you can just post changes to the database with POST, then open the next form in the same transaction.

## Opening the ORDERS Form from the CUSTOMERS Form

This When-Button-Pressed trigger on :CONTROL.Orders\_Button opens the ORDERS form, and passes control immediately to it. ORDERS will use the same database session and transaction.

```
:GLOBAL.customerid := :CUSTOMERS.customer_id;  
OPEN_FORM('ORDERS');
```

# Global Variables: Restricted Query at Startup

## When-New-Form-Instance



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Performing a Restricted Query on Startup

To display a query automatically in the opened form, with data in context to the calling form, you produce two triggers:

- **When-New-Form-Instance:** This form-level trigger fires when the form is opened (regardless of whether control is passed to this form immediately or not). You can use this trigger to initiate a query by using the EXECUTE\_QUERY built-in procedure. Executing a query fires a Pre-Query trigger if one is defined. The ORDERS form contains the following When-New-Form-Instance trigger:  
EXECUTE\_QUERY;

- **Pre-Query:** This is usually on the master block of the opened form. Because this trigger fires in Enter Query mode, it can populate items with values from global variables, which are then used as query criteria. This restriction applies for every other query performed on the block thereafter.

This Pre-Query trigger is on the ORDERS block of the ORDERS form:

```
:ORDERS.customer_id := :GLOBAL.customerid;
```

# Assigning Global Variables in the Opened Form

- **DEFAULT\_VALUE ensures the existence of globals.**
- **You can use globals to communicate that the form is running.**

Pre-Form example:

```
DEFAULT_VALUE(' ', 'GLOBAL.customerid');
```

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Assigning Global Variables in the Opened Form

If, for some reason, a global variable has not been initialized before it is referenced in a called form, an error is reported:

FRM-40815: Variable GLOBAL.customerid does not exist.

You can provide independence, and ensure that global variables exist by using the DEFAULT\_VALUE built-in when the form is opening.

### Example

This Pre-Form trigger in the ORDERS form assigns a NULL value to GLOBAL.CUSTOMERID if it does not exist when the form starts. Because the Pre-Form trigger fires before record creation, and before all of the When-New-“object”-Instance triggers, it ensures existence of global variables at the earliest point.

```
DEFAULT_VALUE(' ', 'GLOBAL.customerid');
```

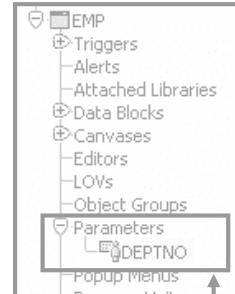
The ORDERS form can now potentially be called without the CUSTOMERS form.

# Linking by Parameters

## Parameters:

- Are form module objects
- Properties:
  - Name
  - Parameter Data Type
  - Maximum Length
  - Parameter Initial Value
- Can optionally receive a new value at run time:

```
http://myhost:8889/forms/frmservlet  
?form=emp.fmx&otherparams=deptno=140
```



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Creating and Passing Parameters

You can create any number of parameters in a form to hold data. Unlike global variables, parameters can be of any data type. However, their use in multiform applications is limited by the fact that they are visible only to the form in which they are defined.

When you run a form, you can pass a value to the parameter as a name-value pair, such as `&otherparams=deptno=140`. After the form receives the parameter value, a trigger can use that value to perform such functionality as restricting a query to records containing that value. You can use the parameter by preceding its name with `:parameter` (for example, `:parameter.deptno`).

In the preceding example, you could construct a Pre-Query trigger to assign the value of `:parameter.deptno` to the `:employees.department_id` form item. When a query is executed on the EMPLOYEES block, only the employees from department 140 would be retrieved.

# Linking by Parameter Lists: The Calling Form

## Example:

```
DECLARE
    pl_id      ParamList;
    pl_name VARCHAR2(10) := 'tempdata';
BEGIN
    pl_id := GET_PARAMETER_LIST(pl_name);
    IF ID NULL(pl_id) THEN
        1 pl_id := CREATE_PARAMETER_LIST(pl_name);
    ELSE
        DELETE_PARAMETER(pl_id,'deptno');
    END IF;
    2 ADD_PARAMETER(pl_id,'deptno',TEXT_PARAMETER,
                   to_char(:departments.department_id));
    3 OPEN_FORM('called_param',ACTIVATE,NO_SESSION,pl_id);
END;
```

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Creating and Passing Parameter Lists: The Calling Form

You can also pass parameters to called forms programmatically by means of a parameter list.

A parameter list is a named programmatic construct that is simply a list of parameter names and character values. The built-in OPEN\_FORM optionally takes as an argument the name or ID of a parameter list. To receive this information, the called form must contain a parameter with the same name as each of those in the parameter list.

To use a parameter list, in the calling form, perform the following steps:

1. Create the parameter list (after checking that it does not already exist).
2. Add a parameter as a name/value pair text parameter.
3. Open the called form and pass the parameter list.

There are several built-ins that enable you to work with parameter lists, including:

```
GET_PARAMETER_LIST
CREATE_PARAMETER_LIST
DESTROY_PARAMETER_LIST
ADD_PARAMETER
DELETE_PARAMETER
```

# Linking by Parameter Lists: The Called Form

## Example: Called form



Create parameter  
in the form

## When-New-Form-Instance Trigger

```
IF :parameter.deptno IS NOT NULL THEN
  SET_BLOCK_PROPERTY('employees',
    DEFAULT WHERE, 'department_id =
      '| :parameter.deptno);
  SET_WINDOW_PROPERTY('window1',
    TITLE, 'Employees in Department
      '| :parameter.deptno);
END IF;
GO_BLOCK('employees');
EXECUTE_QUERY;
```

Use parameter name  
preceded by :parameter

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Creating and Passing Parameter Lists: The Called Form

To use a parameter in the called form, you must first create the parameter in the form. Select the Parameters node in the Object Navigator and click Create. Then change the name of the parameter to be the name that you are passing in the parameter list from the calling form. After you have defined the parameter, you can use it in any of the called form's code by preceding the parameter name with :parameter. You can use the form independently of the calling form if you check to see whether the parameter is NOT NULL before using it or if you set the Parameter Initial Value property of the parameter.

# Linking by Global Record Groups

## 1. Create record group with global scope:

```
DECLARE
    rg_name      VARCHAR2(40) := 'LIST';
    rg_id        RecordGroup;
    Error_Flag   NUMBER;
BEGIN
    rg_id := FIND_GROUP(rg_name);
    IF ID_NULL(rg_id) THEN
        rg_id := CREATE_GROUP_FROM_QUERY('LIST',
            'Select last_name, to_char(employee_id)
            from employees', GLOBAL_SCOPE);
    END IF;
```

## 2. Populate record group:

```
Error_Flag := POPULATE_GROUP(rg_id);
```

## 3. Use record group in any form.

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Sharing Global Record Groups Among Forms

The CREATE\_GROUP\_FROM\_QUERY built-in has a scope argument that defaults to FORM\_SCOPE. However, if you use GLOBAL\_SCOPE, the resulting record group is global, and can be used within all forms in the application. After it is created, a global record group persists for the remainder of the run-time session. For a description of using a record group as a basis for a list of values (LOV), see the lesson titled “Creating LOVs and Editors.” There are many other ways to use record groups that are not covered in this course.

To use a global record group, perform the following steps:

1. Use CREATE\_GROUP\_FROM\_QUERY to create the record group with GLOBAL\_SCOPE.
2. Populate the record group with the POPULATE\_GROUP built-in.
3. Use the record group in any form in the same session.

# Linking by Shared PL/SQL Variables

## Advantages:

- Use less memory than global variables
- Can be of any data type

## To use:

1. Create a PL/SQL library
2. Create a package specification with variables
3. Attach the library to multiple forms
4. Set variable values in calling form
5. OPEN\_FORM with the SHARE\_LIBRARY\_DATA option
6. Use variables in opened form

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

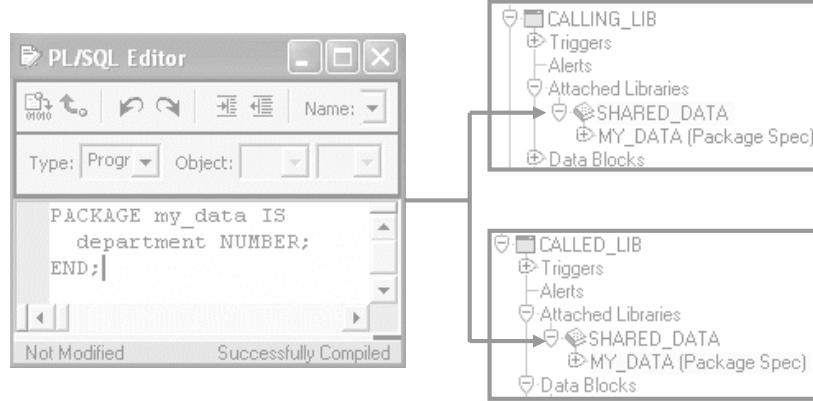
## Linking by Package Variables in Shared PL/SQL Library

Perhaps the simplest and most efficient way to share data among forms is by using packaged variables in PL/SQL libraries. This enables you to use any data type, even user-defined types, to pass information between forms.

You create a library with at least a package specification that contains one or more variable declarations. You then attach that library to the calling and called forms.

When you open the called form with the SHARE\_LIBRARY\_DATA option, the variable value can be set and used by both open forms, making it very easy to share information among multiple forms.

# Linking by Shared PL/SQL Variables



```
OPEN_FORM('called_lib',ACTIVATE,  
NO_SESSION,SHARE_LIBRARY_DATA);
```

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Linking by Package Variables in Shared PL/SQL Library (continued)

For example, with the package shown in the slide in a library that is attached to two forms, in a When-Button-Pressed trigger of the calling form:

```
my_data.department := :departments.department_id;  
OPEN_FORM('called_lib',ACTIVATE,NO_SESSION,SHARE_LIBRARY_DATA);
```

And in the When-New-Form-Instance trigger of the called form:

```
IF my_data.department IS NOT NULL THEN  
    SET_BLOCK_PROPERTY('employees',DEFAULT WHERE,  
    'department_id='||TO_CHAR(my_data.department));  
END IF;  
GO_BLOCK('employees');  
EXECUTE_QUERY;
```

## Summary

In this lesson, you should have learned that:

- **OPEN\_FORM** is the primary method to call one form from another form module
- You define multiple form functionality such as:
  - Whether all forms run in the same session
  - Where the windows appear
  - Whether multiple forms should be open at once
  - Whether users should be able to navigate among open forms
  - How data will be shared among forms

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

### Summary

This lesson is an introductory lesson to multiple form applications. You should have learned how to:

- Open more than one form module in a Forms Runtime session
- Define how multiple forms in an application will function
- Pass information among forms

## Summary

In this lesson, you should have learned that:

- You can share data among open forms with:
  - Global variables, which span sessions
  - Parameter lists, for passing values between specific forms
  - Record groups created in one form with global scope
  - PL/SQL variables in shared libraries



Copyright © 2006, Oracle. All rights reserved.

Oracle Internal & OAI Use Only

## Practice 25: Overview

This practice covers the following topics:

- Using a global variable to link ORDERS and CUSTOMERS forms
- Using built-ins to check whether the ORDERS form is running
- Using global variables to restrict a query in the ORDERS form



Copyright © 2006, Oracle. All rights reserved.

### Practice 25: Overview

In this practice, you produce a multiple form application by linking the CUSTGXX and the ORDGXX form modules.

- Linking ORDERS and CUSTOMERS forms by using a global variable
- Using built-ins to check whether the ORDERS form is running
- Using global variables to restrict a query in the ORDERS form

**Note:** For solutions to this practice, see Practice 25 in Appendix A, “Practice Solutions.”

## Practice 25

1. In the ORDGXX form, create a Pre-Form trigger to ensure that a global variable called Customer\_Id exists.
2. Add a trigger to ensure that queries on the ORDERS block are restricted by the value of GLOBAL.Customer\_Id.
3. Save, compile, and run the form to test that it works as a stand alone.
4. In the CUSTGXX form, create a CONTROL block button called Orders\_Button and set appropriate properties. Place it on the CV\_CUSTOMER canvas below the Customer\_Id item.
5. Define a trigger for CONTROL.Orders\_Button that initializes GLOBAL.Customer\_Id with the current customer's ID, and then opens the ORDGXX form, passing control to it.
6. Save and compile each form. Run the CUSTGXX form and test the button to open the Orders form.
7. Change the window location of the ORDGXX form, if required.
8. Alter the Orders\_Button trigger in CUSTGXX so that it does not open more than one instance of the Orders form, but uses GO\_FORM to pass control to ORDGXX if the form is already running. Use the FIND\_FORM built-in for this purpose.
9. If you navigate to a second customer record and click the Orders button, 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.
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.
11. Add code to the CUSTGXX form so that GLOBAL.Customer\_Id is updated when the current Customer\_Id changes.
12. Save and compile the ORDGXX form. Save, compile, and run the CUSTGXX form to test the functionality.
13. If you have time, you can modify the appearance of the ORDGXX 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



# Practice Solutions

ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

Oracle Internal & OAI Use Only

## Information Needed for Practices

Your instructor will give you information to use in your practices. If you want, you may record it below:

Name	Value	Determined By
Group number		Assigned by instructor (values 1-19)
Lab directory		In standard setup is: e:\labs\lab
connect_string	ora____/oracle@_____	Obtain from instructor
DS_home		In standard setup is: e:\oracle\ds10g
DS_host_name		Right-click My Computer, select Properties, and then click the Network ID tab.
OC4J_port		Get the value of Oracle Developer Suite HTTP port from <DS_home>\ install\portlist.ini .

Oracle Internal & OAI Use Only

## 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 (10.1.2.0.2) initialized.”**

2. Invoke Internet Explorer and enter the following URL:

`http://<machine>:<port>/forms/frmsservlet?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**

4. 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.

**Press [Up] and [Down] (or click Previous Record and Next Record) to browse through the records returned.**

5. 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 on the toolbar).**

**Notice that the status line displays 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 on the toolbar.)**

**Notice that only one record is retrieved.**

6. Execute a restricted query to retrieve the customer whose first name is “Meenakshi.”

**Put the form in Enter-Query mode again. Move to the First Name item and enter the search value Meenakshi. Then execute the query.**

7. Try each of these restricted queries:

- a. Retrieve all cities starting with San.

**Select Query > Enter.**

**Click the Contact Information tab.**

**Enter San% in the City item.**

**Select Query > Execute.**

- b. Retrieve all customers based in the U.S. with a low credit limit.

**Select Query > Enter.**

**Click the Contact Information tab.**

**Enter 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 then 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.**

## Practice 2 Solutions (continued)

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.**

**Select the Online check box.**

**Select Query > Execute. There is only one online order.**

11. Move to the fourth record (Product ID 2322) in the Item block of Order 2355, and then 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 X in upper right of the Stock Levels window. Move to the ORDER block and select Record > Insert, or click Insert Record on the toolbar.**

Note that some items are already populated with default values. Then enter the following:

Item	Value
Online	Unchecked
Status	New Credit Order (poplist)
Sales Rep ID	151 (Enter the ID, or press [Ctrl] + [L]. Then select David Bernstein from the list of values.)

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 on the toolbar.**

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.**

Oracle Internal & OAI Use Only

## 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 the Welcome dialog box is displayed 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.**

**Select 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.

Id	First Name	Last Name	Job Id	Dept Id
EMPLOY	FIRST_NAME	LAST_NAME	JOB_ID	DEPART
EMPLOY	FIRST_NAME	LAST_NAME	JOB_ID	DEPART
EMPLOY	FIRST_NAME	LAST_NAME	JOB_ID	DEPART

### Practice 3 Solutions (continued)

- 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 then 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. Press [Shift] and select 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, and then select the Distribute option in the Vertically column. 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 on the toolbar.**
- 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 clicking 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	forms/frm servlet

**From the Forms Builder menu, choose Edit > Preferences.**

**Click 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 from 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 on the toolbar, 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 box 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, enter `regedit`, and click OK. Expand the registry nodes: HKEY\_LOCAL\_MACHINE > SOFTWARE > ORACLE. Click the ORACLE node, or 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 FORMS\_PATH 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), and then click OK. **No formal solution**

### Practice 3 Solutions (continued)

16. Similarly, set the value for Forms Builder to use for the icon extension, and then close the registry editor.  
**If the UI\_ICON\_EXTENSION key 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, and then click OK.**
17. Close and reopen Forms Builder. Open the Customers form and verify that the flashlight icon is now displayed in the Layout Editor.

**No formal solution**

Oracle Internal & OAI Use Only

## 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 display one record at a time. Set the frame title to Customers. Set prompts and widths as shown in the following table:

Name	Prompt	Width	Height
CUSTOMER_ID	ID	36	14
CUST_FIRST_NAME	First Name	95	14
CUST_LAST_NAME	Last Name	95	14
CUST_ADDRESS_STREET	Address	185	14
CUST_ADDRESS_POSTAL	Zip	50	14
CUST_ADDRESS_CITY	City	140	14
CUST_ADDRESS_STATE	State Province	50	14
CUST_ADDRESS_COUNTRY	Country Id	14	14
PHONE_NUMBERS	Phone	140	14
NLS_LANGUAGE	Nls Language	18	14
NLS_TERRITORY	Nls Territory	140	14
CREDIT_LIMIT	Credit Limit	54	14
CUST_EMAIL	Cust Email	140	14
ACCOUNT_MGR_ID	Account Mgr Id	36	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, and then click Next.

Click Next, and select the “Create the data block, then call the Layout Wizard” option. Click Finish.

In the Layout Wizard, select [New Canvas] and make sure that the Type field is set to Content. Click Next.

Include all items and click Next.

Set values for prompts as shown, and then click Next.

Set Style to Form, and click Next.

Set Frame Title to Customers, and click Finish.

In Object Navigator, rename the canvas 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)

- Save the new module to a file called CUSTGXX, where XX is the group number that your instructor has assigned to you.

**No formal solution**

- 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**

- 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].**

- 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 them.**

**Use the Align 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. (Double-click in the item prompt to edit it—the cursor changes to an I-beam.)**

Customers	
ID	CUSTOM
Last Name	CUST_LAST_NAME
Address	CUST_ADDRESS_STREET_ADDRESS
State Province	CUST_ADDF
Zip	CUST_ADDF
Phone	PHONE_NUMBERS
Credit Limit	CREDIT_LIMI
Account Mgr Id	ACCOU
Nls Language	NLS
	Nls Territory
	NLS_TERRITORY

- Save and compile the form module.

Click Run Form to run the form. Execute a query.

**No formal solution**

- 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.

Double-click Next, and then select the “Create the data block, then call the Layout Wizard” option. Click Finish.

In the Layout Wizard, select a new canvas and make sure that the Type field is set to Content.

Include all items, such as leave prompts, widths, and heights at their default values.

Set Style to Form.

Set Frame Title to Orders, and click Finish.

In the Object Navigator, rename the canvas 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 Navigator 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 Base Table to ORDER\_ITEMS.

Include all columns.

Click Create Relationship.

Select ORDERS block as the master block, and then click OK.

Click Finish.

Use the Layout Wizard to create a layout.

Select Canvas as CV\_ORDER.

## Practice 5 Solutions (continued)

Include all items except ORDER\_ID.

Do not change any prompts, widths, or heights.

Set Style to Tabular.

Set Frame Title to Items.

Set Records Displayed to 6.

Select the Display Scrollbar check box.

Click Finish.

In the Object Navigator, if ORDER\_ITEMS is displayed first, drag the ORDER\_ITEMS block to a position below the ORDERS block.

3. Save the new module to a file called ORDGXX, 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 Navigator 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 Base Table to INVENTORIES.

Include all columns.

Click Finish.

Use the Layout Wizard to create a layout.

Select a New Canvas.

Include all items.

Do not change any prompts, widths, or heights.

Set Style to Tabular.

Set Frame Title to Stock.

Set Records Displayed to 4.

Select the Display Scrollbar check box.

Click Finish.

In the Object Navigator, rename the canvas 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. Select Deferred and deselect Auto Query. Enter the join condition `order_items.product_id = inventories.product_id`, and then 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 then 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 then 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. Execute a query in 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 then save.

No formal solution

## Practice 6 Solutions

### CUSTGXX Form

1. Create a control block in the CUSTGXX 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 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, 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. After 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 now resize the frame, if desired, to encompass all the items.**

4. Save and run the CUSTGXX form.

Test the effects of the properties that you have set.

**Ensure that queried 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 practice).

## Practice 6 Solutions (continued)

### ORDGXX Form

5. Create a CONTROL block in the ORDGXX 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 pt. (If that font is not available on your window manager, use any available font.)

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 can enter this value if you do 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 pt., 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 [Shift]-click, 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. Multiselect 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 **ORDGXX** 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

### CUSTGXX 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 CUSTGXX 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.

The screenshot shows the Oracle Forms Developer 10g interface with the CUSTGXX form open. The title bar reads "Customer Information". The form has a grid layout with several input fields:

Last Name	CUST_LAST_NAME	First Name	CUST_FIRST_NAME		
Address	CUST_ADDRESS_STREET_ADDRESS	City	CUST_ADDRESS_CITY		
State Province	CUST_ADDF	Zip	CUST_ADDF	Country Id	CU
Cust Email	CUST_EMAIL	Phone	PHONE_NUMBERS		
Account Mgr Id	ACCOUNT_M	Credit Limit	CREDIT_LIMI		

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)

### ORDGXX 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 Prompt to Customer Name.  
Set 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 Prompt to Sales Rep Name.  
Set 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.**

- Justify the values of Unit\_Price, Quantity, and Item\_Total to the right.

**For each of the items, set Justification to Right.**

- 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.**

- In the ORDGXX 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 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

The screenshot shows the 'Order Information' form in Oracle Forms Developer. The form is divided into two main sections: 'Order Details' and 'Order Items'.

**Order Details:**

- Order Id: ORDER\_ID
- Order Date: ORDER\_DATE
- Order Mode: ORDER\_MODE
- Customer Id: CUSTOMER\_ID
- Customer Name: CUSTOMER\_NAME
- Sales Rep Id: SALES\_R
- Sales Rep Name: SALES REP NAME

**Order Items:**

Item#	Product Id	Unit Price	Quantity	Item Total
LINE_PRODUCT1		JUNIT_PRICE	QUANTITY	EM_TOTAL ▲
LINE_PRODUCT2		JUNIT_PRICE	QUANTITY	EM_TOTAL
LINE_PRODUCT3		JUNIT_PRICE	QUANTITY	EM_TOTAL
LINE_PRODUCT4		JUNIT_PRICE	QUANTITY	EM_TOTAL

## 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**

Warehouse Id	In Stock
WAR	QUANTITY

15. Save, compile, and run the form to test the changes.

**No formal solution**

## Practice 8 Solutions

1. In the ORDGXX 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 (if you choose to import the query, in the Open dialog box, select All Files(\*.\*)) from the “Files of type” list):

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

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 then 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 250.

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 ORDGXX 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
```

Then 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, click “Look up return item.” Select ORDERS . SALES \_REP \_NAME, and then 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 CUSTGXX 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
```

Then 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 then 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.

## Practice 8 Solutions (continued)

5. In the CUSTGXX 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 by selecting the Editors node in the Object Navigator and clicking Create. 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**

Oracle Internal & OAI Use Only

## Practice 9 Solutions

1. In the ORDGXX form, convert the Order\_Status item into a poplist 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 List Style to Poplist (this is the default).

Set the Mapping of Other Values to 11. Set the Initial Value to 0.

Select the Elements in List property, and then 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 box.

Open the Layout Editor, and resize the item so that elements do not truncate.

2. In the ORDGXX 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 Value when Checked to online.

Set 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.

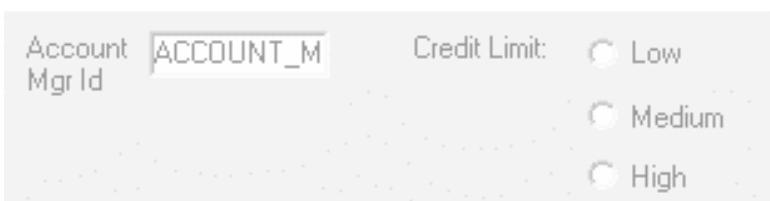
- Save and compile the form.

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

No formal solution

- In the CUSTGXX 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. Arrange as shown below:



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 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.

- Save, compile, and run the form to test the changes.

No formal solution

## Practice 10 Solutions

1. In the ORDER\_ITEMS block of the ORDGXX 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 Display Item to the right of the Product\_Id. Move the other items to the right to accommodate the display item.  
Set Name property of the display item to DESCRIPTION.  
Set Maximum Length to 50.  
Set Width to 80. Set Height to 14.  
Set 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 ORDGXX 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.

The screenshot shows the Oracle Forms Developer 10g Layout Editor. The window title is "Order Information".  
Top section (Order Information):

- Order Id: ORDER\_ID
- Online?:
- Order Status: ORDER\_STATUS
- Customer Id: CUSTOMER\_ID
- Customer Name: CUSTOMER\_NAME
- Sales Rep Id: SALES\_R
- Sales Rep Name: SALES REP NAME

  
Middle section (Order Details):

Item#	Product Id	Description	Unit Price	Quantity	Item Total
LINE_1	PRODUCT_ID	DESCRIPTION	UNIT PRICE	QUANTITY	ITEM TOTAL
LINE_2	PRODUCT_ID	DESCRIPTION	UNIT PRICE	QUANTITY	ITEM TOTAL
LINE_3	PRODUCT_ID	DESCRIPTION	UNIT PRICE	QUANTITY	ITEM TOTAL
LINE_4	PRODUCT_ID	DESCRIPTION	UNIT PRICE	QUANTITY	ITEM TOTAL

  
Bottom section (Order Items):

LINE#	PRODUCT_ID	DESCRIPTION	UNIT PRICE	QUANTITY	ITEM TOTAL
LINE_1	PRODUCT_ID	DESCRIPTION	UNIT PRICE	QUANTITY	ITEM TOTAL
LINE_2	PRODUCT_ID	DESCRIPTION	UNIT PRICE	QUANTITY	ITEM TOTAL
LINE_3	PRODUCT_ID	DESCRIPTION	UNIT PRICE	QUANTITY	ITEM TOTAL
LINE_4	PRODUCT_ID	DESCRIPTION	UNIT PRICE	QUANTITY	ITEM TOTAL

  
On the right side of the window, there is an image item labeled "IMAGE: PRODUCT\_IMAGE" with a placeholder image showing a large "X". Below it is a text item labeled "IMAGE\_DESCRIPTION".

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 Name to `IMAGE_DESCRIPTION`.

Set Synchronize with Item to `Description`.

Set Maximum Length property to 0.

Set Database Item to No.

Set Number of Items Displayed to 1.

Set Width to 90. Set Height to 14.

4. In the `CONTROL` block of the `ORDGXX` 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 `Product_Id`.

Set 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 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 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, as shown below. 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 ORDGXX 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 CUSTGXX 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 CUSTGXX 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 20, 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 in the upper left of the canvas.**

**Set 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**

## Practice 11 Solutions

1. Modify the window in the CUSTGXX 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 CV\_Customer 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 and Height 250.

The suggested X, Y positions are 10, 10.

Note: You can change the size of the window by selecting View > Show View and then resizing the viewport in the Layout Editor.

2. Save, compile, and run the form to test the changes.

No formal solution

3. Modify the window in the ORDGXX 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 right-most positions of objects on the CV\_Order 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 275.

The suggested X, Y positions are 10, 10.

Note: You can change the size of the window by selecting View > Show View and then resizing the viewport in the Layout Editor.

4. In the ORDGXX form, create a new window called WIN\_INVENTORY that is 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 right-most positions of objects on the CV\_Inventory canvas, and plan the height and width for the window. Look at the CV\_Order 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 WIN\_INVENTORY window. 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 Window property to WIN\_INVENTORY for the CV\_INVENTORY canvas.**  
**Click Run Form to compile and run the form.**
6. Save the form.  
**No formal solution**

Oracle Internal & OAI Use Only

## Practice 12 Solutions

### Toolbar Canvases

1. In the ORDGXX 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

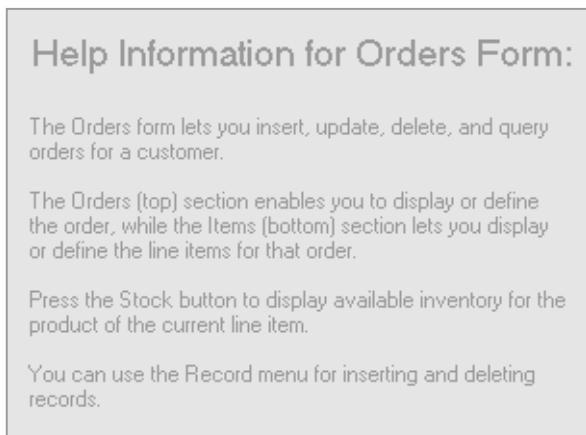


In the Layout Editor of the TOOLBAR canvas, with the CONTROL block selected, use the Button tool to create three push buttons. Select all three, invoke the Property Palette, and set the Mouse Navigate, Keyboard Navigable, Height, Width, and Background Color properties. Then select each button individually 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 ORDGXX 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:



#### 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 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.

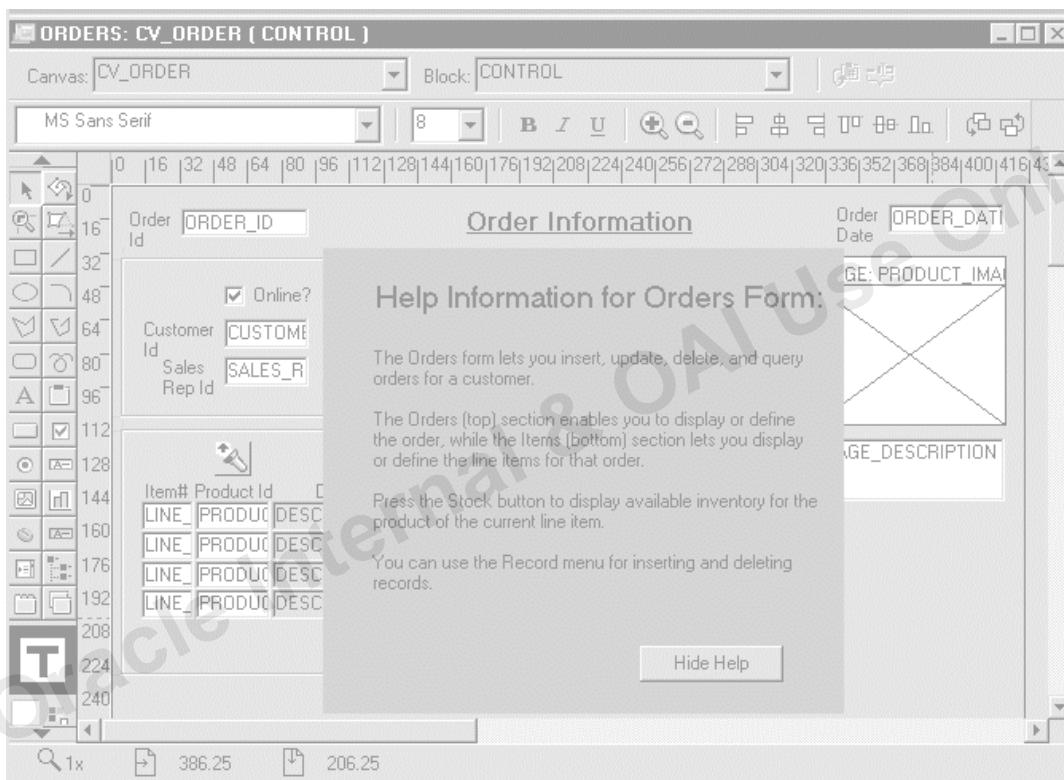
Do this by planning the top-left position of the view in the Layout Editor, while showing CV\_ORDER. Define the Viewport X and Viewport Y Positions in the Property Palette. You can resize the view in the Layout Editor.

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 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. You may need to adjust the size of the canvas because of the Java fonts used at run time.**
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



## Practice 12 Solutions (continued)

### Tab Canvases

Modify the CUSTGXX form 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, the Corner Style property to Square, and the 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, the Corner Style property to Square, and the 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 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.**

The screenshot shows a tab canvas with three tabs: Name, Contact Information, and Account Information. The Name tab is currently selected. Inside the Name tab, there are two input fields: 'First Name' containing 'CUST\_FIRST\_NAME' and 'Last Name' containing 'CUST\_LAST\_NAME'. The background of the tab canvas is gray, and the tabs are white with black text.

## 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`.

The screenshot shows the Oracle Forms Layout Editor interface. At the top, there is a tab bar with three tabs: "Name", "Contact Information", and "Account Information". The "Contact Information" tab is currently selected. Below the tabs, there are several input fields arranged in a grid-like layout. The first row contains "Address" (with value "CUST\_ADDRESS\_STREET\_ADDRESS") and "City" (with value "CUST\_ADDRESS\_CITY"). The second row contains "Zip" (with value "CUST\_ADDF"), "Country Id" (with value "CUS"), and "State Province" (with value "CUST\_ADDF"). The third row contains "Email" (with value "CUST\_EMAIL") and "Phone Numbers" (with value "PHONE\_NUMBERS"). A gray rectangle is drawn around the "Phone Numbers" field.

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.

The screenshot shows the Oracle Forms Layout Editor interface. At the top, there is a tab bar with three tabs: "Name", "Contact Information", and "Account Information". The "Account Information" tab is currently selected. Below the tabs, there are two input fields. The first field is "Account Mgr Id" (with value "ACCOUNT\_MG") and has a small icon next to it. The second field is "Credit Limit" (with value "Low") and contains three radio buttons labeled "Low", "Medium", and "High". A gray rectangle is drawn around the "Credit Limit" field.

## Practice 12 Solutions (continued)

The content canvas should now look similar to the following screenshot:

The screenshot shows a content canvas titled "Customer Information". At the top left, there is a button labeled "ID" with a dropdown menu showing "CUSTOM". Below the title, there is a horizontal tab bar with three tabs: "Name", "Contact Information", and "Account Information". The "Name" tab is selected. Under the "Name" tab, there are two input fields: "First Name" containing "CUST\_FIRST\_NAME" and "Last Name" containing "CUST\_LAST\_NAME".

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 after you tab out of that item initially.

**Note:** Because 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 CUSTGXX 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 the 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 ORDGXX 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 Exit\_Button is clicked.

**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 Product\_Lov\_Button and that the form exits when you press 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 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.TRIGGER\_BLOCK system variable 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 CUSTGXX.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, enter 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. Compile your form and click Run Form to run it. 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. In 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 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 CUSTGXX 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 have outstanding credit orders exceeding the credit limits, so you should receive the warning for most customers. (If you want 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 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 ORDGXX 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, width, height, 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 Name to Image\_Button.**

**Set the Keyboard Navigable property to No.**

**Set the Mouse Navigate property to No; set Width and Height to 50 and 16, respectively.**

**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 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 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 CUSTGXX 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 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 ORDGXX 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 ORDGXX 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 the 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 the 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 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 CUSTGXX 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.

**Note:** If the background color looks different from that of the canvas, click in the Background Color property and click Inherit on the Property Palette toolbar.

**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 (check box):**

```
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 When-Checkbox-Changed trigger.

**Note:** If the background color looks different from that of the canvas, click in the Background Color property and click Inherit on the Property Palette toolbar.

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. Modify the properties of the CONTROL block so that the check boxes can be checked or unchecked at run time.

**Hint:** The CONTROL block contains a single new record.

Set the block property Insert Allowed to Yes.

9. 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.

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

No formal solution

## Practice 19 Solutions

1. In the CUSTGXX 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 ORDGXX 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 the 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 ORDGXX form, create triggers to write the correct values to Customer\_Name whenever validation occurs on Customer\_Id, and to the Sales\_Rep\_Name when validation occurs on Sales\_Rep\_Id. Fail the triggers 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

- When the ORDGXX 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 the form level:**

```
SET_CUSTOM_PROPERTY('order_items.quantity',1,  
    'FILTER_TYPE','NUMERIC');  
GO_BLOCK('ORDERS');  
EXECUTE_QUERY;
```

- 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 the ORDER\_ITEMS block:**

```
get_image;
```

- Define the same trigger type and code on the ORDERS block.

**When-New-Record-Instance on the ORDERS block: get\_image;**

- 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.**

- 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 CUSTGXX 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 CUSTGXX 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 ORDGXX 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 the 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 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**

## Practice 21 Solutions (continued)

6. Open the CUSTGXX form module. Create three global variables called GLOBAL.INSERT, GLOBAL.UPDATE, and GLOBAL.DELETE. These variables indicate the number of inserts, updates, and deletes, respectively. You need to write Post-Insert, Post-Update, and Post-Delete triggers to initialize and increment the value of each global variable. You may import the code from pr21\_6a.txt, pr21\_6b.txt, and pr21\_6c.txt.

**Post-Insert at the 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 message line to be displayed. 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 ORDGXX 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;
```

Oracle Internal & OAI Use Only

## Practice 22 Solutions (continued)

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, Width: 16, 50	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 CUSTGXX 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 ORDGXX form to the Blocks\_Button of the CUSTGXX form. Run the CUSTGXX 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 ORDGXX 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 STOCKXX.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`.

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.**

**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 ORDGXX form, drag the CONTROL block, the Toolbar, and the Question\_Alert to the Personal tab of the Object Library.**

**Save the Summit object library as `summit.olb`.**

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 to 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. Open the Toolbar canvas and move the Exit button close to the Show Blocks button.

**Follow the practice steps; for a solution, see the `dept.fmb` file in the `solutions\Practice23` directory.**

10. Save this form as DEPTGXX, 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?"

Run the Departments form again. You should see the Exit button is now gray, rather than white as it was before.

13. In the new unsaved form, 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 `Employees.fmb` in the `Solutions\Practice23` directory for an example.

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.

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. 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  
    IF :orders.customer_id IS NOT NULL THEN
```

## Practice 23 Solutions (continued)

```
SELECT cust_first_name || ' ' || cust_last_name
  INTO :orders.customer_name
  FROM customers
 WHERE customer_id = :orders.customer_id;
END IF;
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
 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
   AND job_id = 'SA REP';
 END IF;
EXCEPTION
 WHEN NO_DATA_FOUND THEN
 MESSAGE('Invalid sales rep id');
 RAISE form_trigger_failure;
END;
```

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

```
get_customer_name;
```

In the When-Validate-Item trigger for the Orders.Sales\_Rep\_Id item, replace the existing code with:

```
get_sales_rep_name;
```

### **Practice 23 Solutions (continued)**

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.

Oracle Internal & OAI Use Only

## Practice 24

1. In the ORDGXX form, integrate the WebUtil objects and library.  
Open the `Ordgxx.fmb` form. Also open the `webutil.olb` object library from the `<DS_HOME>\forms` directory.  
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.

See the previous lesson for the steps to attach a PL/SQL library to a form. Attach the WebUtil\_lib library (the `webutil_lib.p11` file) from the `<DS_HOME>\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`

## Practice 24 (continued)

Click OK. Click Run.

The first time you run a form with WebUtil configured, you may need to respond to a 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 box, 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 inst\_loc environment variable. You can import the code from pr24\_4a.txt.

To the end of the trigger, add the code:

```
DECLARE
    v_message VARCHAR2(250);
BEGIN
    CLIENT_TOOL_ENV.GETVAR('inst_loc',v_message);
    MESSAGE('inst_loc=' || 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;
```

## Practice 25 Solutions

1. In the ORDGXX form, create a Pre-Form trigger to ensure that a global variable called Customer\_Id exists.

**Pre-Form at the 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 the ORDERS block:**

```
:ORDERS.customer_id := :GLOBAL.customer_id;
```

3. Save, compile, and run the form to test that it works as stand alone.

**No formal solution**

4. In the CUSTGXX form, create a CONTROL block button called Orders\_Button and set appropriate properties. Place it on the CV\_CUSTOMER canvas below the Customer\_Id item.

**Create a button.**

**Set 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 ORDGXX 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 CUSTGXX form and test the button to open the Orders form.

**No formal solution**

7. Change the window location of the ORDGXX form, if required.

**No formal solution**

## Practice 25 Solutions (continued)

- Alter the Orders\_Button trigger in CUSTGXX so that it does not open more than one instance of the Orders form, but uses GO\_FORM to pass control to ORDGXX 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 file name in the OPEN\_FORM built-in.

- 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 the ORDERS form:**

```
IF :GLOBAL.customer_id IS NOT NULL
AND :ORDERS.customer_id != :GLOBAL.customer_id THEN
    EXECUTE_QUERY;
END IF;
```

- 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 the ORDERS block:**

```
:ORDERS.customer_id := :GLOBAL.customer_id;
```

- Add code to the CUSTGXX 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;
```

- Save and compile the ORDGXX form. Save, compile, and run the CUSTGXX form to test the functionality.

**No formal solution**

- If you have time, you can modify the appearance of the ORDXX form to make it easier to read, similar to what you see in ORDERS.fmb.



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

# B

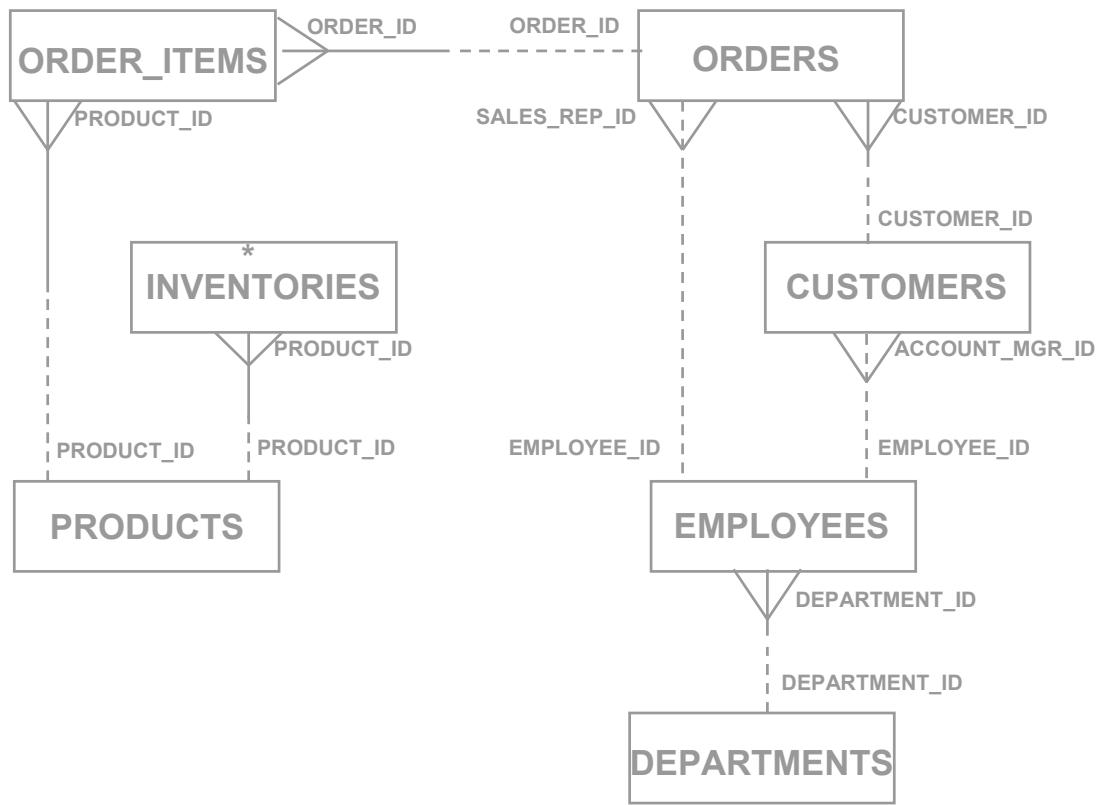
## Table Descriptions

ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

Oracle Internal & OAI Use Only

## 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
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	
PHONE_NUMBERS	NLS_TERRITORY	NLS_TERRITORY
CREDIT_LIMIT		
CUST_EMAIL	ACCOUNT_MGR_ID	

## 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
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
--------------	--------------

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



# Introduction to Query Builder

ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

Oracle Internal & OAI Use Only

# Query Builder: Features

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

ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## What Is Query Builder?

Query Builder is the tool used in the LOV Wizard to build the query on which a record group is based.

### 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.

## What Is Query Builder? (continued)

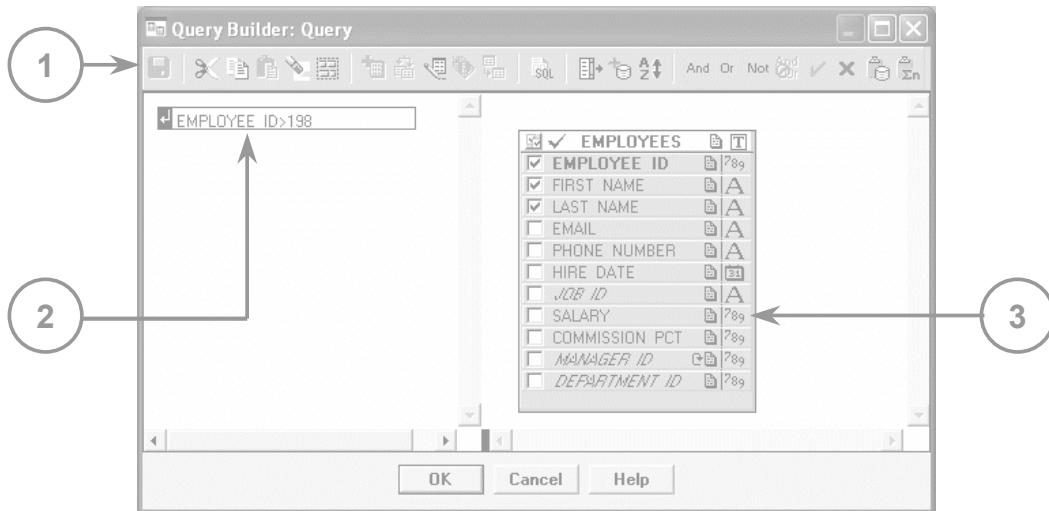
Query Builder is useful for the following reasons:

- 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.
- Graphical representation of tables and their relationships enables you to see the structure of your data.

With Query Builder, 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.
- 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
- Combine and nest conditions graphically using logical operators. You can disable conditions temporarily for *what-if* analysis.

# Query Builder Window



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

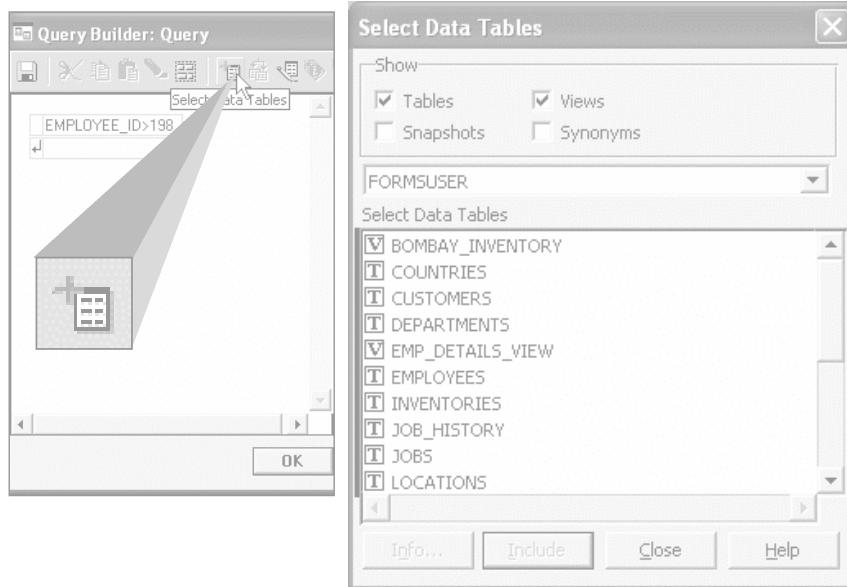
## Query Builder Window

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

The Query Builder window comprises:

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



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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

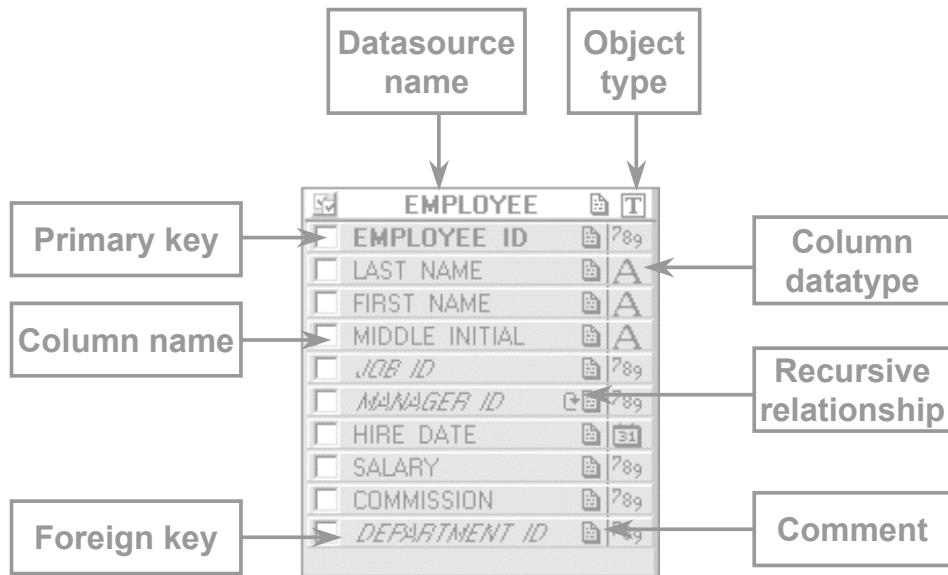
1. Choose Select Data Tables from the toolbar.  
The Select Data Tables dialog box appears.  
**Note:** When you open Query Builder to create a new query, the Select Data Tables dialog box is open by default.
2. Select the table name and then click Include, or simply double-click the desired table name.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



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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 data type.

Columns also provide additional information.

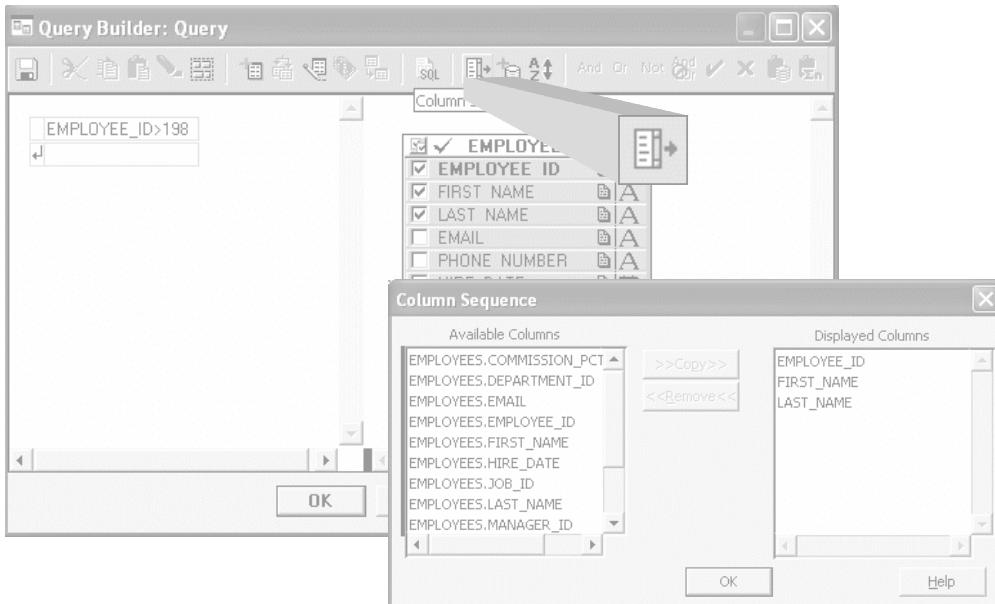
- **Primary keys** are displayed in bold.
- **Foreign keys** are displayed in italics.
- **Recursive relationships** are indicated by a self-relationship icon.

### 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.

Oracle Internal & OAI Use Only

# Refining a Query



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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.

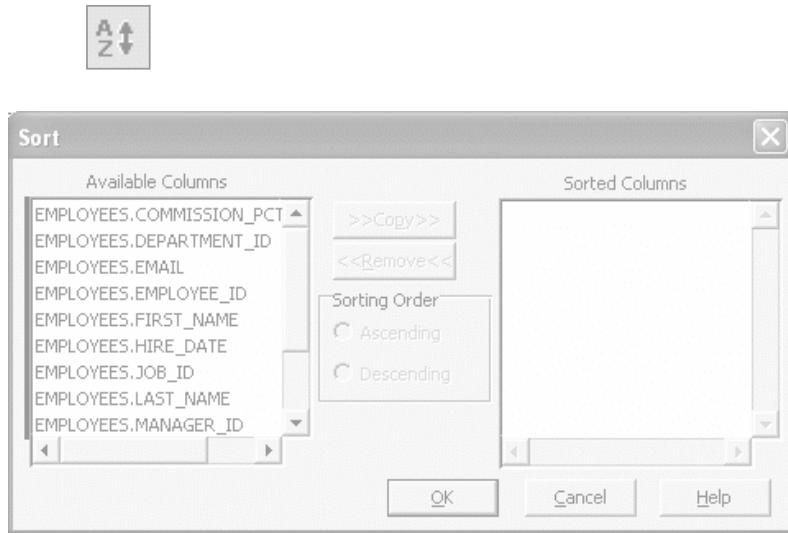
### Changing Column Position in a Query

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 the 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



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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

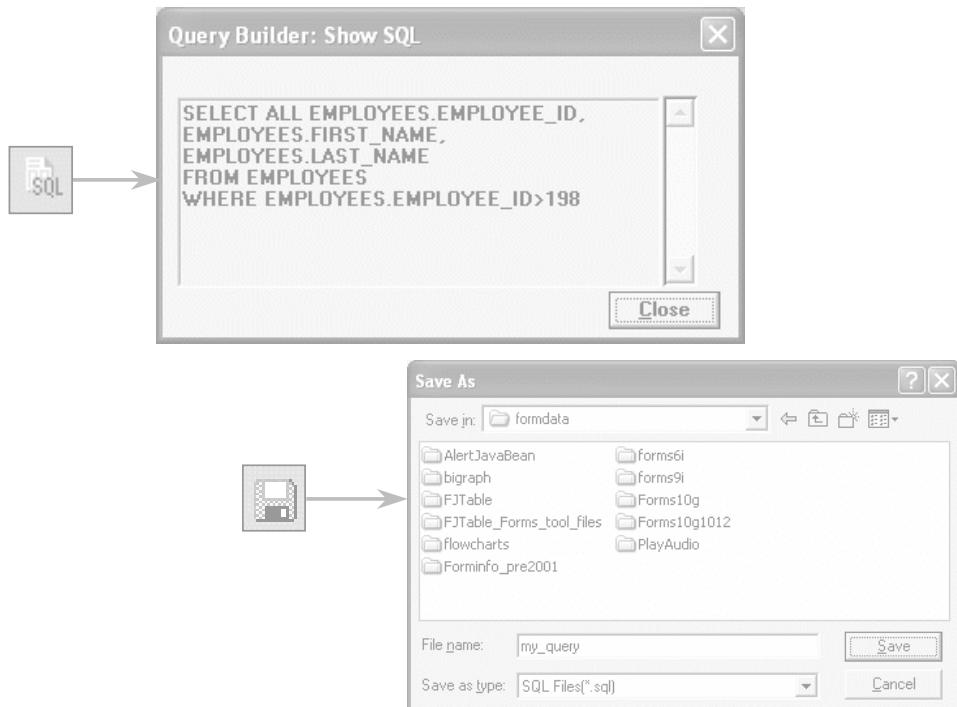
1. Select the Sort tool. The Sort dialog box appears.
2. Select the column you want to sort from the Available Columns list.
3. Select Copy.

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



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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.

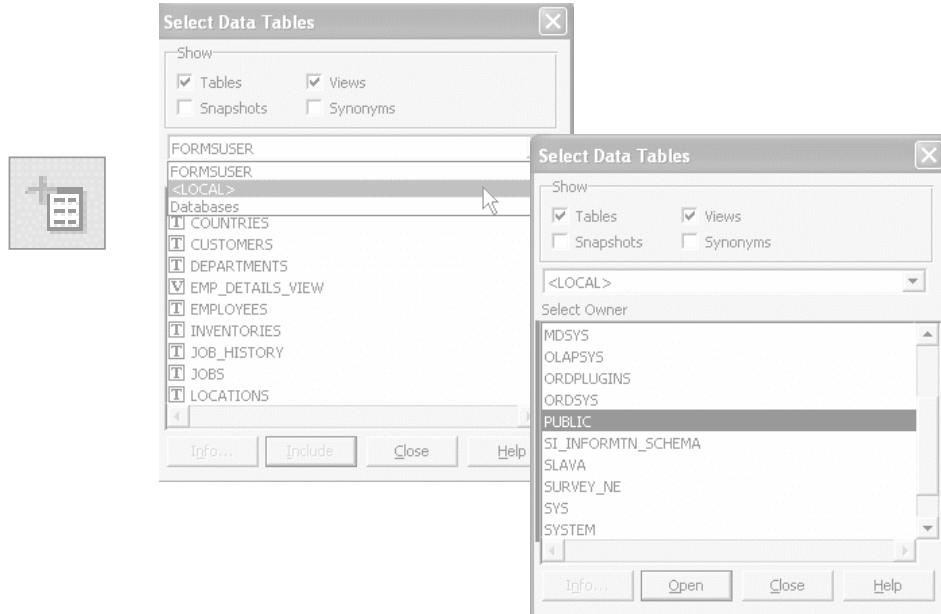
2. Enter a file name.

**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



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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, then 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)

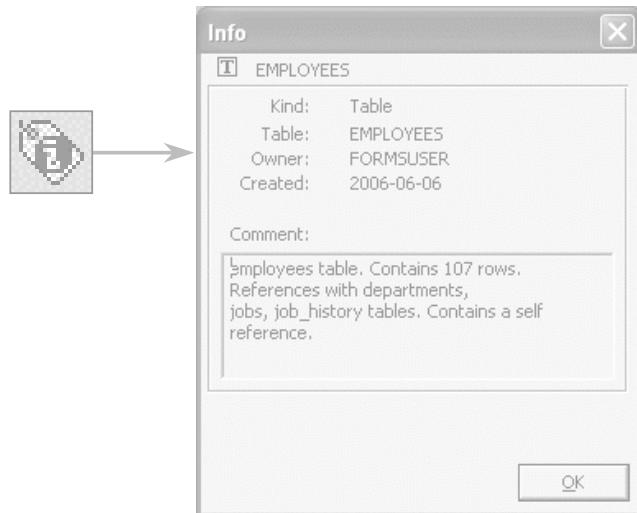
### How to Find Tables in Other Databases

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

Oracle Internal & OAI Use Only

# Viewing Comments



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## How to View Comments

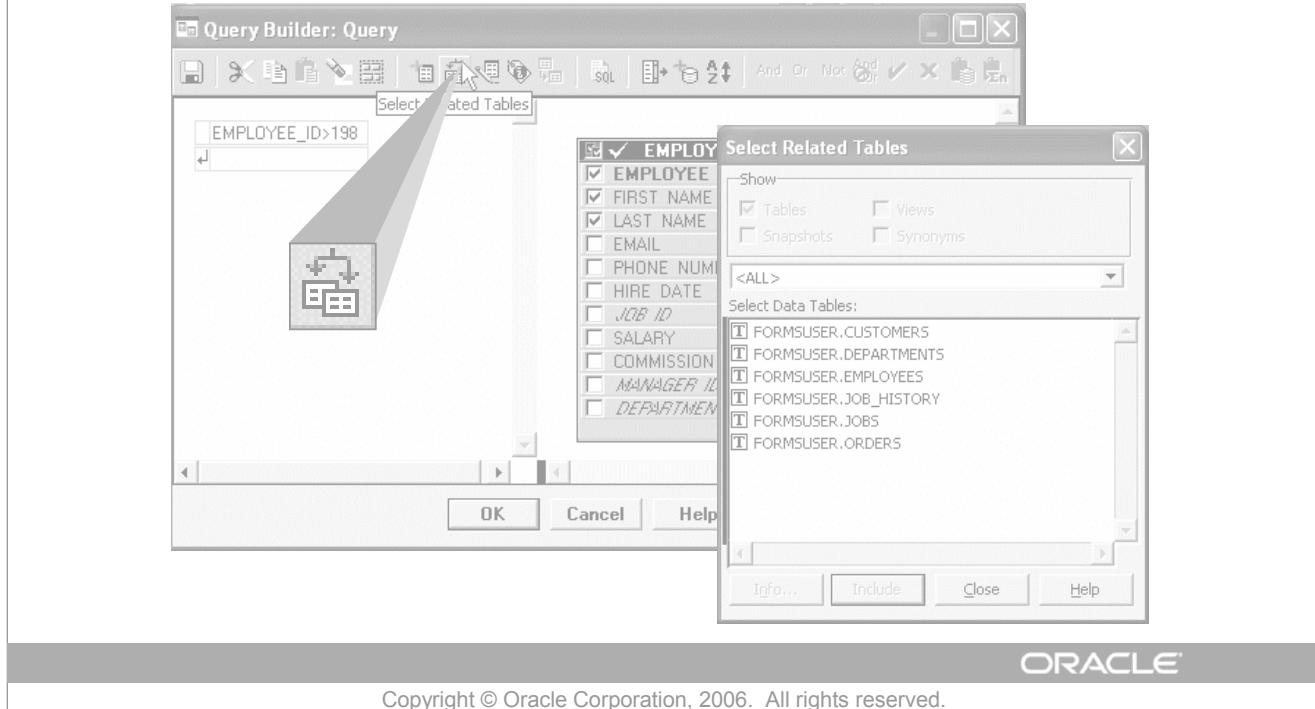
Sometimes, details about the kind of data stored in the database are not reflected by the table or column names. This can often make it difficult to decide which objects to include in your query. Query Builder features the Info dialog box to provide this kind of information for tables and columns.

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.

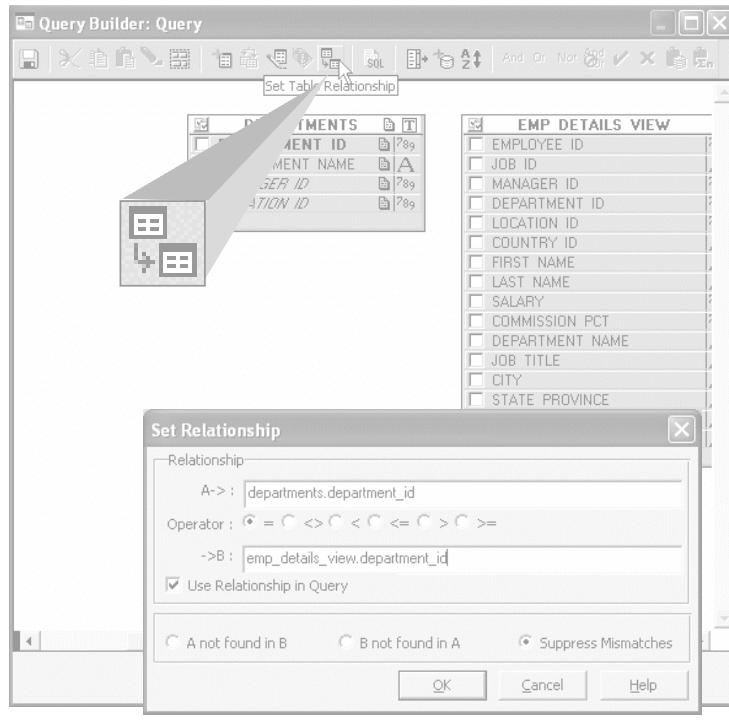
### How to Find and Include Related Tables

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.

After 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



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## Creating a User-Defined Relationship

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

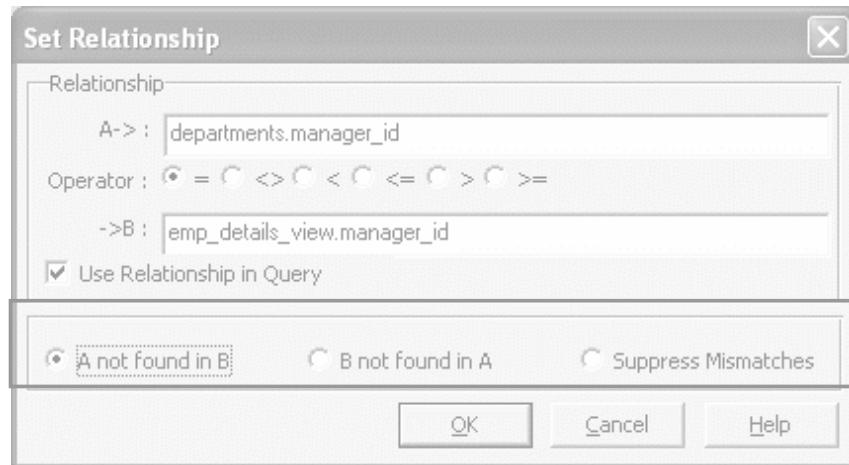
### How to Create a Relationship

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. Enter 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. After the target column is selected, release the mouse button to anchor the relationship line.

# Unmatched Rows



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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.

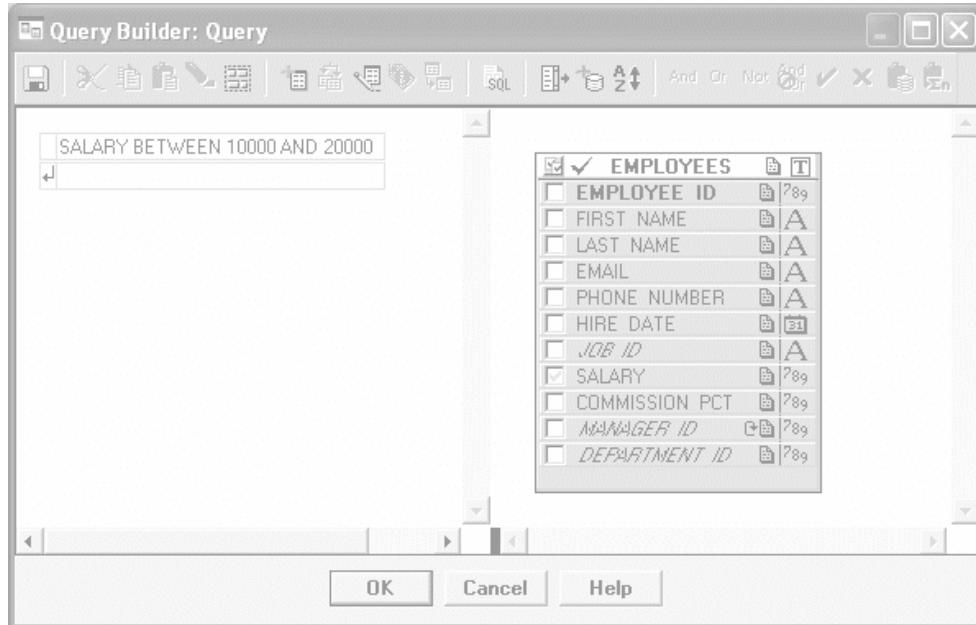
There are three types of relationships that you can choose from:

- 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 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



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## Selecting Rows with Conditions

### 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 in 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:
  - Enter the conditions directly in the Condition field.
  - Click in the columns in the Datasource panel, and then enter the rest of the condition.
  - Select columns and functions, using the appropriate tools.

**Note:** Character and date values must be enclosed in single quotation marks.

## 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.

### 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.

Oracle Internal & OAI Use Only

# 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

ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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:** Is 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.

Oracle Internal & OAI Use Only

# Multiple Conditions



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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.

## **Entering Multiple Conditions (continued)**

To change the logical operator, perform the following steps:

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

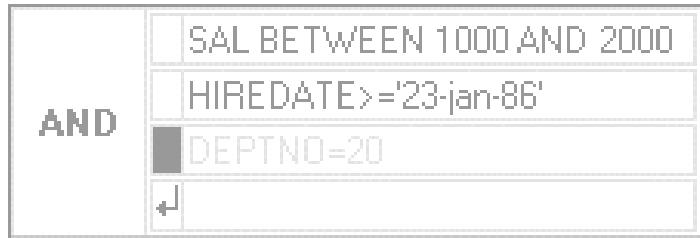
## **How to Create Nested Conditions**

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, perform the following steps:

1. Build each condition to be included in the nest.
2. Press and hold [Shift] and 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



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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

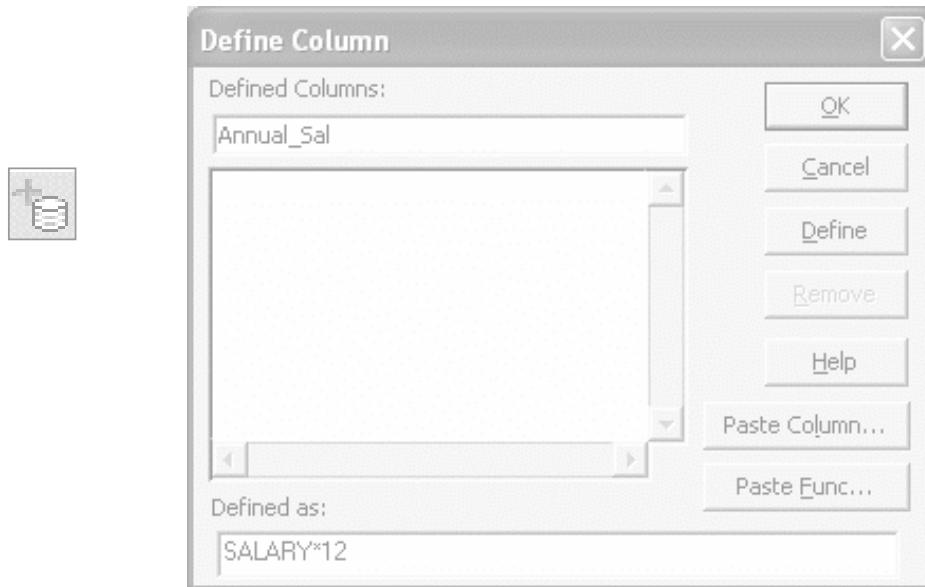
Query Builder enables you to temporarily deactivate a condition so that you can test different scenarios without having to re-create the conditions each time.

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:** Press and hold the [Shift] key to select multiple conditions.
2. Double-click in the box to deactivate the condition.

# Defining Columns by Using an Expression



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## Defining Columns by 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 by Using an Expression (continued)**

### **How to Enter Expressions**

You can enter information in the “Defined as” field in the following ways:

- Enter the expression in directly.
- Select Paste Column.

The Paste Column dialog box appears.

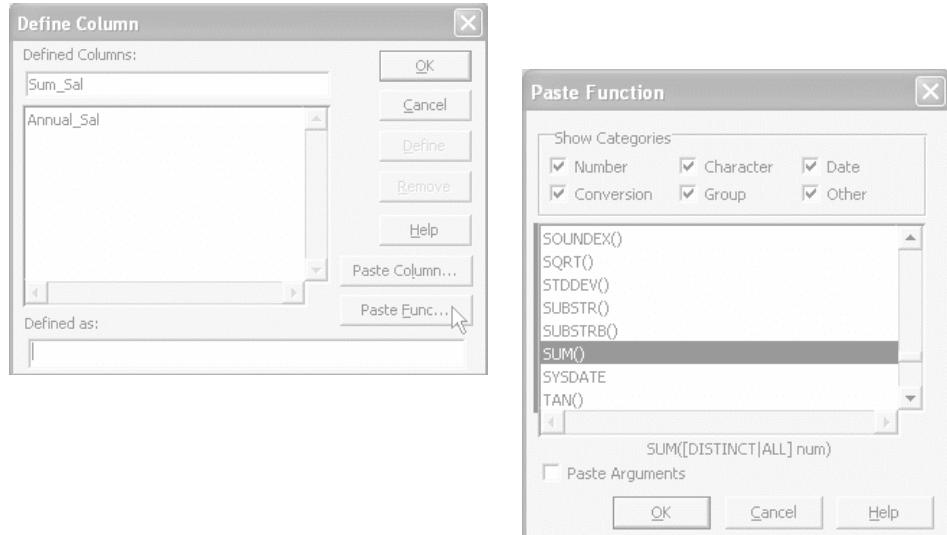
- Select the column from the displayed list.
- Click OK to paste the column into your expression and return to the Define Column dialog box.

### **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.

Oracle Internal & OAI Use Only

# Defining Columns by Using a Function



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## Defining Columns by 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 enter a function directly in the “Defined as” field of the Define Column 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 by 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 data type name of the arguments appropriate for the function is pasted into your expression. You can then replace the description with the actual arguments.

6. Click OK to paste the function into your expression and return to the Define Column dialog box.



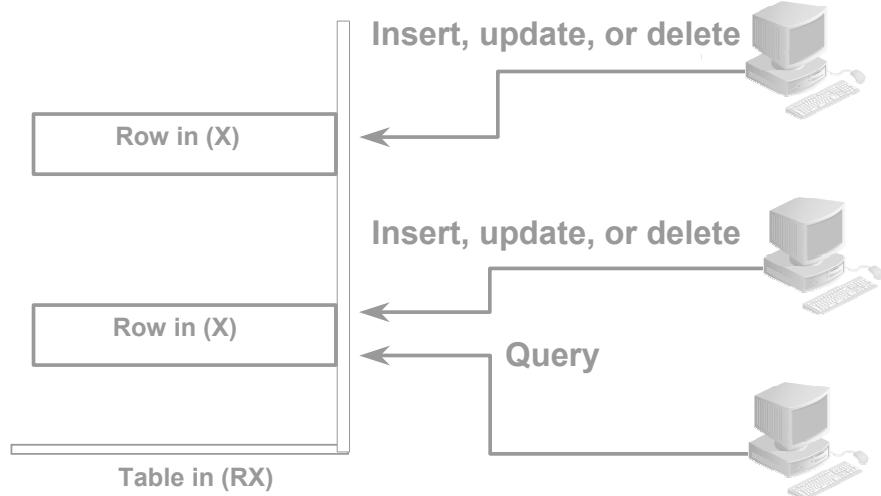
# Locking in Forms

ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

Oracle Internal & OAI Use Only

# Locking



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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:

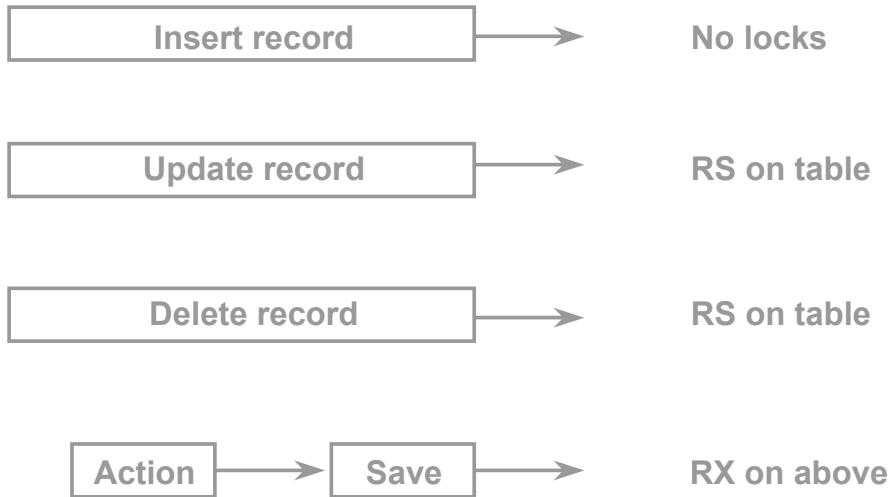
- Oracle uses row-level locking to protect data that is being inserted, updated, or deleted.
- Queries do not prevent other database sessions from performing data manipulation. This also applies to the reverse situation.
- Locks are released at the end of a database transaction (following a rollback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

<b>Lock Type</b>	<b>Description</b>
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



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

### 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.

ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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

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

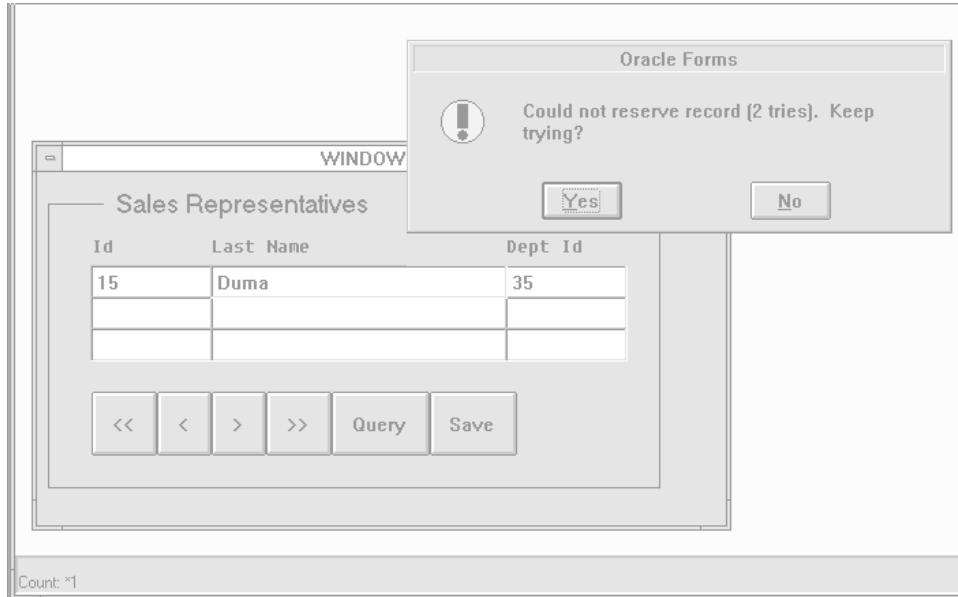
ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

### 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



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

### 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. Because 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.

## User A: Step 3

PERSONNEL					
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

ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

### 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



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

### 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.  
After 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**

ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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.

Oracle Internal & OAI Use Only

# Locking with Built-ins

- **ENTER\_QUERY (FOR\_UPDATE)**
- **EXECUTE\_QUERY (FOR\_UPDATE)**

ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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 following example:

```
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;
```



Copyright © Oracle Corporation, 2006. All rights reserved.

## 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, therefore, 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;
```

Oracle Internal & OAI Use Only

# Oracle Object Features

ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

Oracle Internal & OAI Use Only

# Oracle Scalar Data Types

- **Automatically converted:**
  - FLOAT
  - NLS types
    - NCHAR
    - NVARCHAR2
- **Unsupported:**
  - Time stamp
  - Interval

ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## Oracle Data Types

In Oracle Forms Builder, these data types are automatically converted to existing Forms item data types. Three new scalar data types were introduced with Oracle8i:

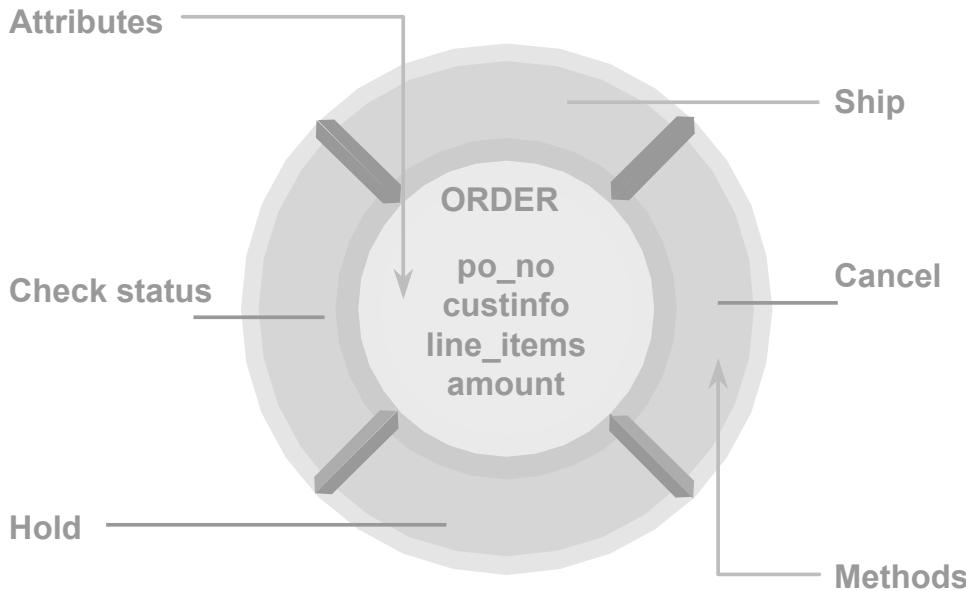
- NCHAR stores fixed-length (blank-padded if necessary) NLS (National Language Support) character data.
- NVARCHAR2 stores variable-length NLS character data.
- FLOAT is a subtype of NUMBER. However, you cannot specify a scale for FLOAT variables. You can specify only a binary precision, which is the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

**NLS (National Language Support) Types in Oracle:** In the Oracle database, data stored in columns of NCHAR or NVARCHAR2 data types 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 data types: 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:** Time stamp and interval data types (new with Oracle9i). You cannot use the Data Block Wizard to create a block when columns are of these data types. As an alternative, you can create blocks manually and then create DATETIME items referencing these database columns.

# Object Types



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## Object Types

An object type is a user-defined composite data type. Oracle requires enough knowledge of a user-defined data type to interact with it. Thus, an object type can be similar to a record type and also 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 data types. 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

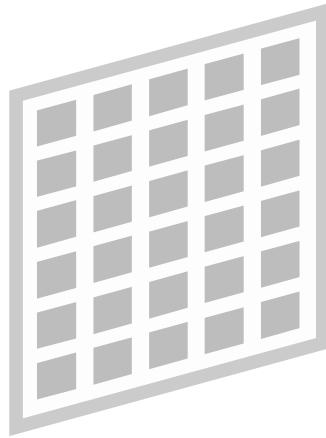
An object type is also similar to a package. After an object is declared, its attributes are similar to package variables. Like packages, object types can contain procedures and functions. In object types, these subprograms are known as *methods*. A method describes the behavior of an object type.

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.

The following 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

ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## Creating Oracle Objects

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

### Object Tables

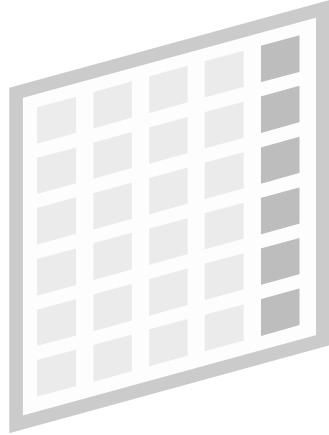
You can create an object by creating a table whose rows are objects of that object type. Here is an example of an object table declaration:

```
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.

# Object Columns



Object column based on object type

ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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:

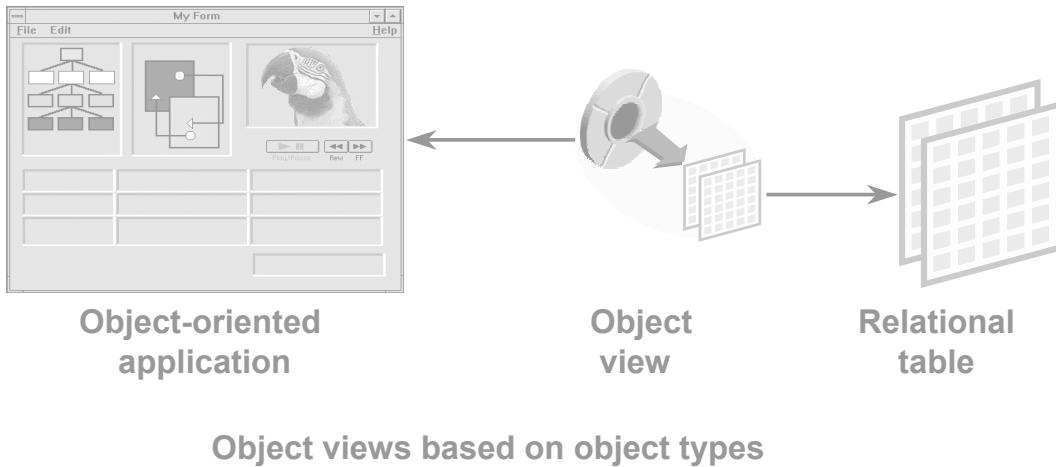
```
CREATE TABLE o_customer (
    custid          NUMBER (6) NOT NULL,
    name            VARCHAR2 (45),
    repid           NUMBER (4) NOT NULL,
    creditlimit     NUMBER (9,2),
    address         address_type,
    phone           phone_type);
```

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 lesson.

# Object Views



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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.

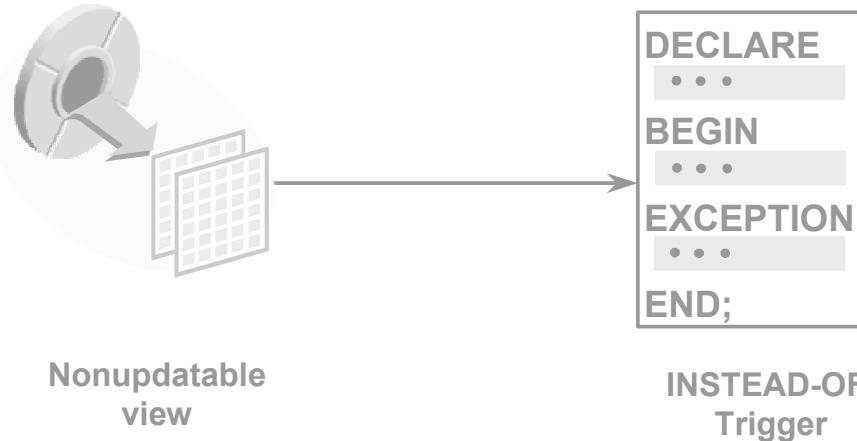
## Object Views (continued)

The following 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;
```

Oracle Internal & OAI Use Only

# INSTEAD-OF Triggers



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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).

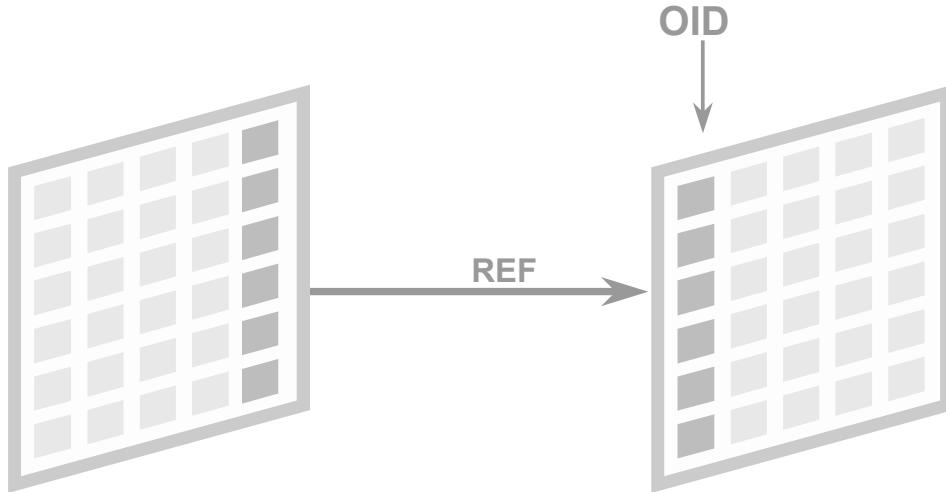
These triggers are called INSTEAD-OF triggers because, unlike other types of triggers, the Oracle server fires the trigger instead of executing the triggering statement. The trigger performs update, insert, or delete operations directly on the underlying tables.

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



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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 REF data type.

## Referencing Objects (continued)

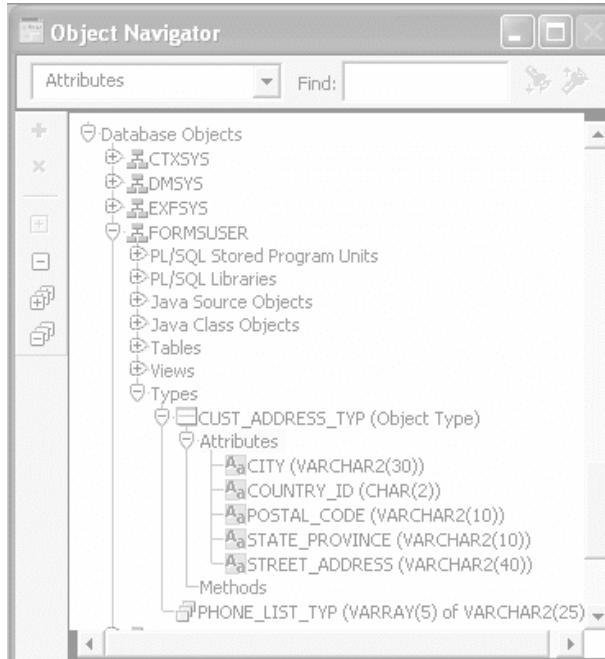
**Note:** Object columns are not assigned OIDs and cannot be pointed to by a REF.

The following is an example of a table declaration that includes a column with a REF data type:

```
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



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## Displaying Oracle Objects in the Object Navigator

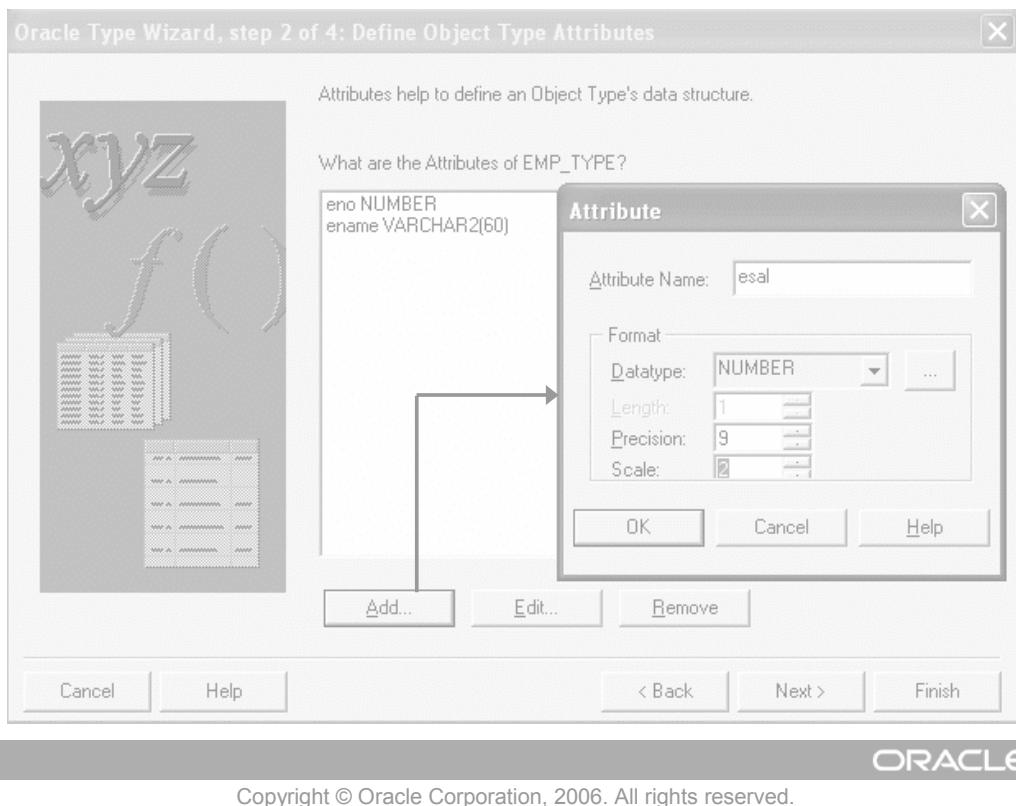
The Object Navigator lists declared types in the “Database Objects” section, along with tables, views, and other Oracle objects.

### 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.

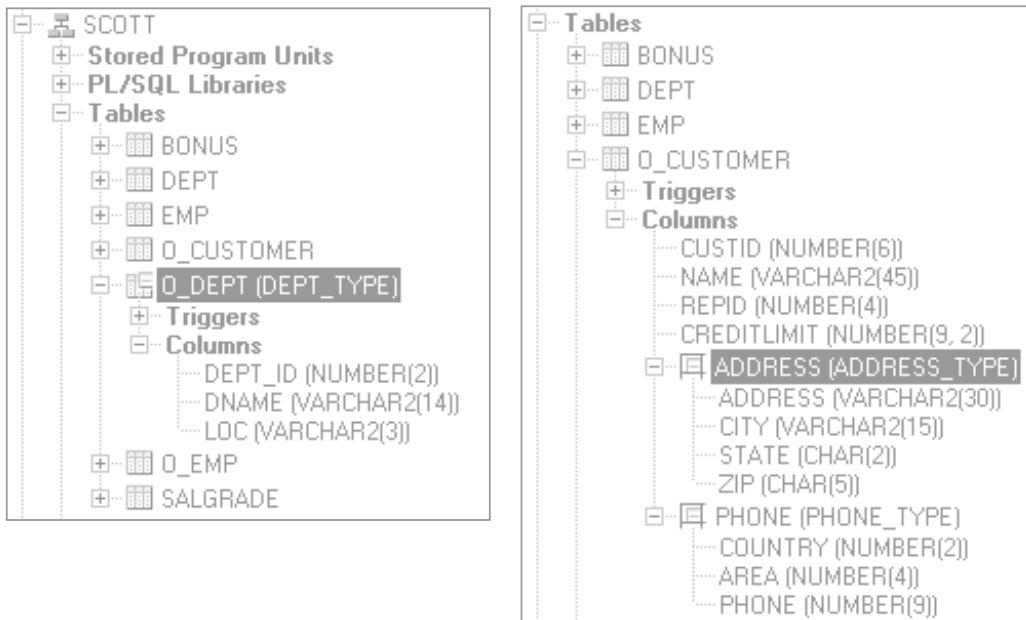
## Oracle Type Wizard



### Oracle Type Wizard

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

# Object Tables and Columns in Object Navigator



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## Object Tables and Columns

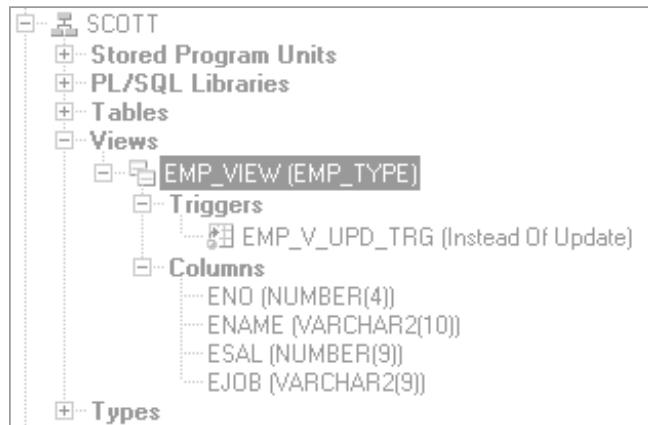
### Object Tables

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



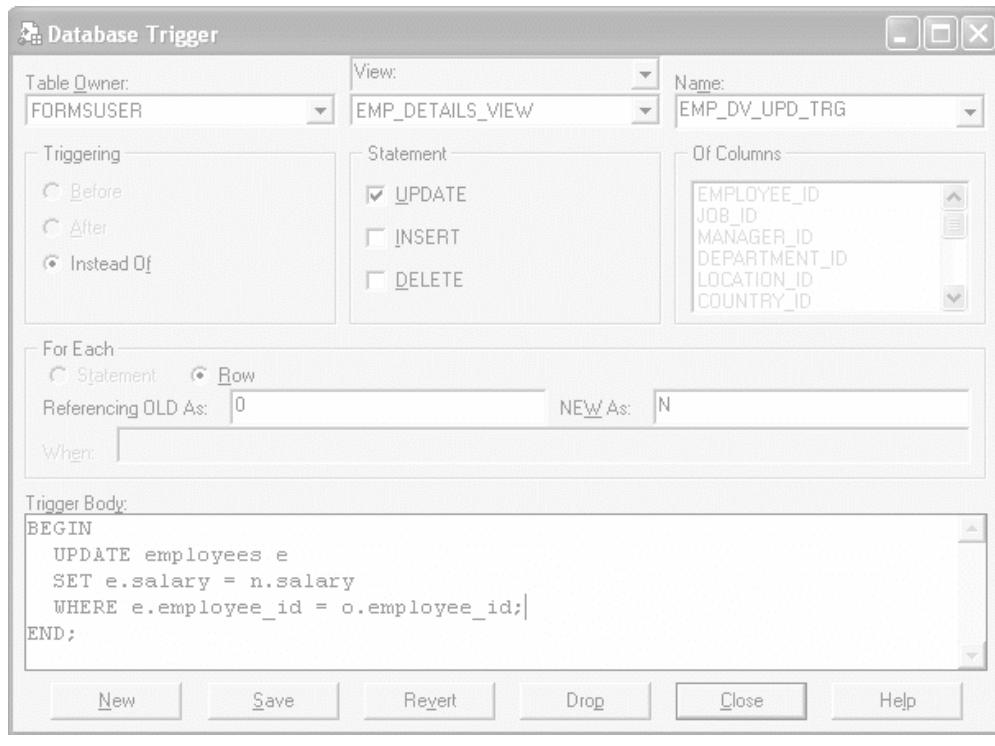
ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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



Copyright © Oracle Corporation, 2006. All rights reserved.

## 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.

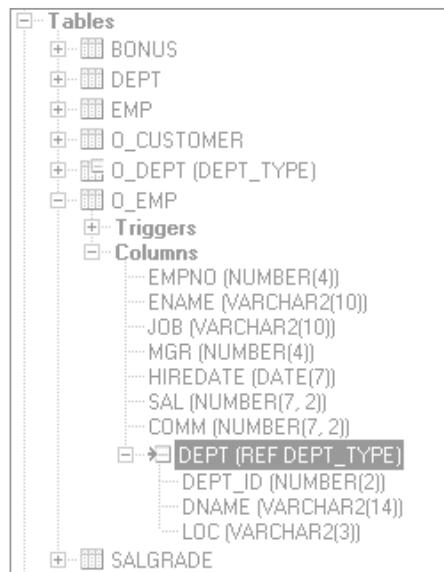
**INSTEAD-OF Triggers:** INSTEAD-OF database triggers can now be created through the trigger creation dialog box, just like any other database trigger.

INSTEAD-OF INSERT, UPDATE, and DELETE triggers enable 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



ORACLE®

Copyright © Oracle Corporation, 2006. All rights reserved.

## 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.



# Using the Layout Editor

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

Oracle Internal & OAI Use Only

# 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

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Why Use the Layout Editor?

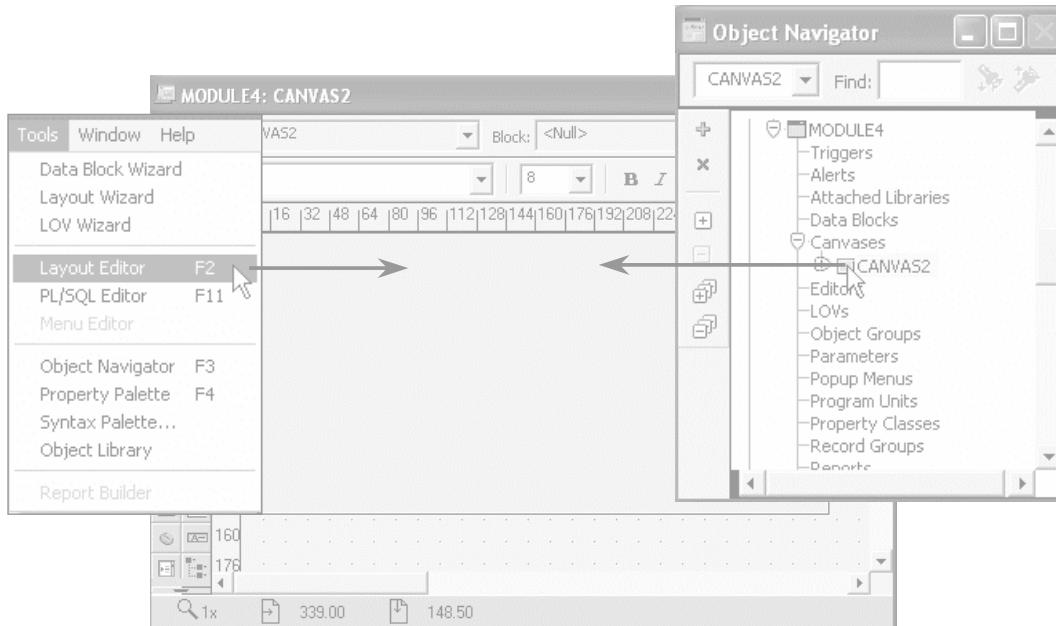
The Layout Editor is a graphical tool for defining the arrangement and appearance of visual objects. The Layout Editor opens windows in Forms Builder to present the surfaces on which you can arrange objects.

The following are the 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



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## 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)**

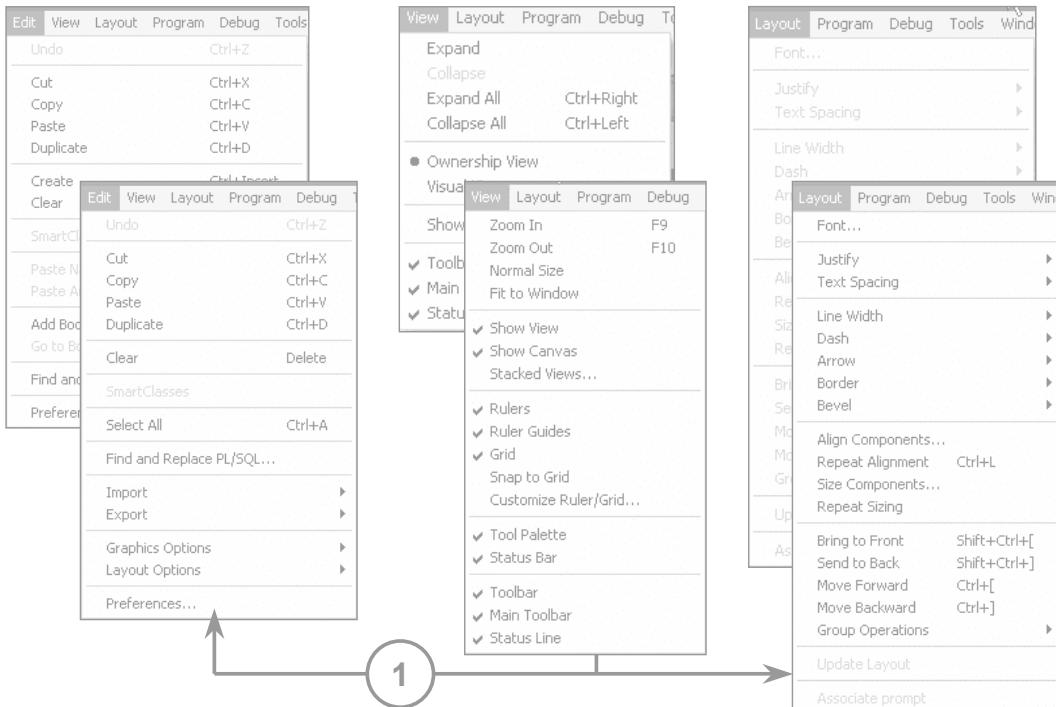
In Forms, you can open several Layout Editor windows—one for each canvas view. Make sure that you have the canvas that you want.

### **Closing the Layout Editor**

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

Oracle Internal & OAI Use Only

# Layout Editor: Components



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

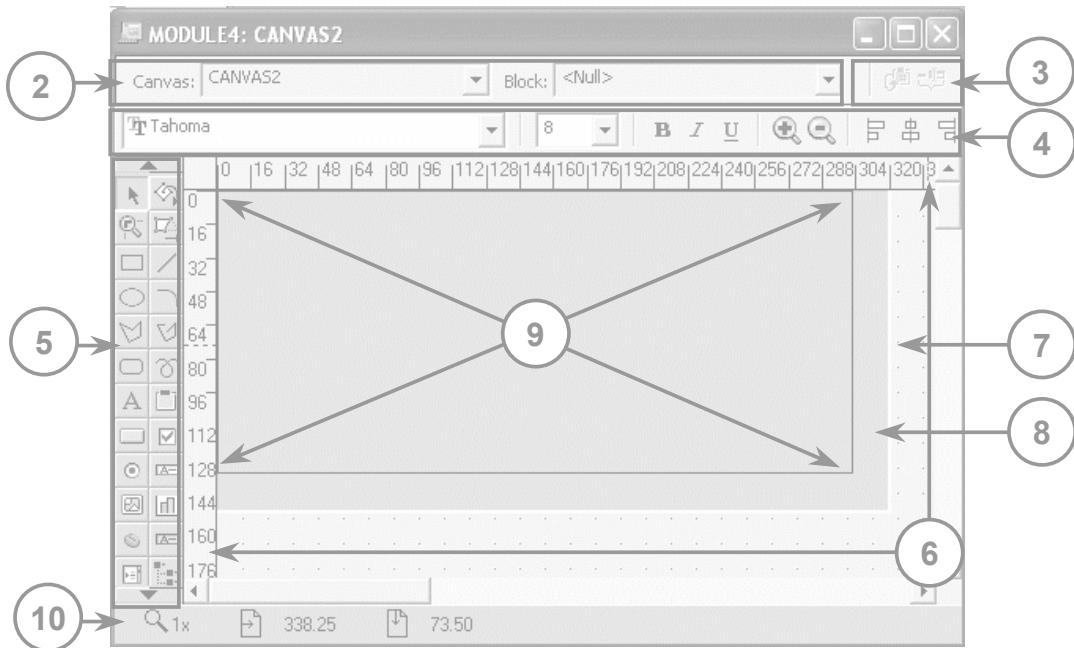
## Components of the Layout Editor

### Common Components of the Layout Editor

- 1. 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 shows the Edit, View, and Layout menus. The screenshots at the top show what these menus look like when the Object Navigator is active, whereas those at the bottom show the expanded choices when the Layout Editor is active.

# Layout Editor: Components



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Components of the Layout Editor (continued)

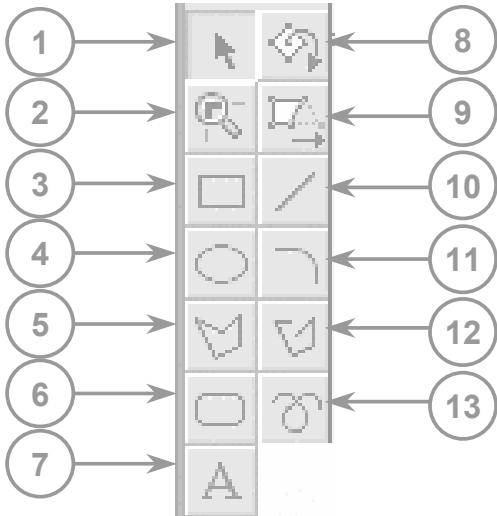
2. **Title bar:** Displays the name of the current form, the name of the canvas being edited, and the name of the current block. When you create an item by drawing it on a canvas in the Layout Editor, Forms Builder assigns the item to the current block, as indicated by Layout Editor block context. You can change the Layout Editor block context using the Block pop-up list on the title bar.
3. **Horizontal toolbar:** Contains buttons that enable you to update the layout or associate text with an item prompt
4. **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.
5. **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)

6. **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.
7. **Layout/Painting region:** Is 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.)
8. **Canvas:** Is 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.
9. **Viewport:** The portion of the canvas that can be seen in the containing window; its height and width are determined by properties of the window rather than by canvas properties.
10. **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.

Oracle Internal & OAI Use Only

# Tool Palette



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## 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 on 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 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 cursor 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.

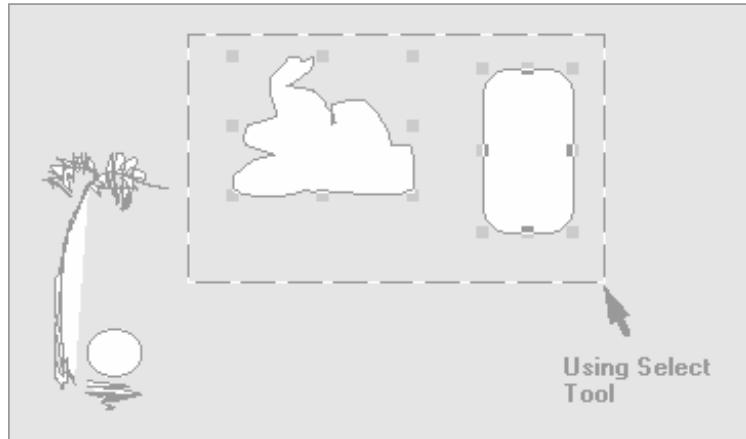
**Note:** You can produce constrained shapes (for example, a circle instead of an ellipse) by pressing the [Shift] key during step 2.

### **Creating Text**

The Text tool (T) lets you open a Boilerplate Text object on the layout. You can enter 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.

Oracle Internal & OAI Use Only

# Selecting Objects



ORACLE®

Copyright © 2006, 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.
- 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.

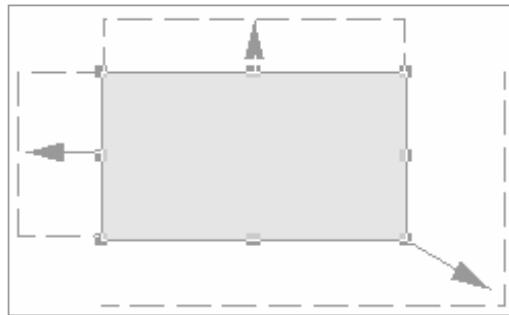
It is convenient to select *several* objects, so that an operation can be performed on them simultaneously.

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

- Press and hold 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

Expand/contract  
in one direction



Expand/contract  
diagonally

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

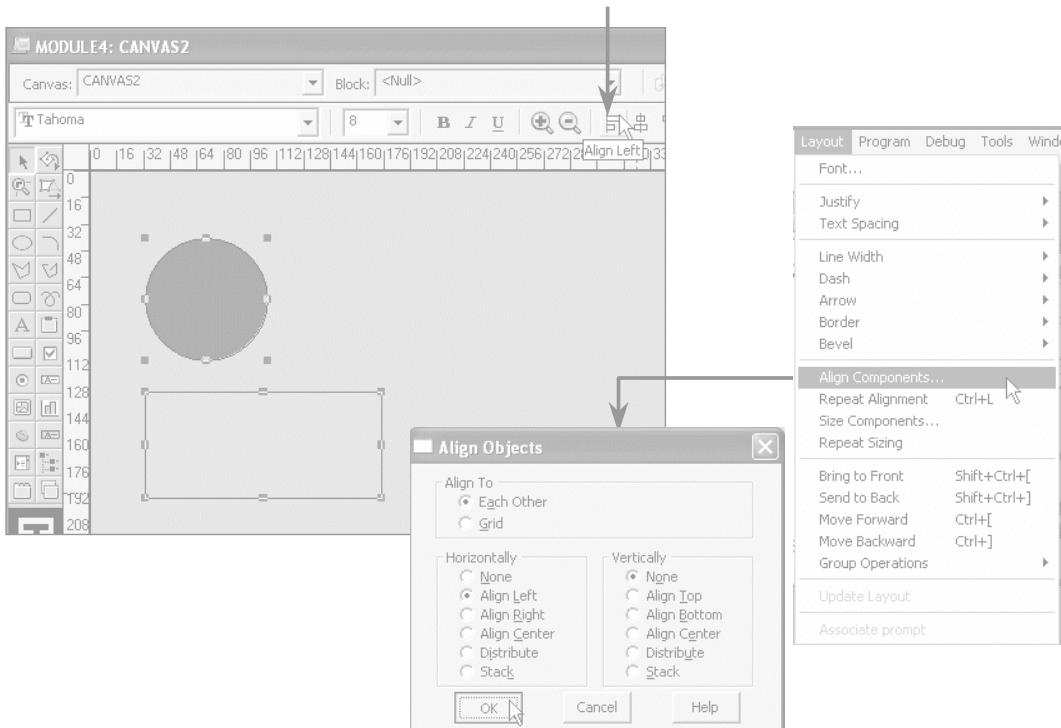
## Changing the Size or Ratio

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

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

**Note:** By pressing and holding 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



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## 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. 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

Oracle Internal & OAI Use Only

# 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.

ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## 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)

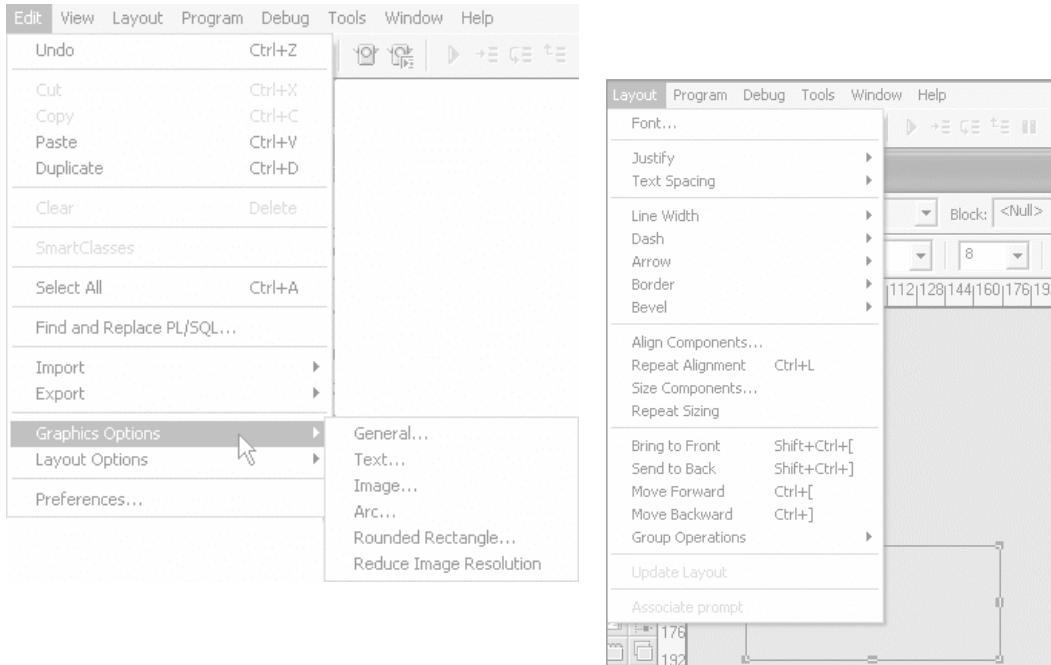
### Other Tools for Manipulating Objects

- **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.

Oracle Internal & OAI Use Only

# Edit and Layout Menus



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## 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**

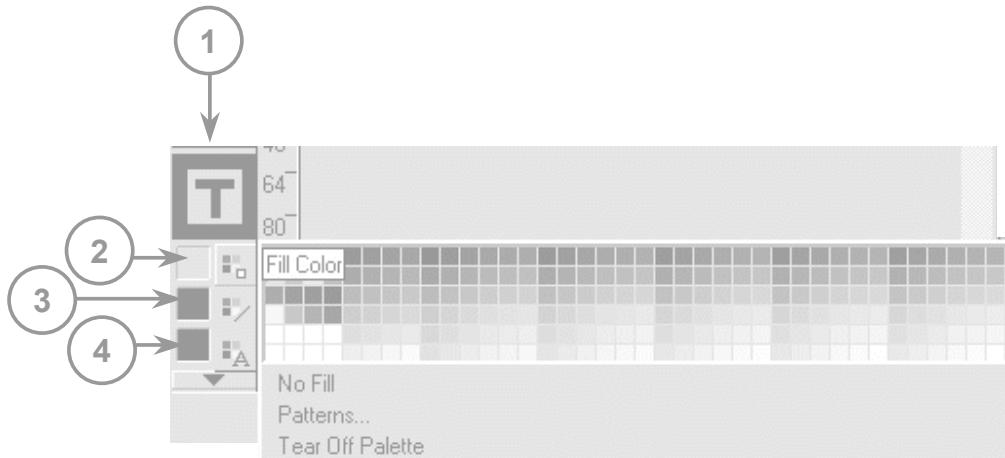
There are a number of ways to change the font characteristics of textual objects. The common steps provided by the Layout Editor are the following:

- 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 then 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.

Oracle Internal & OAI Use Only

# Color and Pattern Tools



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Coloring Objects and Text

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

1. **Sample Window:** This 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.
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 perform 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**

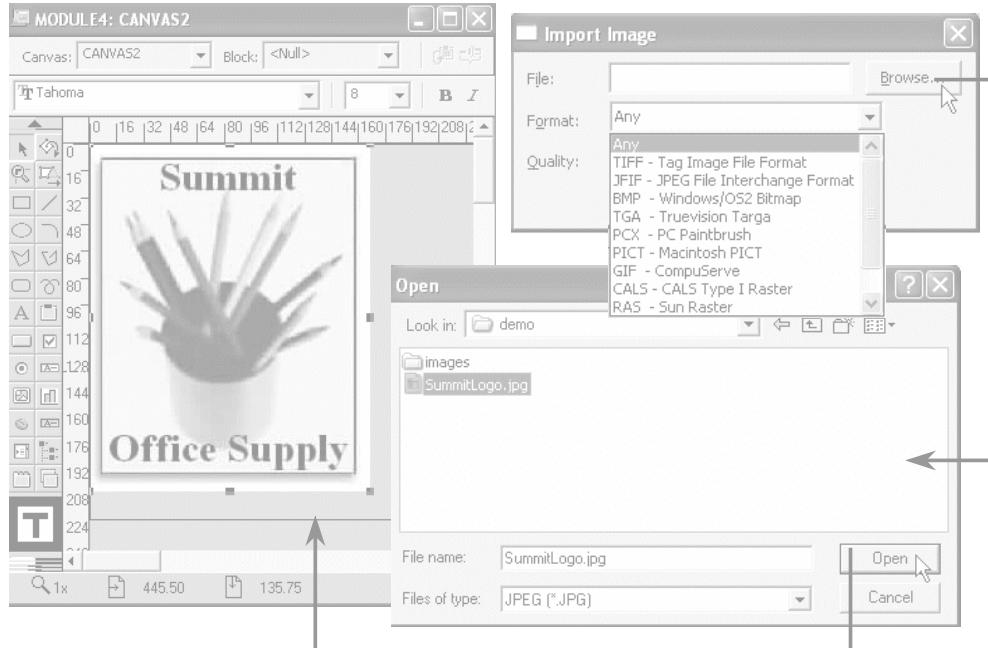
1. Select the textual objects whose color you want to change in the layout.
2. Select the Text Color tool. The color palette appears, showing the available colors.
3. Select a color from the color palette. (Click the appropriate square.)
4. Notice that the selected objects on the layout have adopted 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 box (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.

# Importing Images



ORACLE®

Copyright © 2006, Oracle. All rights reserved.

## Importing Images

Oracle Forms Developer allows both static and dynamic bitmapped 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.

Oracle Internal & OAI Use Only

