Copyright © Oracle Corporation, 2004. All rights reserved.

This documentation contains proprietary information of Oracle Corporation. It is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited. If this documentation is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

**Restricted Rights Legend**

Use, duplication or disclosure by the Government is subject to restrictions for commercial computer software and shall be deemed to be Restricted Rights software under Federal law, as set forth in subparagraph (c)(1)(ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

This material or any portion of it may not be copied in any form or by any means without the express prior written permission of the Education Products group of Oracle Corporation. Any other copying is a violation of copyright law and may result in civil and/or criminal penalties.

If this documentation is delivered to a U.S. Government Agency not within the Department of Defense, then it is delivered with “Restricted Rights,” as defined in FAR 52.227-14, Rights in Data-General, including Alternate III (June 1987).

The information in this document is subject to change without notice. If you find any problems in the documentation, please report them in writing to Worldwide Education Services, Oracle Corporation, 500Oracle Parkway, Box SB-6, Redwood Shores, CA 94065. Oracle Corporation does not warrant that this document is error-free.

Oracle and all references to Oracle Products are trademarks or registered trademarks of Oracle Corporation.

All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

**Author**

Pam Gamer

**Technical Contributors and Reviewers**

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

**Editors**

Nishima Sachdeva
Elizabeth Treacy

**Publisher**

Giri Venugopal
Introduction to Oracle Forms Developer and Oracle Forms Services
Objectives

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

• Define grid computing
• Explain how Oracle 10g products implement grid computing
• Describe the components of Oracle Application Server 10g and Oracle Developer Suite 10g
• Describe the features and benefits of Oracle Forms Services and Oracle Forms Developer
• Describe the architecture of Oracle Forms Services
• Describe the course application
## Internet Computing Solutions

<table>
<thead>
<tr>
<th>Application Type and Audience</th>
<th>Product Approach</th>
<th>Oracle Products</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Enterprise applications, Business developers</strong></td>
<td><strong>Repository-based modeling &amp; generation, Declarative</strong></td>
<td><strong>Oracle Designer, Oracle Forms Developer, &amp; Oracle Forms Services</strong></td>
</tr>
<tr>
<td>Java components, Component developers</td>
<td>Two-way coding, Java and JavaBeans</td>
<td>Oracle JDeveloper, Oracle Application Server 10g</td>
</tr>
<tr>
<td>Self-service applications &amp; content management, Web site developers</td>
<td>Browser-based, Dynamic HTML</td>
<td>Oracle Portal, Oracle Database Server</td>
</tr>
<tr>
<td>Reporting and analytical applications, MIS &amp; business users</td>
<td>Dynamic Web reporting, Drill, Analyzing, Forecasting</td>
<td>Oracle Reports Developer, Oracle Reports Services, Oracle Discoverer, &amp; Oracle Express</td>
</tr>
</tbody>
</table>
Plugging into the Grid

Grid computing is:

- **Software infrastructure** that uses low-cost servers and modular storage to:
  - Balance workloads
  - Provide capacity on demand
- **Made possible by innovations in hardware**
- **Powered by software**
Oracle Enterprise Grid Computing

Oracle's grid infrastructure products:

- Oracle Database 10g
- Oracle Application Server 10g
- Oracle Enterprise Manager 10g
- Grid Control
Oracle 10g Products and Forms Development

Oracle 10g Database

Oracle 10g Application Server

Oracle 10g Developer Suite

Forms Services

Forms Developer
Oracle Application Server 10g Architecture
Oracle Application Server 10g Components
Oracle Forms Services Overview

A component of Oracle Application Server that deploys Forms applications to Java clients in a Web environment.
Forms Services Architecture

Client Tier

- JRE
- Java applet

Middle Tier:

- Forms Listener Servlet
- Forms Servlet
- Forms Runtime
  - User interface layer
  - Application logic layer
  - Data manager/PL/SQL engine
- File containing application code

Database Tier

- Net Services
- DB
Benefits and Components of Oracle Developer Suite 10g

- OWB
- Discoverer
- Reports

Application Development

Business Intelligence

- JDeveloper
- Forms
- Designer
- SCM
Oracle Developer Suite 10g
Application Development

**Application Development - Modeling**
Systems analysis and generation for PL/SQL and Java
Designer

**Application Development - RAD**
Declarative 4GL for PL/SQL and Java
Forms Developer

**Application Development - J2EE and Web Services**
Java and XML IDE
JDeveloper

**Application Development - Team Support**
Software configuration management
Software Configuration Management
Oracle Developer Suite 10g
Business Intelligence

- Business Intelligence and Reporting
  - Extract, Transform and Load (ETL)
    - Warehouse Builder

- Business Intelligence and Reporting
  - End User Query and Analysis
    - Discoverer Administrator

- Business Intelligence and Reporting
  - Enterprise Reporting
    - Reports Developer
Oracle Forms Developer Overview

Oracle Forms Developer:

• Is a productive development environment for Internet business applications

• Provides for:
  – Data entry
  – Queries
Oracle Forms Developer: Key Features

- Tools for rapid application development
- Application partitioning
- Flexible source control
- Extended scalability
- Object reuse
Summit Application

CUSTOMERS Table

CV_Customer Canvas

CUSTOMERS Block

Orders Form

ORDERS Table

CV_Order Canvas

ORDERS Block

ORDER_ITEMS Table

CV_Inventory Canvas

ORDER_ITEMS Block

INVENTORIES Block

INVENTORIES Table
Summary

In this lesson, you should have learned that:

- Grid computing makes computing power available without regard to its source
- Oracle 10g products provide the software to implement enterprise grid computing
- Oracle Application Server 10g provides services for building and deploying Web applications
- Oracle Developer Suite 10g includes components for application development and business intelligence
Summary

• Benefits of Oracle Forms Services include:
  – Optimized Web deployment of Forms applications
  – Rich Java UI without Java coding
  – Generic Java applet to deploy any Forms application
• Oracle Forms Services consists of the Forms client, the Forms Servlet, the Forms Listener Servlet, and the Forms Runtime Engine.
• Benefits of Oracle Forms Developer include rapid application development, application partitioning, flexible source control, extended scalability, and object reuse.
• The course application is a customer and order entry application for Summit Office Supply.
Running a Forms Developer Application
Objectives

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

• Start OC4J
• Describe the run-time environment
• Describe the elements in a running form
• Navigate a Forms application
• Describe the two main modes of operation
• Run a form in a Web browser
  – Retrieve both restricted and unrestricted data
  – Insert, update, and delete records
  – Display database errors
Testing a Form: OC4J Overview

Oracle Application Server Containers for J2EE (OC4J) is:

- Preferred to run Forms applications
- Included with Oracle Developer Suite to enable testing
Testing a Form: Starting OC4J

- On NT, run batch file to start OC4J: `startinst.bat`.
- OC4J starts in DOS window:
  - Minimize window
  - Closing window aborts OC4J
- Run batch file to stop OC4J: `stopinst.bat`.
Running a Form

Oracle Forms Services deployment:

Browser URL → Java Applet → Server
Running a Form: Browser

How do I access this application?

http://summit.com:8889/forms90/f90servlet?form=customers.fmx&userid=username/password@database&buffer_records=NO&debug_messages=NO&array=YES&query_only=NO
The Java Runtime Environment

- The Forms applet runs in a Java Runtime Environment (JRE) on the client machine.
- Types of JREs:
  - Java-enabled browser (native)
  - JInitiator (Oracle-supplied plug-in to Web browser) that provides:
    Incremental Java archive (JAR) file downloading
    JAR file caching
    Applet instance caching
    Automatic Java security configuration
Starting a Run-Time Session

Client Tier
- Web Browser
  - URL: http://summit.com:8889/forms90/f90

Middle Tier: Application Server
- Web Server
  - Static HTML files
  - OC4J or HTTP Server
  - Forms Services
    - Forms Servlet
    - Forms Listener Servlet
    - Forms Runtime Engine
- Forms Application Executables
  - FMX files
  - MMX files
  - PLX files
- DB
Starting a Run-Time Session

Client Tier

Web Browser

URL: http://summit.com:8889/forms90/f90

Applet started

Middle Tier: Application Server

Web Server

Static HTML files

OC4J or HTTP Server

Forms Services

Forms Servlet

Forms Listener Servlet

Forms Runtime Engine

Forms Application Executables

FMX files

MMX files

PLX files

DB
Starting a Run-Time Session

Client Tier
- Web Browser
- URL: http://summit.com:8889/forms90/f90

Middle Tier: Application Server
- Web Server
  - OC4J or HTTP Server
- Forms Services
  - Forms Servlet
  - Forms Listener Servlet
- Forms Runtime Engine
- Forms Application Executables
  - FMX files
  - MMX files
  - PLX files
- DB
The Forms Servlet

URL Pointing to Forms Servlet

http://summit.com:8889/forms90/f90servlet?form=customers.f

Desktop Client

Application Server

Web Server

Static HTML files
HTTP Server or OC4J

Forms Services

Forms Client
Base HTML files
Forms Servlet
Forms Listener Servlet
Forms Runtime Engine

Dynamic HTML file is created

URL PARAMETERS:
?form=customers.fmx
&userid=un/pw@db
&buffer_records=NO...

formsweb.cfg
basejini.html

Copyright © 2004, Oracle. All rights reserved.
The Forms Client

- Generic Java applet
- Responsibilities:
  - Displays the form’s user interface
  - Processes user interaction back to Forms Services
  - Processes incoming messages from Forms Services
The Forms Listener Servlet

Java Servlet that:

- Creates Forms Runtime process for each client
- Stops the Runtime process at session end
- Manages network communications between client and Forms Runtime process
- Communicates through Web server process
The Runtime Engine

The Forms Runtime Engine:
- Is a process (ifweb90) that runs on the Application Server
- Manages application logic and processing
- Communicates with the client browser and the database
What You See at Run Time
Identifying the Data Elements
Modes of Operation: Enter-Query Mode

Allows:
- Unrestricted and restricted queries
- Query/Where dialog box
- Record count by using Query > Count Hits

Does not allow:
- Navigation out of current data block
- Exiting run-time session
- Certain functions
- Insert, update, delete
Modes of Operation: Normal Mode

Allows:
- Unrestricted queries
- Insert, update, delete
- Commit (Save)
- Navigation out of current data block
- Exiting run-time session

Does Not Allow:
- Restricted queries
- Query/Where dialog box
Retrieving Data

Unrestricted query

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Restricted query

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Retrieving Restricted Data

- Do not use quotation marks with character and date items.
- The \texttt{LIKE} operator is implied with \% or \_.
- Use hash (\#) in front of SQL operators.
- Use \texttt{Query/Where} for complex query conditions.
- Use default date format (DD-MON-RR) in \texttt{Query/Where}.
- Use quotes around literals in \texttt{Query/Where}. 
Query/Where Dialog Box

• **Invoke by:**
  - Entering `:variable_name`
  - Executing query

• **Used to write:**
  - Complex search conditions
  - Queries with OR predicates
  - ORDER BY clause
Query/Where Dialog Box

:i = 104 OR :n BETWEEN 'F' and 'H'
Inserting, Updating, and Deleting

Form module

Memory
Deletes
Updates
Inserts
Making Changes Permanent

- Select Action > Save to make changes permanent.
- Select Action > Clear All to discard changes.
Displaying Errors

- Use to view Oracle errors
- Select Help > Display Error
- Shows Database Error window:
  - SQL statement
  - Error information
Summary

In this lesson, you should have learned that:

• You can use OC4J on the development machine to run a Forms application in a Web browser

• At run time:
  – The Forms Client is downloaded
  – The Forms Servlet creates a start HTML file
  – The Forms Listener Servlet starts a run-time session and maintains communication between it and the Forms Client
  – The Runtime Engine carries out application logic and maintains a database connection on behalf of the Forms Client
Summary

• When you run a form you see a Java applet running in a browser and displaying a menu, menu toolbar, console, and several kinds of data elements.

• Users navigate a Forms application using the menu, toolbar, the mouse, buttons, or function keys.

• The two main modes of operation are Normal mode and Enter-Query mode.

• Executing a query returns all records, unless the query is restricted by search criteria.
Summary

• In normal mode you can insert, update, and delete records and commit changes to the database.
• You display database errors from the menu (Help > Display Error)
Practice 2 Overview

This practice covers the following topics:

• Starting OC4J
• Running the course application:
  – Querying records
  – Inserting a record
  – Updating a record
  – Deleting a record
  – Displaying a database error
Working in the Forms Developer Environment
Objectives

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

- Describe Forms Builder components
- Navigate the Forms Builder interface
- Identify the main objects in a form module
- Customize the Forms Builder session
- Use the online help facilities
- Identify the main Forms executables
- Describe the Forms module types
- Set environment variables for design and run time
- Run a form from within Forms Builder
Forms Builder Key Features

With Forms Builder you can:

- Provide an interface for users to insert, update, delete, and query data
- Present data as text, image, and custom controls
- Control forms across several windows and database transactions
- Use integrated menus
- Send data to Oracle Reports
Forms Builder Components: Object Navigator

- Client-side and server-side objects displayed hierarchically
- Toolbar to create, delete or unload, expand or contract
- Icons to represent objects
- Fast search feature
Forms Builder Components:
Layout Editor
Getting Started in the Forms Builder Interface

- Start Forms Builder
- Connect to the database:
  - Menu: Select File > Connect
  - Toolbar: Click Connect
Forms Builder: Menu Structure
Blocks, Items, and Canvases

Canvas 1
- Block A
  - Items

Canvas 2
- Block B
  - Items

Block A and Block B are connected through Canvas 1 and Canvas 2.
Navigation in a Block

Canvas 1

Canvas 2
Data Blocks

Master Data Block
- A
- B
- C
- D

Detail Data Block
- X
- Y
- Z

Table
- A
- B
- C

Table
- X
- Y
- Z

Copyright © 2004, Oracle. All rights reserved.
Form Module Hierarchy
Customizing Your Forms Builder Session

![Preferences window]

- General tab: 
  - Save Before Building
  - Build Before Running

- Subclass tab: 
  - Suppress Hints
  - Use System Editor

- Wizards tab: 
  - Color Palette: [Browse...]

- Runtime tab: 
  - Color Mode: Read Only - Shared

- Printer:

[OK] [Cancel] [Help]
Saving Preferences

Existing Preferences File

Modified preferences

Updated, merged Preferences File

Motif:
prefs.ora
Windows:
cauprefs.ora
Using the Online Help System

Oracle Forms 10g is Oracle's award-winning Web Rapid Application Development product. It is a highly productive, end-to-end, PL/SQL based, development environment for building enterprise-class, database-centric Internet applications. Oracle Application Server 10g provides out-of-the-box optimized Web Application development tools.

Oracle Forms 10g is part of the Oracle Developer Suite 10g. It is a highly productive, end-to-end, PL/SQL based, development environment for building enterprise-class, database-centric Internet applications. Oracle Application Server 10g provides out-of-the-box optimized Web Application development tools.

Copyright © 2004, Oracle. All rights reserved.
Forms Developer Executables

Forms Builder

Forms Compiler

Definitions

Run files

Forms Services
Forms Developer Module Types

- Menus
- Forms
- Libraries

- Oracle Forms Developer components
- Data sources
- Database

- PL/SQL Library
- Object Library
Defining Forms Environment Variables for Run Time

Set on middle-tier machine (used at run time):

- **FORMS90_PATH**
- **ORACLE_PATH**
- **CLASSPATH**

For Forms deployment, the settings in the environment control file override system settings.
Defining Forms Environment Variables for Design Time

Set on Developer Suite machine (used by Forms Builder):

- FORMS90_BUILDER_CLASSPATH
- UI_ICON
- UI_ICON_EXTENSION
- FORMS90_HIDE_OBR_PARAMS

Windows: Modify in Registry (REGEDIT.EXE or REGEDT32.EXE)
Environment Variables and Y2K Compliance

- `NLS_DATE_FORMAT`
- `FORMS90_USER_DATE_FORMAT`
- `FORMS90_USER_DATETIME_FORMAT`
- `FORMS90_OUTPUT_DATETIME_FORMAT`
- `FORMS90_OUTPUT_DATETIME_FORMAT`
- `FORMS90_ERROR_DATE_FORMAT`
- `FORMS90_ERROR_DATETIME_FORMAT`
Forms Files to Define Run-Time Environment Variables

Environment control file:
- \forms90\server\default.env or
- Other file specified in Forms configuration file

Forms configuration file:
- \forms90\server\formsweb.cfg or other
- Used to specify:
  - System parameters, such as envFile and workingDirectory
  - User parameters, such as form and user ID
  - Settings for the Java client
  - Other settings
Testing a Form: The Run Form Button

• With the Run Form menu command or button, you can:
  – Run a form from Forms Builder
  – Test the form in a three-tier environment

• The Run Form command takes its settings from Preferences:
  – Edit > Preferences
  – Runtime tab
  – Set Web Browser Location if desired
  – Set Application Server URL to point to Forms Servlet:
    http://127.0.0.1:8889/forms90/f90servlet
Summary

In this lesson, you should have learned that:

- **Forms Builder includes the Object Navigator, the Property Palette, the Layout Editor, and the PL/SQL Editor**
- You can use the Object Navigator or the menu and its associated toolbar icons to navigate around the Forms Builder interface
- The main objects in a form module are blocks, items, and canvases
- The Edit > Preferences dialog box enables you to customize the Forms Builder session
Summary

- The Help menu enables you to use the online help facilities to look up topics, or you can invoke context-sensitive help.
- The Forms Developer executables are the Forms Builder and the Forms Compiler.
- The Forms Developer module types are forms, menus, and libraries.
- You can set environment variables in the Forms environment file (for run time) or on the development machine (for design time).
- You can use the Run Form button to run a form from within Forms Builder.
Practice 3 Overview

This practice covers the following topics:

• Becoming familiar with the Object Navigator
• Setting Forms Builder preferences
• Using the Layout Editor to modify the appearance of a form
• Setting run-time preferences to use OC4J to test applications
• Running a form application from within Forms Builder
• Setting environment variables so the Layout Editor in Forms Builder displays .gif images on iconic buttons
Creating a Basic Form Module
Objectives

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

• Create a form module
• Create a data block
• Save and compile a form module
• Identify Forms file formats and their characteristics
• Describe how to deploy a form module
• Explain how to create documentation for a Forms application
Creating a New Form Module

1. Create an empty module
2. Create data blocks and items
3. Apply standards
4. Fine-tune layout
5. Set object properties
6. Add code
7. Test form module
Creating a New Form Module

Choose one of the following methods:

• **Use wizards:**
  – Data Block Wizard
  – Layout Wizard

• **Build module manually**

• **Use template form**
Form Module Properties

Name property

Coordinate System property
Creating a New Data Block

- Use Forms Builder Wizards:
  - Data Block Wizard: Create a data block with associated data source quickly and easily
  - Layout Wizard: Lay out data block contents for visual presentation
- Create manually
Creating a New Data Block

Launch Data Block Wizard

Enter data source

Launch Layout Wizard

Lay out data block contents

New Data Block

Reentrant mode
Navigating the Wizards

Available only in reentrant mode

Exit without saving

Save without exiting

Next screen

Invoke online help

Previous screen

Save and exit

Tabbed Interface: Available only in reentrant mode

<table>
<thead>
<tr>
<th>Data Block Wizard</th>
<th>Layout Wizard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Table</td>
</tr>
<tr>
<td>Data Block</td>
<td>Items</td>
</tr>
</tbody>
</table>
Launching the Data Block Wizard

In Forms Builder, do one of the following:

- Select Tools > Data Block Wizard.
- Right-click and select Data Block Wizard.
- Select the Data Blocks node and click Create icon; select Use the Data Block Wizard option.
- Use the Data Block Wizard button on the toolbar in the Layout Editor.
Welcome to the Data Block Wizard!

This wizard allows you to quickly and easily create a data block based on a table, a view, or a set of stored procedures (an advanced option).

Click Next to begin creating your data block.

Display this page next time
Data Block Wizard: Table Page

Enter a table or view on which to base your data block. Then select the columns that should appear as items.

Table or view:
- customers

Available Columns:
- CUSTOMER_ID
- CUST_FIRST_NAME
- CUST_LAST_NAME
- CUST_ADDRESS
- PHONE_NUMBERS
- NLS_LANGUAGE
- NLS_TERRITORY
- CREDIT_LIMIT
- CUST_EMAIL
- ACCOUNT_MGR_ID

Database Items:

Options:
- Refresh
- Enforce data integrity
Data Block Wizard: Finish Page

Congratulations!
You have finished describing your data block.

Before you click Finish to create your new data block, click a radio button to tell the Data Block Wizard what you want to do next.

- [ ] Create the data block, then call the Layout Wizard
- [ ] Just create the data block

You can also use the Data Block Wizard to modify your existing data blocks.

Simply select the data block in the Object Navigator and click the Data Block Wizard toolbar button, or choose 'Data Block Wizard' from the 'Tools' menu.
Layout Wizard: Items Page

<table>
<thead>
<tr>
<th>Name</th>
<th>Prompt</th>
<th>Width</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTOMER_ID</td>
<td>Customer Id</td>
<td>36</td>
<td>14</td>
</tr>
<tr>
<td>CUST FIRST_NAME</td>
<td>Cust First Name</td>
<td>95</td>
<td>14</td>
</tr>
<tr>
<td>CUST LAST_NAME</td>
<td>Cust Last Name</td>
<td>95</td>
<td>14</td>
</tr>
<tr>
<td>CUST ADDRESS_STREET</td>
<td>Street Address</td>
<td>185</td>
<td>14</td>
</tr>
<tr>
<td>CUST ADDRESS_POST</td>
<td>Postal Code</td>
<td>50</td>
<td>14</td>
</tr>
<tr>
<td>CUST ADDRESS_CITY</td>
<td>City</td>
<td>140</td>
<td>14</td>
</tr>
<tr>
<td>CUST ADDRESS_STATE</td>
<td>State Province</td>
<td>50</td>
<td>14</td>
</tr>
<tr>
<td>CUST ADDRESS_COUNTY</td>
<td>Country Id</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>PHONE_NUMBERS</td>
<td>Phone Numbers</td>
<td>140</td>
<td>14</td>
</tr>
<tr>
<td>NLS_LANGUAGE</td>
<td>Nls Language</td>
<td>18</td>
<td>14</td>
</tr>
<tr>
<td>NLS_TERRITORY</td>
<td>Nls Territory</td>
<td>140</td>
<td>14</td>
</tr>
<tr>
<td>CREDIT_LIMIT</td>
<td>Credit Limit</td>
<td>54</td>
<td>14</td>
</tr>
<tr>
<td>CUST_EMAIL</td>
<td>Cust Email</td>
<td>140</td>
<td>14</td>
</tr>
<tr>
<td>ACCOUNT_MGR_ID</td>
<td>Account Mgr Id</td>
<td>36</td>
<td>14</td>
</tr>
</tbody>
</table>
Layout Wizard: Style Page

Select a layout style for your frame by clicking a radio button below.

- Form
- Tabular
Layout Wizard: Rows Page

Enter a title for the frame. Also be sure to specify the number of database records to be displayed in the frame, as well as the distance between each record.

To display a scrollbar in the frame that can be used to scroll through database records, check the 'Display Scrollbar' check box.

Frame Title: 

Records Displayed: 1

Distance Between Records: 0

Display Scrollbar

Finish
Data Block Functionality

Once you create a data block with the wizards, Forms Builder automatically creates:

- A form module with database functionality including query, insert, update, delete
- A frame object
- Items in the data block
- A prompt for each item
- Triggers needed to enforce database constraints if “Enforce data integrity” is checked
Template Forms
Saving a Form Module

To save the form module:

• Select File > Save
  OR
  Click the Save icon
• Enter a filename
• Navigate to desired location
• Click Save
Compiling a Form Module
## Module Types and Storage Formats

<table>
<thead>
<tr>
<th>Module Type</th>
<th>Form Module</th>
<th>Menu Module</th>
<th>PL/SQL Library</th>
<th>Object Library</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.fmb</td>
<td>.mmb</td>
<td>.pll</td>
<td>.olb</td>
</tr>
<tr>
<td></td>
<td>.fmt</td>
<td>.mmx</td>
<td>.plx</td>
<td>.olt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>.pld</td>
<td></td>
</tr>
</tbody>
</table>
Deploying a Form Module

1. Move module files to middle tier
2. Generate module on middle tier
3. Run in browser using Forms Services on middle tier
Text Files and Documentation

- Convert a binary file to a text file.
- Create an ASCII file for a form module.
Summary

In this lesson, you should have learned that:

• To create a form module, you create an empty module, then add data blocks and other elements
• You can create a data block manually or with the Data Block Wizard and Layout Wizard
• You can save and compile a form module using the File and Program menus or from the toolbar
• You can store form, menu, and library modules in text format (useful for documentation), in a portable binary format, or a non-portable binary executable format
• To deploy a form module, you move it to the application server machine and generate it
Practice 4 Overview

This practice covers the following topics:

- Creating a new form module
- Creating a data block by using Forms Builder wizards
- Saving and running the form module
Creating a Master-Detail Form
Objectives

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

- Create data blocks with relationships
- Modify a data block
- Modify the layout of a data block
- Run a master-detail form
Form Block Relationships

[Diagram showing Form Block Relationships with Master and Detail blocks connected by arrows indicating master-detail relationships.]
Form Block Relationships

Orders → Items

Customers → Orders

Orders → Account Rep

Items
Data Block Wizard: Master-Detail Page
Relation Object

- New relation object created in Object Navigator under master data block node
- Default name assigned: MasterDataBlock_DetailDataBlock
- Triggers and program units generated automatically
Creating a Relation Manually
Join Condition

- The join condition creates primary-foreign key link between blocks
- Define a join condition using:
  - Block and item names (not table and column names)
  - Do not precede names with colon
  - SQL equijoin syntax
Deletion Properties

- **Isolated**: Only master is deleted.
- **Cascading**: Master and all details are deleted.
- **Non-isolated**: If no detail record, master is deleted.
- **Non-isolated**: Master is not deleted if there are any detail records.

Master-Detail Records

- [Diagram showing deletion properties with green for deleted and blue for records.]

Copyright © 2004, Oracle. All rights reserved.
Modifying a Relation
Coordination Properties

- Default
- Deferred with auto query
- Deferred without auto query
Running a Master-Detail Form Module

- Automatic block linking for:
  - Querying
  - Inserting

- Default deletion rules: Cannot delete master record if detail records exist
Modifying the Structure of a Data Block

- **Reentrant Data Block Wizard:**
  1. Select frame or object in Layout Editor, or data block or frame in Object Navigator
  2. Select Tools > Data Block Wizard OR Right-click and select Data Block Wizard OR Click Data Block Wizard

- **Object Navigator:**
  - Create or delete items
  - Change item properties

- **Block Property Palette:** Change property values
Modifying the Layout of a Data Block

- **Reentrant Layout Wizard:**
  - Select frame in Object Navigator or Layout Editor
  - Select Tools > Layout Wizard
  - OR
  - Right-click and select Layout Wizard
  - Click Layout Wizard

- **Layout Editor:**
  - Select Tools > Layout Editor
  - Make changes manually

- **Frame Property Palette:** Change property values
Summary

In this lesson, you should have learned that:

• You can create data blocks with relationships by using the Data Block Wizard or by manually creating a Relation object
• When you run a master-detail form, block coordination is automatic depending on properties of the Relation object
• You can modify a data block manually or with the Data Block Wizard in reentrant mode
• You can modify the layout manually or with the Layout Wizard in reentrant mode
Practice 5 Overview

This practice covers the following topics:

• Creating a master-detail form module
• Modifying data block layout by using the Layout Wizard in reentrant mode
• Saving and running the form module
Working with Data Blocks and Frames
Objectives

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

• Identify the components of the Property Palette
• Manage object properties
• Create and use Visual Attributes
• Control the behavior and appearance of data blocks
• Control frame properties
• Create blocks that do not directly correspond to database tables
• Delete data blocks and their components
Managing Object Properties

- Reentrant Wizard
  - Data Block Wizard
  - Layout Wizard
- Layout Editor
- Property Palette
To display the Property Palette, use one of the following methods:

- Select Tools > Property Palette (or use the shortcut key).
- Double-click the object icon in the Object Navigator.
- Double-click the object in the Layout Editor.
- Right-click the object icon in the Object Navigator.
- Right-click the object in the Layout Editor.
Property Palette: Features

- **Find field**

- **Search backward**

- **Search forward**

- **Property name**

- **Property value**

- **Help:** Press [F1]

- **Toolbar**

- **Expand/collapse**

---

**Find field**

- **Find field**

**Search backward**

- **Search backward**

**Search forward**

- **Search forward**

---

**Property name**

- **Property name**

**Property value**

- **Property value**

---

**Help:**

- **Help:** Press [F1]

---

**Toolbar**

- **Toolbar**

**Expand/collapse**

- **Expand/collapse**

---

Copyright © 2004, Oracle. All rights reserved.
Property Controls

Text field

More button

LOV window

Pop-up list
Property Controls

- Changed
  - Visual Attribute Group: VISUAL_A
  - Prompt Visual Attribute Group: DEFAULT

- Default
  - Color
  - Foreground Color: magenta
  - Background Color: gray

- Overridden

- Inherited
Visual Attributes

A Visual Attribute is a named set of properties defining:

- Font
- Color
- Pattern
How to Use Visual Attributes

1. Create a Visual Attribute.
2. Set the Visual Attribute–related property of an object to the desired Visual Attribute.
3. Run the form to see the effect.
Font, Pattern, and Color Pickers
Controlling Data Block Behavior and Appearance

Data Block Property Groups:
- General
- Navigation
- Records
- Database
- Advanced Database
- Scrollbar
- Visual Attributes
- Color
- International
Navigation Properties

<table>
<thead>
<tr>
<th>Navigation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigation Style</td>
<td>Same Record</td>
</tr>
<tr>
<td>Previous Navigation Data Block</td>
<td>&lt;Null&gt;</td>
</tr>
<tr>
<td>Next Navigation Data Block</td>
<td>&lt;Null&gt;</td>
</tr>
</tbody>
</table>

Order

Previous Navigation Data Block

Item

Same Record

Next Record

Next Navigation Data Block
Records Properties

- Current Record Visual Attribute Group: HIGHLIGHTED
- Query Array Size: 0
- Number of Records Buffered: 0
- Number of Records Displayed: 6
- Query All Records: Yes

Item

Number of Records Displayed

Current Record
Records Properties

Vertical Record Orientation

Horizontal Record Orientation
Database Properties

Use properties in the Database group to control:
- Type of block—data or control block
- Query, insert, update, and delete operations on the data block
- Data block’s data source
- Query search criteria and default sort order
- Maximum query time
- Maximum number of records fetched

<table>
<thead>
<tr>
<th>Database</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Data Block</td>
<td>Yes</td>
</tr>
<tr>
<td>Enforce Primary Key</td>
<td>No</td>
</tr>
<tr>
<td>Query Allowed</td>
<td>Yes</td>
</tr>
<tr>
<td>Query Data Source Type</td>
<td>Table</td>
</tr>
<tr>
<td>Query Data Source Name</td>
<td>ORDERS</td>
</tr>
<tr>
<td>Query Data Source Columns</td>
<td></td>
</tr>
<tr>
<td>Query Data Source Arguments</td>
<td></td>
</tr>
<tr>
<td>Alias</td>
<td></td>
</tr>
<tr>
<td>Include REF Item</td>
<td>No</td>
</tr>
<tr>
<td>WHERE Clause</td>
<td></td>
</tr>
<tr>
<td>ORDER BY Clause</td>
<td>order_id</td>
</tr>
<tr>
<td>Optimizer Hint</td>
<td></td>
</tr>
<tr>
<td>Insert Allowed</td>
<td>Yes</td>
</tr>
<tr>
<td>Update Allowed</td>
<td>Yes</td>
</tr>
<tr>
<td>Locking Mode</td>
<td>Automatic</td>
</tr>
<tr>
<td>Delete Allowed</td>
<td>Yes</td>
</tr>
<tr>
<td>Key Mode</td>
<td>Automatic</td>
</tr>
<tr>
<td>Update Changed Columns Only</td>
<td>No</td>
</tr>
<tr>
<td>Enforce Column Security</td>
<td>No</td>
</tr>
<tr>
<td>Maximum Query Time</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Records Fetched</td>
<td>0</td>
</tr>
</tbody>
</table>
Database Properties

SELECT ....
WHERE Clause
[ORDER BY Clause]

Records fetched

Records buffered

Block display

Work file
Scroll Bar Properties

- Scroll Bar X/Y Position
- Scroll Bar Width
- Scroll Bar Height

Record

Copyright © 2004, Oracle. All rights reserved.
Controlling Frame Properties
Controlling Frame Properties

Form Layout Style

Order

Tabular Layout Style

Item

Distance between records
Displaying Multiple Property Palettes

Two Palettes for Two Items:

Two Palettes for One Item:
Setting Properties on Multiple Objects
Copying Properties

Copy

Paste

Name → ITEMS

<table>
<thead>
<tr>
<th>Source objects</th>
<th>Destination objects</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Query All Records</strong></td>
<td><strong>Yes</strong></td>
</tr>
<tr>
<td><strong>Query Allowed</strong></td>
<td><strong>Yes</strong></td>
</tr>
<tr>
<td><strong>Insert Allowed</strong></td>
<td><strong>Yes</strong></td>
</tr>
<tr>
<td><strong>Update Allowed</strong></td>
<td><strong>Yes</strong></td>
</tr>
<tr>
<td><strong>Delete Allowed</strong></td>
<td><strong>Yes</strong></td>
</tr>
</tbody>
</table>

| **Query All Records** | **Yes** |
| **Query Allowed** | **Yes** |
| **Insert Allowed** | **Yes** |
| **Update Allowed** | **No** |
| **Delete Allowed** | **Yes** |
Creating a Control Block

• Click the Data Blocks node
• Click the Create icon
  OR
Select Edit > Create.
• Select the “Build a new data block manually” option in the New Data Block dialog box.
Deleting a Data Block

- Select a data block for deletion
- Click the Delete icon
- OR
- Press [Delete]
- Click Yes in the alert box.
Summary

In this lesson, you should have learned that:

• **The Property Palette:**
  – Contains property names and values that enable you to modify Forms objects
  – Has tools to search for properties, inherit properties, expand or collapse property categories, and pop up lists and dialog boxes for various properties
  – Shows different icons for default, changed, inherited, and overridden properties

• **Block properties** control the behavior and appearance of data blocks

• **Frame properties** control how block items are arranged

• You can create blocks that do not directly correspond to database tables by choosing to create the block manually rather than using the Data Block Wizard

• Deleting a data block deletes all of its components
Practice 6 Overview

This practice covers the following topics:

• Creating a control block
• Creating a Visual Attribute
• Invoking context-sensitive help from the Property Palette
• Modifying data block properties
• Modifying frame properties
Working with Text Items
Objectives

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

• Describe text items
• Create a text item
• Modify the appearance of a text item
• Control the data in a text item
• Alter the navigational behavior of a text item
• Enhance the relationship between the text item and the database
• Add functionality to a text item
• Display helpful messages
Text Item Overview

What is a text item?

- Default item type
- Interface object for:
  - Querying
  - Inserting
  - Updating
  - Deleting
- Behavior defined in the Property Palette
Creating a Text Item

Canvas selection  Block selection
Modifying the Appearance of a Text Item: General and Physical Properties

![Property Palette](Image)

- **General**
  - Name: ORDER_ID
  - Item Type: Text Item

- **Functional**

- **Navigation**

- **Data**

- **Calculation**

- **Records**
  - Current Record Visual Attribute Group: <Null>
  - Distance Between Records: 0
  - Number of Items Displayed: 0

- **Physical**
  - Visible: Yes
  - Canvas: <Null>
  - Tab Page: <Null>
  - X Position: 0
  - Y Position: 0
  - Width: 63
  - Height: 14
  - Bevel: Lowered
  - Rendered: Yes
  - Show Vertical Scroll Bar: No
Modifying the Appearance of a Text Item: Records Properties

Orders

Items

Distance between records

Number of items displayed
Modifying the Appearance of a Text Item: Font and Color Properties

Use properties in the Font and Color groups to specify an item's:
- Visual attributes
- Font name, size, weight, style, color, and pattern
Modifying the Appearance of a Text Item: Prompts

- A prompt specifies the text label that is associated with an item.
- Several properties are available to arrange and manage prompts.
- Use prompt properties to change the appearance of an item prompt.
Associating Text with an Item Prompt

1. Canvas: CV_INVENTORY  Block: INVENTORIES
2. Stock Information
   - Product Id: PRODUCT
   - Warehouse Id: In Stock
3. Associate prompt
4. Replace the existing prompt with the new one?
   Yes No
Controlling the Data of a Text Item

Use properties in Data group to control the data:

- Type
- Length
- Format
- Value

US7ASCII
VARCHAR2 (5 CHAR)

JA16SJIS
VARCHAR2 (5 CHAR)

UTF8
VARCHAR2 (5 CHAR)
Controlling the Data of a Text Item: Format

Format masks:

- **Standard SQL formats**
  - Dates: FXDD-MON-YY
  - Numbers: L099G990D99

- **Nonstandard formats**
  Use double quotes for embedded characters ("099")099"-"0999

  Note: Allow for format mask’s embedded characters when defining Width property.
Controlling the Data of a Text Item: Values

Initial Values:
- Are used for every new record
- Can be overwritten
- Must be compatible with item’s data type
- Use:
  - Raw value
  - System variable
  - Global variable
  - Form parameter
  - Form item
  - Sequence
Controlling the Data of a Text Item:
Copy Value from Item

<table>
<thead>
<tr>
<th>Id</th>
<th>Last Name</th>
<th>First Name</th>
<th>Title</th>
<th>Dept Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Nagayama</td>
<td>Midori</td>
<td>VP, Sales</td>
<td>31</td>
</tr>
<tr>
<td>11</td>
<td>Magee</td>
<td>Colin</td>
<td>Sales Rep</td>
<td>31</td>
</tr>
</tbody>
</table>

Employee
Controlling the Data of a Text Item: Synchronize with Item
Alter the navigational behavior of text items.

- Established by order of entries in Object Navigator
- Alter by:
  - Keyboard Navigable
  - Previous Navigation Item
  - Next Navigation Item
Enhancing the Relationship Between Text Item and Database

Use properties in the Database group to control:

- Item’s data source—base table item or control item
- Query, insert, and update operations on an item
- Maximum query length
- Query case
### Adding Functionality to a Text Item

<table>
<thead>
<tr>
<th>Item Id</th>
<th>Product Id</th>
<th>Price</th>
<th>Quantity</th>
<th>Item Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>10011</td>
<td>135</td>
<td>500</td>
<td>67,500.00</td>
<td></td>
</tr>
<tr>
<td>10013</td>
<td>380</td>
<td>400</td>
<td>152,000.00</td>
<td></td>
</tr>
</tbody>
</table>

**Order Details**
- **Order Id**: 100
- **Payment Type**: CREDIT
- **Enabled**: No
- **Case Restriction**: Upper
- **Justification**: Start
- **Justification**: Right

**Item Details**

<table>
<thead>
<tr>
<th>Id</th>
<th>Product Id</th>
<th>Price</th>
<th>Quantity</th>
<th>Item Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10011</td>
<td>135</td>
<td>500</td>
<td>67,500.00</td>
</tr>
<tr>
<td>2</td>
<td>10013</td>
<td>380</td>
<td>400</td>
<td>152,000.00</td>
</tr>
</tbody>
</table>
Adding Functionality to a Text Item: Conceal Data Property
Adding Functionality to a Text Item: Keyboard Navigable and Enabled

- Set both properties to allow or disallow navigation and interaction with text item.
- When Enabled is set to Yes, Keyboard Navigable can be set to Yes or No.
- When Enabled is set to No, the item is always nonnavigable.
Adding Functionality to a Text Item: Multi-line Text Items

Total text = Maximum length

Text
Text
Text
Text

Width

Height
Displaying Helpful Messages: Help Properties
Summary

In this lesson, you should have learned that:

- Text items are interface objects that usually correspond to database columns
- You can create a text item with:
  - The Text Item tool in the Layout Editor
  - The Create icon in the Object Navigator
  - The Data Block Wizard
Summary

- You can modify a text item in its Property Palette:
  - General, Records, and Physical properties control the appearance of the text item
  - Data properties control the length, datatype, format, and other aspects of the data.
  - Navigation properties control how to navigate to and from a text item.
  - Database properties specify the relationship between the text item and its corresponding database column.
  - Functional properties control how the text item functions.
  - Help properties specify the display of helpful messages.
Practice 7 Overview

This practice covers the following topics:

- Deleting text items
- Modifying text item properties
- Creating text items
Creating LOVs and Editors
Objectives

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

• Describe LOVs and editors
• Design, create, and associate LOVs with text items in a form module
• Create editors and associate them with text items in a form module
Overview of LOVs and Editors

Editor

Text item

Text item

Text item

LOV

Supporting data record group
Overview of LOVs and Editors

- **LOVs**
  - List of values for text items
  - Dynamic or static list
  - Independent of single text items
  - Flexible and efficient

- **Editors**
  - Override default editor
  - Used for special requirements such as larger editing window, position, color, and title
LOVs and Record Groups

- Text item
- LOV
- Record group based on static data
- OR
- Text item
- LOV
- Record group
- OR
- SQL
- Query-based record group
- Database
LOVs and Record Groups

Sales Representatives LOV

Sales Rep record group

Sales Representatives

<table>
<thead>
<tr>
<th>Employee_id</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>174</td>
<td>Ellen Abel</td>
</tr>
<tr>
<td>166</td>
<td>Sundar Ande</td>
</tr>
<tr>
<td>167</td>
<td>Amit Banda</td>
</tr>
<tr>
<td>172</td>
<td>Elizabeth Bates</td>
</tr>
<tr>
<td>151</td>
<td>David Bernstein</td>
</tr>
<tr>
<td>169</td>
<td>Harrison Bloom</td>
</tr>
</tbody>
</table>

SELECT employee_id, first_name || ' ' || last_name NAME, phone_number
FROM employees
WHERE job_id = 'SA_REP'
ORDER BY last_name

EMPLOYEES table
Creating an LOV Manually

1. Define the Record Group.

2. Set the Column Mapping Properties.

3. Map the LOV Column.

4. Add the List of Values (LOV).
Creating an LOV with the LOV Wizard: SQL Query Page
Creating an LOV with the LOV Wizard: Column Selection Page
Creating an LOV with the LOV Wizard: Column Properties Page
Creating an LOV with the LOV Wizard: Display Page

What title would you like to display in your LOV window?

Title: Sales Representatives

What size would you like your LOV to be? The units for the LOV size and position are Points.

Width: 180  Height: 135

Do you want Forms Runtime to position your LOV?

- Yes, let Forms position my LOV automatically
- No, I want to position it manually

Left: 0  Top: 0

Copyright © 2004, Oracle. All rights reserved.
Creating an LOV with the LOV Wizard: Advanced Properties Page

Do you want to modify the advanced properties that affect the behavior of your LOV?

If you are not familiar with their usage, it is recommended that you accept the defaults as they appear below.

- Retrieve **20** rows at a time.
- **✓** Refresh record group data before displaying LOV
- **☐** Let the user filter records before displaying them
Creating an LOV with the LOV Wizard: Assign to Item Page
LOV Properties

(X,Y) Width

LOV Height

Column Mapping

Filter Before Display

? Return Items

Automatic Select

Automatic Skip
Setting LOV Properties

- Automatic Column Width
- Automatic Position
- Column Mapping
- Return Items
LOVs: Column Mapping

orders.sales_rep_id | orders.sales_rep_name | orders.salesrep_phone

Hidden column

Phone number 1-415-555-6281

Copyright © 2004, Oracle. All rights reserved.
Defining an Editor
Setting Editor Properties

![Property Palette](image)

- **General**
  - Name: MY_EDITOR
  - Subclass Information
  - Comments

- **Functional**
  - Title: Product Description Editor
  - Bottom Title
  - Wrap Style: Word

- **Physical**
  - X Position: 0
  - Y Position: 0
  - Width: 180
  - Height: 135
  - Show Vertical Scroll Bar: No

- **Visual Attributes**
  - Color
  - Font
Associating an Editor with a Text Item

- Associate one of two types of editors with a text item.
- Set text item’s Editor property to one of the following:
  - Null (default Forms Builder editor)
  - Editor name (customized editor)
Summary

In this lesson, you should have learned that:

• An LOV is a scrollable pop-up window that enables a user to pick the value of an item from a multicolumn dynamic list
• The easiest way to design, create, and associate LOVs with text items is to use the LOV Wizard
• An Editor is a separate window that enables the user to view multiple lines of a text item simultaneously, search and replace text in it, and modify the text
• You create editors in the Object Navigator and associate them with text items in the item's Property Palette
Practice 8 Overview

This practice covers the following topics:

- Creating an LOV and attaching the LOV to a text item
- Creating an Editor and attaching it to a text item
Creating Additional Input Items
Objectives

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

• Identify the item types that allow input
• Create a check box
• Create a list item
• Create a radio group
Input Items Overview

What are input items?

• Item types that accept user input include:
  – Check boxes
  – List items
  – Radio groups

• Input items enable insert, update, delete, and query.
Check Boxes Overview

What Are Check Boxes?

- Two-state interface object:
  - Checked
  - Unchecked
- Not limited to two values
Creating a Check Box

- Convert an existing item.
- Use the Check Box tool in the Layout Editor.
- Use the Create icon in the Object Navigator.
Converting an Existing Item into a Check Box

Convert text item to check box
Creating a Check Box in the Layout Editor

Use check box tool in Layout Editor
Setting Check Box Properties

- Data Type
- Label
- Access Key
- Value When Checked
- Value When Unchecked
- Check Box Mapping of Other Values
- Mouse Navigate
Check Box Mapping of Other Values

Order_Mode

Y -> Checked
Y
N
Null
A

Unchecked

Y
N

Check Box Mapping of Other Values

Unchecked
List Items Overview

What Are List Items?

- Set of mutually exclusive choices, each representing a different value
- Three list styles available:
  - Space-saving alternative to a radio group
  - Smaller-scale alternative to an LOV
Creating a List Item

- Convert an existing item.
- Use the List Item tool in the Layout Editor.
- Use the Create icon in the Object Navigator.
Converting an Existing Item into a List Item
Creating a List Item in the Layout Editor

Use list item tool
in Layout Editor
Setting List Item Properties

- Elements in List:
  - List elements
  - List item value
- List Style
- Mapping of Other Values
- Mouse Navigate
List Item Mapping of Other Values

Values for Forms Items

Order_Status

10
4
2
0
12

Displayed Values

List Elements

CREDIT order paid
New CREDIT order
CASH backorder
New CASH order
Unknown

Mapping of Other Values = 11 (Unknown)
Radio Groups Overview

What are radio groups?

• Set of mutually exclusive radio buttons, each representing a value

• Use:
  – To display two or more static choices
  – As an alternative to a list item
  – As an alternative to a check box
Creating a Radio Group

- Convert an existing item.
- Create a new radio button in the Layout Editor.
- Use the Create icon in the Object Navigator.
Converting Existing Item to Radio Group

Change Item Type and set other properties

Create radio buttons for the radio group
Creating Radio Group in Layout Editor
Setting Radio Properties

Radio group:

Radio button:
Radio Group Mapping of Other Values

Values for Forms Items
- Credit_Limit
- 500
- 2000
- Null
- 5000

Displayed Values

List Elements
- LOW_BUTTON
  - Low
- MEDIUM_BUTTON
  - Medium
- HIGH_BUTTON
  - High

Mapping of Other Values
2000
Summary

In this lesson, you should have learned that:

- Check boxes, list items, and radio groups are the item types that allow input.
- You create these items by:
  - Changing the item type of an existing item
  - Using the appropriate tool in the Layout Editor
- You can use a check box for items that have only two possible states.
- You can use a list item to enable users to pick from a list of mutually exclusive choices.
- You can use a radio group for two or three mutually exclusive alternatives.
Practice 9 Overview

This practice covers the following topics:

- Converting a text item into a list item
- Converting a text item into a check box item
- Converting a text item into a radio group
- Adding radio buttons to the radio group
Creating Noninput Items
Objectives

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

• Identify item types that do not allow input
• Create a display item
• Create an image item
• Create a button
• Create a calculated item
• Create a hierarchical tree item
• Create a bean area item
Noninput Items Overview

Item types that do not accept direct user input include:

- Display items
- Image items
- Buttons
- Calculated items
- Hierarchical tree items
- Bean area items
Display Items

Display items:
• Are similar to text items.
• Cannot:
  – Be edited
  – Be queried
  – Be navigated to
  – Accept user input
• Can display:
  – Nonbase table information
  – Derived values
Creating a Display Item
Image Items

Use image items to display images:
- From file system—supported file type
- From database—LONG RAW column or a BLOB column
Image File Formats

Image files
- JPEG
- CALS
- TIFF
- GIF
- JFIF
- BMP
- PICT
- RAS
- TPIC

Read

Image item

Write

Image files
- JPEG
- CALS
- GIF
- TIFF
- JFIF
- BMP
- PICT
- RAS
- TPIC

Copyright © 2004, Oracle. All rights reserved.
Creating an Image Item
Setting Image-Specific Item Properties

- Image Format
- Image Depth
- Compression Quality
- Display Quality
- Sizing Style
- Show Horizontal Scroll Bar
- Show Vertical Scroll Bar

## Functional
- Enabled: Yes
- Image Format: TIFF
- Image Depth: Original
- Compression Quality: Minimum
- Display Quality: High
- Show Palette: No
- Sizing Style: Adjust
- Popup Menu: <null>

## Navigation

## Data

## Records

## Database

## Physical
- Visible: Yes
- Canvas: CV_ORDER
- Tab Page: <null>
- X Position: 335
- Y Position: 32
- Width: 80
- Height: 65
- Bevel: None
- Show Horizontal Scroll Bar: No
- Show Vertical Scroll Bar: No
Push Buttons

Push buttons:
- Cannot display or represent data
- Are used to initiate an action
- Display as:
  - Text button
  - Iconic
Push Button Actions

Use buttons to:
• Move input focus
• Display an LOV
• Invoke an editor
• Invoke another window
• Commit data
• Issue a query
• Perform calculations
Creating a Push Button
Setting Push Button Properties

- Label
- Iconic
- Icon Filename
- Default Button
- Mouse Navigate
- Tooltip
- Tooltip Visual Attribute Group
Calculated Items

What are calculated items?

- They accept item values that are based on calculations.
- They are read-only.
- They can be expressed as:
  - Formula
  - Summary
Creating a Calculated Item by Setting Properties

- **Formula**
  - A calculated item value is the result of a horizontal calculation.
  - It involves bind variables.

- **Summary**
  - A calculated item value is a vertical calculation.
  - A summary is performed on values of a single item over all rows in a block.

```
<table>
<thead>
<tr>
<th>Calculation Mode</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formula</td>
<td>Sum:sum</td>
</tr>
<tr>
<td>Summarized Block</td>
<td>&lt;Null&gt;</td>
</tr>
<tr>
<td>Summarized Item</td>
<td>ITEM_TOTAL</td>
</tr>
</tbody>
</table>
```
Setting Item Properties for the Calculated Item

- **Formula**
  - Calculation Mode
  - Formula

- **Summary**
  - Calculation Mode
  - Summary Function
  - Summarized Block
  - Summarized Item
## Summary Functions

- **AVG**
- **COUNT**
- **MAX**
- **MIN**
- **STDDEV**
- **SUM**
- **VARIANCE**
Calculated Item Based on a Formula

\[ \text{NVL((:order_items.unit_price \times :order_items.quantity), 0)} \]
Rules for Calculated Item Formulas

Create calculated item formulas according to the following rules:

• A formula item must not invoke restricted built-ins.
• A formula item cannot execute any DML statements.
• Do not terminate a PL/SQL expression with a semicolon.
• Do not enter a complete PL/SQL statement in assignment expressions.
### Calculated Item Based on a Summary

#### Orders

<table>
<thead>
<tr>
<th>Item#</th>
<th>Prod Id</th>
<th>Description</th>
<th>Unit Price</th>
<th>Quantity</th>
<th>Item Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>200</td>
<td>5</td>
<td>1,000</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>120</td>
<td>4</td>
<td>480</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>50</td>
<td>9</td>
<td>450</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>25</td>
<td>3</td>
<td>75</td>
</tr>
</tbody>
</table>

#### Items

#### Summary item

<table>
<thead>
<tr>
<th>Item#</th>
<th>Prod Id</th>
<th>Description</th>
<th>Unit Price</th>
<th>Quantity</th>
<th>Item Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>200</td>
<td>5</td>
<td>1,000</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>120</td>
<td>4</td>
<td>480</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>50</td>
<td>9</td>
<td>450</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>25</td>
<td>3</td>
<td>75</td>
</tr>
</tbody>
</table>

Order Total: 2,005
Rules for Summary Items

• Summary item must reside in:
  – The same block as the summarized item
  – A control block with Single Record property set to Yes

• Summarized item must reside in:
  – A data block with Query All Records property or Precompute Summaries property set to Yes
  – A control block

• Datatype of summary item must be Number, unless using MAX or MIN
Creating a Hierarchical Tree Item
Setting Hierarchical Tree Item Properties

- Allow empty branches
- Multi selection
- Show lines
- Show symbols
- Record group
- Data query
Bean Area Items

The Bean Area item enables you to:

• Add a JavaBean to a form
• Extend Forms functionality
• Interact with client machine
• Reduce network traffic
Creating a Bean Area Item

Create bean area in Layout Editor

Convert existing item to bean area

Property Palette

- General
  - Name: TEXT_ITEM5
  - Item Type: Text Item
- Subclass Information
  - Subclass Information: Bean Area
- Comments
- Help Book Topic
- Functional
  - Enabled
  - Justification

Type of item: Bean Area
Setting Bean Area Item Properties
The JavaBean at Run Time
Summary

In this lesson, you should have learned that:

• The following item types do not allow input:
  – Display items
  – Image items
  – Push buttons
  – Calculated items
  – Hierarchical tree items
  – Bean area items

• You create noninput items by:
  – Changing the type of an existing item and setting certain properties
  – Using the appropriate tool in the Layout Editor
Summary

• You can use:
  – A display item to show nonbase table information
  – An image item to display an image
  – A push button to initiate action
  – A calculated item to display the results of a formula or a summary function of another item
  – A hierarchical tree item to display related data in a hierarchical fashion
  – A bean area item to execute client-side Java code
Practice 10 Overview

This practice covers the following topics:

• Creating display items
• Creating an image item
• Creating iconic buttons
• Creating calculated items:
  – Formula
  – Summary
• Creating a bean area item
Creating Windows and Content Canvases
Objectives

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

• Describe the relationship between windows and content canvases
• Create windows and content canvases
• Display a form module in multiple windows
• Display a form module on multiple layouts
Windows and Canvases

- Window: Container for Forms Builder visual objects
- Canvas: Surface on which you “paint” visual objects
- To see a canvas and its objects, display the canvas in a window.
Window, Canvas, and Viewport
The Content Canvas

- “Base” canvas
- View occupies entire window
- Default canvas type
- Each window should have at least one content canvas
Relationship Between Windows and Content Canvases

Canvas 1

Window

Canvas 2

Canvas 3
The Default Window

WINDOW1:

- Created by default with each new form module
- Is modeless
- You can delete, rename, or change its attributes
Displaying a Form Module in Multiple Windows

- Use additional windows to:
  - Display two or more content canvases at once
  - Switch between canvases without replacing the initial one
  - Modularize form contents
  - Take advantage of the window manager

- Two types of windows
  - Modal
  - Modeless
Creating a New Window

Object Navigator: Click Create with Windows node selected

Property Palette: Set properties
Setting Window Properties
GUI Hints

• GUI hints are recommendations to the window manager about window appearance and functionality.
• If the window manager supports a specific GUI Hint and its property is set to Yes, it will be used.
• Functional properties for GUI Hints:
  – Close Allowed
  – Move Allowed
  – Resize Allowed
  – Maximize Allowed
  – Minimize Allowed
  – Inherit Menu
Displaying a Form Module on Multiple Layouts

PROPERTIES:

Canvas CV_ORDER
Window: WIN_ORDERS

Canvas CV_INVENTORY
Window: WIN_INVENTORY
Creating a New Content Canvas

- Implicitly:

- Explicitly:
Setting Content Canvas Properties

Viewport X/Y Position on Canvas

Viewport
Canvas

Stock Information

Viewport X Position on Canvas: 10
Viewport Y Position on Canvas: 0
Width: 320
Height: 175
Summary

In this lesson, you should have learned that:

• Windows can display multiple content canvases, but can display only one canvas at a time
• Content canvases are displayed only in the window to which they are assigned
• You must assign at least one content canvas to each window in your application
• You create windows in the Object Navigator; one is created by default with each new module
• You create canvases in the Object Navigator, by using the Layout Wizard, or by invoking the Layout Editor in a module without a canvas
• You can display a multiple layouts by assigning canvases to different windows.
Practice 11 Overview

This practice covers the following topics:
• Changing a window size, position, name, and title
• Creating a new window
• Displaying data block contents in the new window
Working with Other Canvas Types
Objectives

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

• Describe the different types of canvases and their relationships to each other
• Identify the appropriate canvas type for different scenarios
• Create an overlay effect by using stacked canvases
• Create a toolbar
• Create a tabbed interface
Overview of Canvas Types

Content canvas

Stacked canvas

Horizontal toolbar

Vertical toolbar

Tab

Tab page
The Stacked Canvas

- Displayed on top of a content canvas
- Shares a window with a content canvas
- Size:
  - Usually smaller than the content canvas in the same window
  - Determined by viewport size
- Created in:
  - Layout Editor
  - Object Navigator
The Stacked Canvas

Help Information for Orders form:

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

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

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

You can use the Record menu for inserting and deleting records.
Creating a Stacked Canvas
## Setting Stacked Canvas Properties

![Property Palette](image)

### Functional
- **Raise on Entry**: No
- **Popup Menu**: Null

### Viewport
- **Viewport X Position**: 71
- **Viewport Y Position**: 30
- **Viewport Width**: 243
- **Viewport Height**: 190

### Physical
- **Visible**: No
- **Window**: WIN_ORDER
- **Viewport X Position on Canvas**: 0
- **Viewport Y Position on Canvas**: 0
- **Width**: 360
- **Height**: 324
- **Bevel**: Lowered
- **Show Horizontal Scroll Bar**: No
- **Show Vertical Scroll Bar**: No

### Visual Attributes
- **Show Vertical Scroll Bar**: No
The Toolbar Canvas

• Special type of canvas for tool items
• Two types:
  – Vertical toolbar
  – Horizontal toolbar
• Provide:
  – Standard look and feel
  – Alternative to menu or function key operation
The MDI Toolbar

Runtime parameter:
otherparams=useSDI=no

Window property:

Form property:
Creating a Toolbar Canvas

1. Create:
   - Click Create in Object Navigator
   - Change Canvas Type
   - Set other properties as required
2. Add functionality
3. Resize the canvas (not the view)
4. Assign to window and/or form
Setting Toolbar Properties

- **Canvas properties:**
  - Canvas Type
  - Window
  - Width or Height

- **Window properties:**
  - Horizontal Toolbar Canvas
  - Vertical Toolbar Canvas

- **Form Module properties:**
  - Form Horizontal Toolbar Canvas
  - Form Vertical Toolbar Canvas
The Tab Canvas

- Enables you to organize and display related information on separate tabs
- Consists of one or more tab pages
- Provides easy access to data
Creating a Tab Canvas

• Create in:
  – Object Navigator
  – Layout Editor
• Define tab pages
• Place items on tab pages
Creating a Tab Canvas in the Object Navigator

Create new Canvas  Set Canvas Type  Create Tab Pages
Setting Tab Canvas, Tab Page, and Item Properties
Placing Items on a Tab Canvas

- Place items on each tab page for user interaction.
- Set the item properties:
  - Canvas
  - Tab Page
Summary

In this lesson, you should have learned:

• Canvas types other than content canvases:
  – Stacked: Overlays and shares window with content canvas; use to create cascading or revealing effect within a single window, display additional information, display or hide information conditionally, or display context-sensitive help
  – Toolbar: Area that displays at the top or to the left of a content canvas; use to hold buttons and other frequently used GUI elements with a standard look and feel across canvases displayed in the same window
  – Tab: Has multiple pages where you navigate using tabs; use to organize and display related information on different tabs
Summary

- You can create these in Object Navigator and change the canvas type, then set properties.
- You can create stacked or tab canvases with the appropriate tool in the Layout Editor.
- You can attach a Toolbar canvas to single window, or to entire form if using MDI.
- After creating a tab canvas, create tab pages and place related items on them.
Practice 12 Overview

This practice covers the following topics:

- Creating a toolbar canvas
- Creating a stacked canvas
- Creating a tab canvas
- Adding tab pages to the tab canvas
Introduction to Triggers
Objectives

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

• Define triggers
• Identify the different trigger categories
• Plan the type and scope of triggers in a form
• Describe the properties that affect the behavior of a trigger
Which trigger would you use to perform complex calculations after a user enters data into an item?
Grouping Triggers into Categories

Triggers may be grouped into functional categories:

- Block processing triggers
- Interface event triggers
- Master-detail triggers
- Message handling triggers
- Navigational triggers
- Query-time triggers
- Transactional triggers
- Validation triggers

Triggers may be grouped into categories based on name:

- When-Event triggers
- On-Event triggers
- Pre-Event triggers
- Post-Event triggers
- Key triggers
Defining Trigger Components

What event?  What action?  What level?

Type  Code  Scope
Trigger Type

- Pre-
- Post-
- When-
- On-
- Key-
- User-named

What event?
## Trigger Type

### Forms Builder Trigger Types

<table>
<thead>
<tr>
<th>Trigger Type</th>
<th>Trigger Type</th>
<th>Trigger Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>[User-named]</td>
<td>KEY-NXTKEY</td>
<td>ON-UPDATE</td>
</tr>
<tr>
<td>KEY-CLRLBLK</td>
<td>KEY-NXTREC</td>
<td>POST-BLOCK</td>
</tr>
<tr>
<td>KEY-CLRFRM</td>
<td>KEY-NXTSET</td>
<td>POST-CHANGE</td>
</tr>
<tr>
<td>KEY-CLREC</td>
<td>KEY-OTHERS</td>
<td>POST-DATABASE-COMMIT</td>
</tr>
<tr>
<td>KEY-COMMIT</td>
<td>KEY-PREV-ITEM</td>
<td>POST-DELETE</td>
</tr>
<tr>
<td>KEY-CQUERY</td>
<td>KEY-PRINT</td>
<td>POST-FORM</td>
</tr>
<tr>
<td>KEY-CREREC</td>
<td>KEY-PRVBLK</td>
<td>POST-HEADERS-COMMIT</td>
</tr>
<tr>
<td>KEY-DELREC</td>
<td>KEY-PRVREC</td>
<td>POST-INSERT</td>
</tr>
<tr>
<td>KEY-DOWN</td>
<td>KEY-SCRDOWN</td>
<td>POST-LOGON</td>
</tr>
<tr>
<td>KEY-DUP-ITEM</td>
<td>KEY-SCRUP</td>
<td>POST-LOGOUT</td>
</tr>
<tr>
<td>KEY-DUPREC</td>
<td>KEY-UP</td>
<td>POST-QUERY</td>
</tr>
<tr>
<td>KEY-EDIT</td>
<td>KEY-UPDREC</td>
<td>POST-RECORD</td>
</tr>
<tr>
<td>KEY-ENTER</td>
<td>ON-CHECK-DELETE-MASTER</td>
<td>POST-SELECT</td>
</tr>
<tr>
<td>KEY-ENTQRY</td>
<td>ON-CHECK-UNIQUE</td>
<td>POST-TEXT-ITEM</td>
</tr>
<tr>
<td>KEY-EXQRY</td>
<td>ON-CLOSE</td>
<td>POST-UPDATE</td>
</tr>
<tr>
<td>KEY-EXIT</td>
<td>ON-COLUMN-SECURITY</td>
<td>PRE-BLOCK</td>
</tr>
<tr>
<td>KEY-F0</td>
<td>ON-COMMIT</td>
<td>PRE-COMMIT</td>
</tr>
<tr>
<td>KEY-F1</td>
<td>ON-COUNT</td>
<td>PRE-DELETE</td>
</tr>
<tr>
<td>KEY-F2</td>
<td>ON-DELETE</td>
<td>PRE-INSERT</td>
</tr>
<tr>
<td>KEY-F3</td>
<td>ON-FETCH</td>
<td>PRE-LOGON</td>
</tr>
<tr>
<td>KEY-F4</td>
<td>ON-INSERT</td>
<td>PRE-LOGOUT</td>
</tr>
<tr>
<td>KEY-F5</td>
<td>ON-LOGON</td>
<td>PRE-POPUP-MENU</td>
</tr>
<tr>
<td>KEY-F6</td>
<td>ON-LOGOUT</td>
<td>PRE-QUERY</td>
</tr>
<tr>
<td>KEY-F7</td>
<td>ON-MESSAGE</td>
<td>PRE-RECORD</td>
</tr>
<tr>
<td>KEY-F8</td>
<td>ON-POPULATE-DETAILS</td>
<td>PRE-SELECT</td>
</tr>
<tr>
<td>KEY-F9</td>
<td>ON-ROLLBACK</td>
<td>PRE-TEXT-ITEM</td>
</tr>
<tr>
<td>KEY-HELP</td>
<td>ON-SAVEPOINT</td>
<td>PRE-UPDATE</td>
</tr>
<tr>
<td>KEY-LISTVAL</td>
<td>ON-SELECT</td>
<td>WHEN-BUTTON-PRESSED</td>
</tr>
<tr>
<td>KEY-MENU</td>
<td>ON-SEQUENCE-NUMBER</td>
<td>WHEN-CLEAR-BLOCK</td>
</tr>
<tr>
<td>KEY-NXT-ITEM</td>
<td>ON-SEQUENCE-NUMBER</td>
<td>WHEN-CREATE-RECORD</td>
</tr>
<tr>
<td>KEY-NXTBLK</td>
<td>ON-SEQUENCE-NUMBER</td>
<td>WHEN-CUSTOM-ITEM-EVENT</td>
</tr>
</tbody>
</table>

---

Copyright © 2004, Oracle. All rights reserved.
Trigger Code

What action?

- Statements
- PL/SQL
- User subprograms
- Built-in subprograms
Trigger Scope

Levels
- Form
- Block
- Item

What level?
Trigger Scope
Specifying Execution Hierarchy

Form level
- On-Message: EH = Before

Block level
- On-Message: EH = After
- On-Message: EH = Override

Item level
- On-Message: EH = After
Summary

In this lesson, you should have learned that:

- Triggers are event-activated program units
- You can categorize triggers based on function or name to help you understand how they work
- Trigger components are:
  - Type: Defines the event that fires the trigger
  - Code: The actions a trigger performs
  - Scope: Specifies the level (form, block, or item) at which the trigger is defined
- The Execution Hierarchy trigger property alters the firing sequence of a trigger
Producing Triggers
Objectives

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

• Write trigger code
• Explain the use of built-in subprograms in Forms applications
• Describe the When-Button-Pressed trigger
• Describe the When-Window-Closed trigger
Creating Triggers in Forms Builder

To produce a trigger:
1. Select a scope in the Object Navigator.
2. Create a trigger and select a name from the Trigger LOV, or use the SmartTriggers menu option.
3. Define code in the PL/SQL Editor.
4. Compile.
Creating a Trigger

Step One:
Select Trigger Scope.

- Form level
- Block level
- Item level
Creating a Trigger

Step Two: Invoke the Trigger LOV.
Creating a Trigger

Step Three:
Use the PL/SQL Editor to define the trigger code.

Step Four:
Compile.
Setting Trigger Properties
PL/SQL Editor Features

1. **Split view**
   - Example code:
     ```plsql
     DECLARE
     old_sal NUMBER;
     new_sal NUMBER;
     BEGIN
     IF NVL(:control, 0) THEN
       old_sal := :employees.salary;
       new_sal := round(old_sal * :employees.salary); END IF;
     ELSE
       message('Please enter a raise'); RAISE form_trigger_error;
     END;
     END;
     ```

2. **Split Bars**
   - Example code:
     ```plsql
     DECLARE
     old_sal NUMBER;
     new_sal NUMBER;
     BEGIN
     IF NVL(:control, 0) THEN
       old_sal := :employees.salary;
       new_sal := round(old_sal * :employees.salary);
     ELSE
       message('Please enter a raise'); RAISE form_trigger_fail;
     END IF;
     END;
     ```

---

Copyright © 2004, Oracle. All rights reserved.
PL/SQL Editor Features

The Syntax Palette

PROCEDURE my_proc
BEGIN
END;

BEGIN
  statements;
  EXCEPTION
  exception_handler;
END;
The Database Trigger Editor
Writing Trigger Code

BEGIN

END;

A PL/SQL Block

DECLARE

--Declarative Statements [Optional]

BEGIN

-- Executable Statements [Required]

EXCEPTION

-- Exception Handlers [Optional]

END;
Using Variables in Triggers

- PL/SQL variables must be declared in a trigger or defined in a package

```
DECLARE
  my_var VARCHAR2(30);
BEGIN
  my_var := 'This is a PL/SQL variable';
  :block_name.item_name := 'This is a Forms Builder variable';
END;
```

- Forms Builder variables
  - Are not formally declared in PL/SQL
  - Need a colon (:) prefix in reference
# Forms Builder Variables

<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Purpose</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>Presentation and user interaction</td>
<td><code>:block_name.item_name</code></td>
</tr>
<tr>
<td>Global variable</td>
<td>Session-wide character variable</td>
<td><code>:GLOBAL.variable_name</code></td>
</tr>
<tr>
<td>System variables</td>
<td>Form status and control</td>
<td><code>:SYSTEM.variable_name</code></td>
</tr>
<tr>
<td>Parameters</td>
<td>Passing values in and out of module</td>
<td><code>:PARAMETER.name</code></td>
</tr>
</tbody>
</table>
Adding Functionality with Built-In Subprograms

Built-ins belong to either:

- The Standard Extensions package where no prefix is required
- Another Forms Builder package where a prefix is required
Limits of Use

- Unrestricted built-ins are allowed in any trigger or subprogram.
- Restricted built-ins are allowed only in certain triggers and subprograms called from such triggers.
- Consult the Help system.

Compiles:

Run-time error when trigger fires:

```
FRM-40737: Illegal restricted procedure GO_ITEM in PRE-TEXT-ITEM trigger.
```
Using Built-In Definitions
Useful Built-Ins

- EDIT_TEXTITEM
- ENTER_QUERY, EXECUTE_QUERY
- EXIT_FORM
- GET_ITEM_PROPERTY, SET_ITEM_PROPERTY
- GO_BLOCK, GO_ITEM
- MESSAGE
- SHOW_ALERT, SHOW_EDITOR, SHOW_LOV
- SHOW_VIEW, HIDE_VIEW
Using Triggers:
When-Button-Pressed Trigger

- Fires when the operator clicks a button
- Accepts restricted and unrestricted built-ins
- Use to provide convenient navigation, to display LOVs and many other frequently used functions

```sql
GO_BLOCK('Stock');
EXECUTE_QUERY;
```
Using Triggers:
When-Window-Closed Trigger

- Fires when the operator closes a window by using a window manager-specific close command.
- Accepts restricted and unrestricted built-ins.
- Used to programmatically close a window when the operator issues a window manager-specific close command. You can close a window by using built-ins.
Summary

In this lesson, you should have learned that:

• You can use the PL/SQL Editor to write trigger code

• Trigger code has three sections:
  – Declaration section (optional)
  – Executable statements section (required)
  – Exception handlers section (optional)

• You can add functionality by calling built-in subprograms from triggers

• Restricted built-ins are not allowed in triggers that fire while navigation is occurring
Summary

• The **When-Button-Pressed** trigger fires when the user presses a button
• The **When-Window-Closed** trigger fires when the user closes a window
Practice 14 Overview

This practice covers the following topics:

- Using built-ins to display LOVs
- Using the `When-Button-Pressed` and `When-Window-Closed` triggers to add functionality to applications
- Using built-ins to display and hide the Help stack canvas
Debugging Triggers
Objectives

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

• Describe the components of the Debug Console
• Use the Run Form Debug button to run a form module in debug mode
• Debug PL/SQL code
The Debugging Process

Monitor and debug triggers by:

- Compiling and correcting errors in the PL/SQL Editor
- Displaying debug messages at runtime
- Invoking the PL/SQL Debugger
The Debug Console

- Stack
- Variables
- Watch
- Form Values
- PL/SQL Packages
- Global and System Variables
- Breakpoints
The Debug Console: Stack Panel

- Stack
- Variables
- Watch
- Form Values
- PL/SQL Packages
- Global and System Variables
- Breakpoints
The Debug Console: Variables Panel

- Stack
- Variables
- Watch
- Form Values
- PL/SQL Packages
- Global and System Variables
- Breakpoints

Read-only:  

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_SAL_PCT</td>
<td>10</td>
<td>NUMBER</td>
</tr>
<tr>
<td>ReturnValue</td>
<td></td>
<td>BOOLEAN</td>
</tr>
</tbody>
</table>

Modifiable:  

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_OKAY</td>
<td>FALSE</td>
<td>BOOLEAN</td>
</tr>
</tbody>
</table>
The Debug Console: Watch Panel

- Stack
- Variables
- Watch
- Form Values
- PL/SQL Packages
- Global and System Variables
- Breakpoints
The Debug Console: Form Values Panel

- Stack
- Variables
- Watch
- Form Values
- PL/SQL Packages
- Global and System Variables
- Breakpoints
The Debug Console: PL/SQL Packages Panel

- Stack
- Variables
- Watch
- Form Values
- PL/SQL Packages
- Global and System Variables
- Breakpoints
The Debug Console:
Global/System Variables Panel

- Stack
- Variables
- Watch
- Form Values
- Loaded PL/SQL Packages
- Global and System Variables
- Breakpoints
The Debug Console: Breakpoints Panel

- Stack
- Variables
- Watch
- Form Values
- Loaded PL/SQL Packages
- Global and System Variables
- Breakpoints
The Debug Console

- Stack
- Variables
- Watch
- Form Values
- Loaded PL/SQL Packages
- Global and System Variables
- Breakpoints
Setting Breakpoints in Client Code

Breakpoints:
- Suspend form execution
- Return control to the debugger
- Remain in effect for the Forms Builder session
- May be enabled and disabled
- Are set in the PL/SQL Editor on executable lines of code

Before setting breakpoint:

After setting breakpoint:
Setting Breakpoints in Stored Code

• Can set on stored program units:
  – Expand Database Objects node
  – Expand <schema> node
  – Expand PL/SQL Stored Program Units node
  – Double-click program unit
  – Set breakpoint in PL/SQL Editor

• Cannot set on database triggers or stored PL/SQL libraries

• Compile with debug information
Debugging Tips

• Connect to the database for SQL compilation.
• The line that fails is not always responsible.
• Watch for missing semicolons and quotation marks.
• Define triggers at the correct level.
• Place triggers where the event will happen.
Running a Form in Debug Mode

- Run Form
- Debug

(Compiles automatically)

Contains source code and executable run file

.FMX

(Runs automatically)

Runs Form in Debug Mode on Server specified in Runtime Preferences

Preferences

- General
- Subclass
- Wizards
- Runtime

- Buffer Records in File
- Debug Messages
- Array Processing
- Query Only Mode

Application Server URL: http://pgamer.us.oracle.co

Web Browser Location:
Stepping Through Code

- Step over
- Step into
- Step out
- Pause
- Stop
- GO
Debug Example

WHEN-BUTTON-PRESSED

Procedure XYZ;

Function ABC;

The results are: ??

...calls...

1

2

3

4

5
Summary

In this lesson, you should have learned that:

- The Debug Console consists of panes to view the call stack, program variables, a user-defined watch list, Form values, loaded PL/SQL packages, global and system variables, and breakpoints.
- You use the Run Debug button to run a form module in debug mode within Forms Builder.
- You can set breakpoints in the PL/SQL Editor by double-clicking to the left of an executable line of code.
- The debug buttons in the Forms Builder toolbar enable you to step through code in various ways.
Practice 15 Overview

This practice covers the following topics:

- Running a form in debug mode from Forms Builder
- Setting breakpoints
- Stepping through code
- Viewing variable values while form is running
Adding Functionality to Items
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
Item Interaction Triggers

- When-Button-Pressed
- When-Checkbox-Changed
- When-Custom-Item-Event
- When-Radio-Changed
- When-Image-Pressed
- When-Image-Activated
- When-List-Changed
- When-List-Activated
- When-Tree-Node-Activated
- When-Tree-Node-Expanded
- When-Tree-Node-Selected
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)
Interacting with Check Boxes

When-Checkbox-Changed

```sql
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;
```
Changing List Items at Run Time

Triggers:
- When-List-Changed
- When-List-Activated

Built-ins:
- ADD_LIST_ELEMENT
- DELETE_LIST_ELEMENT

Index

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excellent</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Good</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Poor</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
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
LOVs and Buttons

When-Button-Pressed

IF SHOW_LOV('myLov')
THEN...

Employees (LOV)

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roel</td>
<td>101</td>
</tr>
<tr>
<td>Glenn</td>
<td>102</td>
</tr>
<tr>
<td>Gary</td>
<td>103</td>
</tr>
<tr>
<td>Michael</td>
<td>104</td>
</tr>
<tr>
<td>Jeff</td>
<td>105</td>
</tr>
<tr>
<td>Lynn</td>
<td>106</td>
</tr>
<tr>
<td>Kate</td>
<td>107</td>
</tr>
<tr>
<td>Patrice</td>
<td>108</td>
</tr>
<tr>
<td>Pam</td>
<td>109</td>
</tr>
</tbody>
</table>
Populating Image Items

Database

Fetch on query

WRITE_IMAGE_FILE

READ_IMAGE_FILE

Image file (in the application server file system)
Loading the Right Image

READ_IMAGE_FILE

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

Image file in the application server file system
Populating Hierarchical Trees

CREATE_GROUP_FROM_QUERY

Record Group

SET_TREE_PROPERTY

Database

When-Button-Pressed

Car
  Ford
  Volvo
  VW
  Toyota
Displaying Hierarchical Trees

When-Button-Pressed

```sql
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);
```
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
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
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`
Interacting with JavaBeans

The JavaBean may:

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

• 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
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
Run Time Messages and Alerts
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
Run-Time Messages and Alerts Overview

**Alerts**
- System
- Application

**Messages**
- Informative
- Error
- Working
- Application

---

Copyright © 2004, Oracle. All rights reserved.
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
Errors and Built-Ins

- Built-In failure does not cause an exception.
- Test built-in success with **FORM_SUCCESS** function.
  
  \[
  \text{IF } \text{FORM\_SUCCESS } \text{THEN . . .}
  \]
  \[
  \text{OR IF NOT FORM\_SUCCESS THEN . . .}
  \]
- What went wrong?
  - **ERROR\_CODE**, **ERROR\_TEXT**, **ERROR\_TYPE**
  - **MESSAGE\_CODE**, **MESSAGE\_TEXT**, **MESSAGE\_TYPE**
Message Severity Levels

Define by:

:SYSTEM.MESSAGE_LEVEL
Suppressing Messages

```plaintext
: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';
```
The **FORM_TRIGGER_FAILURE** Exception

BEGIN

   --
   --
   RAISE form_trigger_failure;

END;

EXCEPTION

   --
   --
   WHEN <exception> THEN
   RAISE form_trigger_failure;

END;

Fail trigger
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
Handling Informative Messages

- On-Message trigger
- Built-in functions:
  - MESSAGE_CODE
  - MESSAGE_TEXT
  - MESSAGE_TYPE
Setting Alert Properties

<table>
<thead>
<tr>
<th>Title</th>
<th>This is the Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
<td>Alert Message (Maximum 200 characters) Can appear on multiple lines</td>
</tr>
<tr>
<td>Alert Style</td>
<td>Caution</td>
</tr>
<tr>
<td>Button 1 Label</td>
<td>Label 1</td>
</tr>
<tr>
<td>Button 2 Label</td>
<td>Label 2</td>
</tr>
<tr>
<td>Button 3 Label</td>
<td>Label 3</td>
</tr>
<tr>
<td>Default Alert Button</td>
<td>Button 1</td>
</tr>
</tbody>
</table>

Alert Styles:
- Caution
- Stop
- Note

This is the Title

Alert Message (Maximum 200 characters)
Can appear on multiple lines

Label 1
Label 2
Label 3

Copyright © 2004, Oracle. All rights reserved.
Planning Alerts

Yes/No questions

Yes/No/Cancel questions

Caution messages

Informative messages
Controlling Alerts

SET_ALERT_PROPERTY

SET_ALERT_BUTTON_PROPERTY

Copyright © 2004, Oracle. All rights reserved.
SHOW_ALERT Function

IF SHOW_ALERT('del_Check')=ALERT_BUTTON1 THEN

...
PROCEDURE Alert_On_Failure IS
  n NUMBER;
BEGIN
  SET_ALERT_PROPERTY('error_alert',
    ALERT_MESSAGE_TEXT,ERROR_TYPE||'-'||TO_CHAR(ERROR_CODE)||'
    '-'||TO_CHAR(ERROR_CODE)||'
    ':'||ERROR_TEXT);
  n := SHOW_ALERT('error_alert');
END;
Causes of Oracle Server Errors

Form

- Base table block
- Implicit DML
- Trigger/PU
- Explicit DML
- Stored PU call

Oracle Server

- Declarative constraint
- Database trigger
- Stored program unit
Trapping Server Errors

Form

Base table block

On-Error:
- DBMS_ERROR_CODE
- DBMS_ERROR_TEXT

Explicit DML/PU call

When Others:
- SQLCODE
- SQLERRM

Oracle Server

Constraint

Predefined message

DB trigger

RAISE_APPLICATION_ERROR

Stored PU

RAISE_APPLICATION_ERROR
Summary

In this lesson, you should have learned that:

- 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
Summary

• You can control system messages with built-ins and triggers:
  – MESSAGE_LEVEL
  – SUPPRESS_WORKING
  – On-[Error | Message] triggers
  – [ERROR | MESSAGE]_[CODE | TEXT | TYPE]

• Types of alerts: Stop, Caution, Note

• Alert built-ins:
  – SHOW_ALERT
  – SET_ALERT_PROPERTY
  – SET_ALERT_BUTTON_PROPERTY
Summary

• 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
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
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
Query Processing Overview

1. Fire Pre-Query trigger
2. Construct `SELECT...`
3. Perform query
4. Fetch a row into a new record
5. Mark record as valid
6. Fire Post-Query trigger
7. Validate any record changes
8. Abort query on failure
9. Flush record on failure
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
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 combined with the OR operator.
**ONETIME_WHERE Property**

Initially shows restricted query

2nd Execute_Query not restricted
ORDER BY Clause

- Two sources for the ORDER BY clause:
  - ORDER BY Clause block property
  - Query/Where dialog box
- Second source for ORDER BY clause overrides the first one
Writing Query Triggers: Pre-Query Trigger

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

```sql
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;
```
Writing Query Triggers:  
Post-Query Trigger

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

```sql
SELECT COUNT(order_id)
INTO :ORDERS.lineitem_count
FROM ORDER_ITEMS
WHERE order_id = :ORDERS.order_id;
```
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.
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
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
Coding Triggers for Enter-Query Mode

• Example

```plaintext
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.
Overriding Default Query Processing

### Additional Transactional Triggers for Query Processing

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Do-the-Right-Thing Built-in</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-Close</td>
<td></td>
</tr>
<tr>
<td>On-Count</td>
<td>COUNT_QUERY</td>
</tr>
<tr>
<td>On-Fetch</td>
<td>FETCH_RECORDS</td>
</tr>
<tr>
<td>Pre-Select</td>
<td></td>
</tr>
<tr>
<td>On-Select</td>
<td>SELECT_RECORDS</td>
</tr>
<tr>
<td>Post-Select</td>
<td></td>
</tr>
</tbody>
</table>
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.
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
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`
Obtaining Query Information at Run Time

- **GET_ITEMPROPERTY**
- **SET_ITEMPROPERTY**
  - Get and set:
    - CASE_INSENSITIVE_QUERY
    - QUERYABLE
    - QUERY_ONLY
  - Get 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
Summary

• 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 nonbase 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
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
Validation
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
The Validation Process

Forms validates at the following levels:

- Form level
- Block level
- Record level
- Item level
The Validation Process

Validation occurs when:

- [Enter] key or ENTER Built-in is obeyed
- Operator or trigger leaves the validation unit (includes a Commit)
Controlling Validation Using Properties: Validation Unit
Controlling Validation Using Properties: Validate from List

- **LOV**
  - **ENAME**
    - MARTIN
    - MARTINEZ
    - SEDAT
    - WARD
  - **HDDATE**
    - 20-FEB-1981
    - 22-FEB-1981
    - 06-MAR-1996
    - 06-FEB-1995
    - 08-SEP-1981

- **Full list**
  - TERRY

- **Partial list**
  - MART

- **Valid**
  - WARD

- **Auto complete**
  - AL
Controlling Validation 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;
Example: Validating User Input

Customer ID

104

When-Validate-Item

SELECT . . .
WHERE customer_id = :ORDERS.customer_id

Trigger failure?
Using Client-Side Validation

- **Forms validation:**
  - Occurs on middle tier
  - Involves network traffic

- **Client-side validation:**
  - Improves performance
  - Implemented with PJC

![Diagram showing client-side validation examples](image)
Using Client-Side Validation

To use a PJC:

1. Set the item’s Implementation Class property

![Property Palette](image)

2. Set properties for the PJC

```java
SET_CUSTOMPROPERTY('order_items.quantity', 1, 'FILTER_TYPE', 'NUMERIC');
```
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
Controlling When Validation Occurs with Built-Ins

- CLEAR_BLOCK, CLEAR_FORM, EXIT_FORM
- ENTER
- SET_FORMPROPERTY
  - (... , VALIDATION)
  - (... , VALIDATION_UNIT)
- ITEM_IS_VALID item property
- VALIDATE (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
Summary

• 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
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
Navigation
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
Navigation Overview

- What is the navigational unit?
  - Outside the form
  - Form
  - Block
  - Record
  - Item

- Entering and leaving objects

- What happens if navigation fails?
Understanding Internal Navigation

- Next Record
- Exit item
- Exit record
- Enter record
- Enter item
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
Using Object Properties to Control Navigation

- Form module
  - Mouse Navigation Limit
  - First Navigation Data Block
Mouse Navigate Property

MOUSE NAVIGATE = YES

- Exit item
- Exit record
- Exit block
- Enter block
- Enter record
- Enter item
Writing Navigation Triggers

Pre- and Post-

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

<table>
<thead>
<tr>
<th>Pre- and Post-</th>
<th>When-New-&lt;object&gt;-Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fire during navigation</td>
<td>Fire after navigation</td>
</tr>
<tr>
<td>Do not fire if validation unit is higher than trigger object</td>
<td>Fire even when validation unit is higher than the trigger object</td>
</tr>
<tr>
<td>Allow unrestricted built-ins</td>
<td>Allow restricted and unrestricted built-ins</td>
</tr>
<tr>
<td>Handle failure by returning to initial object</td>
<td>Are not affected by failure</td>
</tr>
</tbody>
</table>
When-New-<object>-Instance Triggers

- When-New-Form-Instance
- When-New-Block-Instance
- When-New-Record-Instance
- When-New-Item-Instance
SET_<object>_PROPERTY Examples

```sql
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');
```
The Pre- and Post-Triggers

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

Disabling Stock button when leaving the ORDER_ITEMS block:

```
SET_ITEM_PROPERTY('CONTROL.stock_button',
enabled, property_false);
```
The Navigation Trap
Using Navigation Built-Ins in Triggers

<table>
<thead>
<tr>
<th>Built-In</th>
<th>Built-In</th>
</tr>
</thead>
<tbody>
<tr>
<td>GO_FORM</td>
<td>NEXT_SET</td>
</tr>
<tr>
<td>GO_BLOCK</td>
<td>UP</td>
</tr>
<tr>
<td>GO_ITEM</td>
<td>DOWN</td>
</tr>
<tr>
<td>GO_RECORD</td>
<td>PREVIOUS_BLOCK</td>
</tr>
<tr>
<td>NEXT_BLOCK</td>
<td>PREVIOUS_ITEM</td>
</tr>
<tr>
<td>NEXT_ITEM</td>
<td>PREVIOUS_RECORD</td>
</tr>
<tr>
<td>NEXT_KEY</td>
<td>SCROLL_UP</td>
</tr>
<tr>
<td>NEXT_RECORD</td>
<td>SCROLL_DOWN</td>
</tr>
</tbody>
</table>
Using Navigation Built-Ins in Triggers

- **When-New-Item-Instance**

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

- **Pre-Text-Item**

```sql
IF CHECKBOX_CHECKED('ORDERS.order_mode') --Online
    THEN -- order
        ORDERS.order_status := 4; --Credit order
        GO_ITEM('ORDERS.order_status');
END IF;
```
In this lesson, you should have learned that:

• External navigation is visible to the user, while internal navigation occurs behind the scenes.

• You can control navigation with properties of the form, block, or item:
  – Set in Navigation category of the Property Palette
  OR
  – Use SET_[FORM | BLOCK | ITEM]_PROPERTY
Summary

• Navigation triggers:
  – 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]
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
Transaction Processing
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
Transaction Processing Overview

Transaction (Begin)

FORM A
Action Edit
Save

Block#1
New Record
Updated Record

Block#2
Updated Record
Deleted Record

Transaction (End)

INSERT INTO Table1
UPDATE Table1
DELETE FROM Table2
UPDATE Table2
Commit work;

Copyright © 2004, Oracle. All rights reserved.
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
The Commit Sequence of Events

1. Validate the form
2. Pre-Commit
3. Validate the block
4. Pre-Delete
5. More records?
   - Yes: Pre-Delete, On-Delete
   - No: Delete row
6. Post-Delete
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
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.
## Common Uses for Commit Triggers

<table>
<thead>
<tr>
<th></th>
<th>Pre-Commit</th>
<th>Pre-Delete</th>
<th>Pre-Insert</th>
<th>Pre-Update</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Check user authorization; set up special locking</td>
<td>Journaling; implement foreign-key delete rule</td>
<td>Generate sequence numbers; journaling; automatically generated columns; check constraints</td>
<td>Journaling; implement foreign-key update rule; auto-generated columns; check constraints</td>
</tr>
</tbody>
</table>
# Common Uses for Commit Triggers

<table>
<thead>
<tr>
<th>Event</th>
<th>Action Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-Insert/Update/Delete</td>
<td>Replace default block DML statements</td>
</tr>
<tr>
<td>Post-Forms-Commit</td>
<td>Check complex multirow constraints</td>
</tr>
<tr>
<td>Post-Database-Commit</td>
<td>Test commit success; test uncommitted posts</td>
</tr>
</tbody>
</table>
Life of an Update

[Diagram showing the process of an update:

1. Query
2. Update record in form
3. [Save]
4. Pre-Update
5. Row updated
6. Post-Update
7. Commit]
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;
```
Assigning Sequence Numbers

```
SELECT ORDERS_SEQ.nextval INTO :ORDERS.order_id FROM SYS.dual;
```
Keeping an Audit Trail

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

Post-Insert example:

```sql
: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;
```
Testing the Results of Trigger DML

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

```
UPDATE S_ORD
SET date_shipped = SYSDATE
WHERE id = :S_ORD.id;
IF SQL%NOTFOUND THEN
    MESSAGE('Record not found in database');
    RAISE form_trigger_failure;
END IF;
```
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
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.
Overriding Default Transaction Processing

Additional transactional triggers:

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Do-the-Right-Thing Built-in</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-Check-Unique</td>
<td>CHECK_RECORD_UNIQUENESS</td>
</tr>
<tr>
<td>On-Column-Security</td>
<td>ENFORCE_COLUMN_SECURITY</td>
</tr>
<tr>
<td>On-Commit</td>
<td>COMMIT_FORM</td>
</tr>
<tr>
<td>On-Rollback</td>
<td>ISSUE_ROLLBACK</td>
</tr>
<tr>
<td>On-Savepoint</td>
<td>ISSUE_SAVEPOINT</td>
</tr>
<tr>
<td>On-Sequence-Number</td>
<td>GENERATE_SEQUENCE_NUMBER</td>
</tr>
</tbody>
</table>

Note: These triggers are meant to be used when connecting to data sources other than Oracle.
## Overriding Default Transaction Processing

Transactional triggers for logging on and off:

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Do-the-Right-Thing Built-in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Logon</td>
<td>-</td>
</tr>
<tr>
<td>Pre-Logout</td>
<td>-</td>
</tr>
<tr>
<td>On-Logon</td>
<td>LOGON</td>
</tr>
<tr>
<td>On-Logout</td>
<td>LOGOUT</td>
</tr>
<tr>
<td>Post-Logon</td>
<td>-</td>
</tr>
<tr>
<td>Post-Logout</td>
<td>-</td>
</tr>
</tbody>
</table>
Running Against Data Sources Other than Oracle

• Two ways to run against data sources other than Oracle:
  – Oracle Transparent Gateways
  – Write appropriate transactional triggers
Running 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
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
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
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.
Array DML

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

<table>
<thead>
<tr>
<th>Empno</th>
<th>Ename</th>
<th>Job</th>
<th>Hiredate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Jones</td>
<td>Clerk</td>
<td>01-Jan-1995</td>
</tr>
<tr>
<td>1235</td>
<td>Smith</td>
<td>Clerk</td>
<td>01-Jan-1995</td>
</tr>
<tr>
<td>1236</td>
<td>Adams</td>
<td>Clerk</td>
<td>01-Jan-1995</td>
</tr>
<tr>
<td>1237</td>
<td>Clark</td>
<td>Clerk</td>
<td>01-Jan-1995</td>
</tr>
</tbody>
</table>

Fewer round trips
(exact number depends on array size)

2 inserts
2 updates
1 delete

Database
Effect of Array DML on Transactional Triggers

- **Array DML Size = 1**
  - PRE- Fires
  - POST- Fires
  - Repeated for each insert, update, delete

- **Array DML Size > 1**
  - PRE- Fires
  - POST- Fires
  - Fires for each insert, update, delete

**Array DML Size = 1**

**Array DML Size > 1**
Implementing Array DML

1. Enable the Array Processing option.
2. Specify a DML Array Size of greater than 1.
3. Specify block primary keys.
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:
  1. Validate the form.
  3. Fire Pre-Commit.
  4. Validate the block (performed for all blocks in sequential order).
Summary

5. 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

   If the current operation is COMMIT, then:
   7. Issue an SQL-COMMIT statement.
   8. Fire the Post-Database-Commit trigger.
Summary

- 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
Summary

• Use the Pre-Insert trigger to allocate sequence numbers to records as they are applied to tables.
• Check or change commit status:
  – GET_BLOCK_PROPERTY, [GET | SET]_RECORD_STATUS
  – :SYSTEM.[FORM | BLOCK | RECORD]_STATUS
• Use transactional triggers to override or augment default commit processing.
• Reduce network roundtrips by setting DML Array Size block property to implement 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
Writing Flexible Code
Objectives

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

• Describe flexible code
• State the advantages of using system variables
• Identify built-in subprograms that assist flexible coding
• Write code to reference objects:
  – By internal ID
  – Indirectly
What Is Flexible Code?

Flexible code:
- Is reusable
- Is generic
- Avoids hard-coded object names
- Makes maintenance easier
- Increases productivity
Using System Variables for Current Context

- **Input focus:**
  - SYSTEM.CURSOR_BLOCK
  - SYSTEM.CURSOR_RECORD
  - SYSTEM.CURSOR_ITEM
  - SYSTEM.CURSOR_VALUE

```sql
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;
```
Using System Variables for Current Context

- Trigger focus:
  - `SYSTEM.TRIGGER_BLOCK`
  - `SYSTEM.TRIGGER_RECORD`
  - `SYSTEM.TRIGGER_ITEM`
System Status Variables

When-Button-Pressed

```
ENTER;
IF :SYSTEM.BLOCK_STATUS = 'CHANGED' THEN
   COMMIT_FORM;
END IF;
CLEAR_BLOCK;
```
GET_<object>_PROPERTY
Built-Ins

• GET_APPLICATIONPROPERTY
• GET_FORMPROPERTY
• GET_BLOCKPROPERTY
• GET_RELATIONPROPERTY
• GET_RECORDPROPERTY
• GET_ITEMPROPERTY
• GET_ITEMINSTANCEPROPERTY
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
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
SET_<object>_PROPERTY
Built-Ins

- SET_LOVPROPERTY
- SET_RADIO_BUTTONPROPERTY
- SET_MENU_ITEMPROPERTY
- SET_CANVASPROPERTY
- SET_TAB_PAGEPROPERTY
- SET_VIEWPROPERTY
- SET_WINDOWPROPERTY
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')
FIND_Built-Ins

- FIND_ALERT
- FIND_BLOCK
- FIND_CANVAS
- FIND_EDITOR
- FIND_FORM
- FIND_ITEM
- FIND_LOV
- FIND_RELATION
- FIND_VIEW
- FIND_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.
Using Object IDs

Example:

```sql
DECLARE
    item_var item;
BEGIN
    item_var := FIND_ITEM(:SYSTEM.CURSOR_ITEM);
    SET_ITEMPROPERTY(item_var,position,30,55);
    SET_ITEMPROPERTY(item_var,prompt_text,'Current');
END;
```
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
Referencing Objects Indirectly

Direct reference

ITEM A
Welles

Indirect reference

ITEM B
ITEM A

ITEM A
Welles
Referencing Objects Indirectly

The `NAME_IN` function:

- Returns:
  - The contents of variable
  - Character string
- Use conversion functions for NUMBER and DATE
Referencing Objects Indirectly

The COPY procedure allows:

• Direct copy:

```sql
COPY('Welles','CUSTOMERS.cust_last_name');
```

• Indirect copy:

```sql
COPY('Welles',NAME_IN('global.customer_name_item'));
```
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`
Summary

• The \[\text{GET | SET}]_{\text{<object>}_\text{PROPERTY}}\] built-ins are useful in flexible coding.

• Code that references objects is more efficient and generic:
  – By internal ID: Use \text{FIND}_\text{<object>} built-ins
  – Indirectly: Use \text{COPY} and \text{NAME_IN} built-ins
Practice 22 Overview

This practices covers the following topics:

• Populating product images only when the image item is displayed.

• Modifying the When-Button-Pressed trigger of the Image_Button in order to use object IDs instead of object names.

• Write generic code to print out the names of the blocks in a form.
Sharing Objects and Code
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
Benefits of Reusing Objects and Code

- Increases productivity
- Decreases maintenance
- Increases modularity
- Maintains standards
- Improves application performance
What Are Property Classes?

- Canvas properties
- Relation properties
- Block properties
- Item properties
- LOV properties
Creating a Property Class

Add Property

Inherit Property

Delete Property

Property Class
Inheriting from a Property Class

Property palette
- Default property
- Default property
- Default property

Property palette
- Default property
- Default property
- Changed property

Property class

Property palette
- Inherited property
- Inherited property
- Inherited property

Property palette
- Inherited property
- Inherited property
- Inherited property

Change

Apply

Inherit

Change
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.

<table>
<thead>
<tr>
<th>Inherited Property</th>
<th>Variant Property</th>
<th>Default Property</th>
<th>Changed Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background Color</td>
<td>Background Color</td>
<td>Fill Pattern</td>
<td>Fill Pattern</td>
</tr>
<tr>
<td>r100g0b50</td>
<td>r100g100b50</td>
<td>&lt;Unspecified&gt;</td>
<td>v45waves</td>
</tr>
</tbody>
</table>
What Are Object Groups?

Object groups:
- Are logical containers
- Enable you to:
  - Group related objects
  - Copy multiple objects 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
Copying and Subclassing Objects and Code
Subclassing

- Ability to add to child object
- Ability to inherit changes
- Ability to alter properties of child object

Company Name: [ ]
Company Code: [ ]
Balance: [ ]

Company Name: [ ]
Company Code: [ ]
Balance: [ ]

Company Name: [ ]
Company Code: [ ]
Balance: [ ]

Company Name: [ ]
Company Code: [ ]
Balance: [ ]

Code: [ ]
Address: [ ]
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
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
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 file
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
  – Can be defined in many object libraries
  – Is the preferred method to promote similarity among objects for performance

• You can have many SmartClasses of a given object type.
Working with SmartClasses

1. Right-click an object in the Layout Editor or Navigator.
2. From the pop-up menu, select SmartClasses.
3. Select a class from the list.
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
What Are PL/SQL Libraries?

Applications

Form modules
Menu modules
Report modules

.pll file

Procedures
Functions
Packages

Library
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.
Creating Library Program Units

New Program Unit

Name: 

Types
- Procedure
- Function
- Package Spec
- Package Body
- Type Spec
- Type Body
- Java Source

OK  Cancel  Help
Attach Library Dialog Box
Calls and Searches

Calls

procedure ( );
...function...
package.call ( );

Searches

- Program Units
  - PROCA
  - PROCB
- Attached Libraries
- Database
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, 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.
Summary

- 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
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
Using WebUtil to Interact with the Client
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
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
Benefits of the WebUtil Utility

Why use WebUtil?

• Developer has only to code in PL/SQL (no Java knowledge required)
• Free download (part of Forms 10g in a patch set)
• Easy to integrate into a Forms application
• Extensible
• WebUtil provides:
  – Client-server parity APIs
  – Client-server added value functions
  – Public functions
  – Utility functions
  – Internal functions
Integrating WebUtil into a Form

Step 1: Attach the WEBUTIL library.
Integrating WebUtil into a Form

Step 2: Subclass the WEBUTIL object group.
When to Use 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
## Interacting with the Client

<table>
<thead>
<tr>
<th>Forms Built-Ins / Packages</th>
<th>WebUtil Equivalents</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST</td>
<td>CLIENT_HOST</td>
</tr>
<tr>
<td>GET_FILE_NAME</td>
<td>CLIENT_GET_FILE_NAME</td>
</tr>
<tr>
<td>READ_IMAGE_FILE</td>
<td>CLIENT_IMAGE_READ</td>
</tr>
<tr>
<td>WRITE_IMAGE_FILE</td>
<td>(WRITE)_IMAGE_FILE</td>
</tr>
<tr>
<td>OLE2</td>
<td>CLIENT_OLE2</td>
</tr>
<tr>
<td>TEXT_IO</td>
<td>CLIENT_TEXT_IO</td>
</tr>
<tr>
<td>TOOL_ENV</td>
<td>CLIENT_TOOL_ENV</td>
</tr>
</tbody>
</table>
Example: Opening a File Dialog on the Client

DECLARE
    v_file VARCHAR2(250):= CLIENT_GET_FILE_NAME('', '', 'Gif Files|*.gif|JPEG Files|*.jpg|', 'Select a photo to upload', open_file, TRUE);
Example: Reading an Image File into Forms from the Client

DECLARE
    v_file VARCHAR2(250):= CLIENT_GET_FILE_NAME('','','Gif Files|*.gif|JPEG Files|*.jpg|','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;
Example: Writing Text Files on the Client

DECLARE
    v_dir VARCHAR2(250) := 'c:\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;
DECLARE

v_dir VARCHAR2(250) := 'c:\temp';
ft_tempfile CLIENT_TEXT_IO.FILE_TYPE;

begin

    ft_tempfile := CLIENT_TEXT_IO.FOPEN(v_dir || '	empdir.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;

Example: Executing Operating System Commands on the Client
Example: Performing OLE Automation on the Client

You can use the following for OLE automation:

- `CLIENT_OLE2.OBJ_TYPE`
- `CLIENT_OLE2.LIST_TYPE`
- `CLIENT_OLE2.CREAT_OBJ`
- `CLIENT_OLE2.SET_PROPERTY`
- `CLIENT_OLE2.GET_OBJ_PROPERTY`
- `CLIENT_OLE2.INVOKE_OBJ`

- `CLIENT_OLE2.CREATE_ARGLIST`
- `CLIENT_OLE2.ADD_ARG`
- `CLIENT_OLE2.INVOKE`
- `CLIENT_OLE2.DESTROY_ARGLIST`
- `CLIENT_OLE2.RELEASE_OBJ`
Example: Obtaining Environment Information about the Client

```
CLIENT_TOOL_ENV.GETVAR(:control.env_var,
                        :control.env_value);
```
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
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
  – Use OLE automation to create a form letter on the client
Introducing Multiple Form Applications
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
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
Benefits of Multiple Form Applications

Breaking your application into multiple forms offers the following advantages:

- Easier debugging
- Modularity
- Performance and scalability
Starting Another Form Module

OPEN_FORM

MDI

FORM A

OPEN_FORM

FORM A

FORM B

Modeless

MDI
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.
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
Conditional Opening

Example

```
IF ID_NULL(FIND_FORM('ORDERS')) THEN
    OPEN_FORM('ORDERS');
ELSE
    GO_FORM('ORDERS');
END IF;
```
Closing the Session

“Will the last one out please turn off the lights?”
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.

```sql
ENTER;
IF :SYSTEM.FORM_STATUS = 'CHANGED' THEN
   EXIT_FORM( DO_COMMIT );
ELSE
   EXIT_FORM( NO_COMMIT );
END IF;
```
Other Useful Triggers

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
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
Linking by Global Variables

CUSTOMERS
Customer
ID
GLOBAL.CUSTOMERID

ORDERS
Order
Customer_ID

Copyright © 2004, Oracle. All rights reserved.
Global Variables: Opening Another Form

Example

```plaintext
: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.
Global Variables: Restricted Query at Startup

When-New-Form-Instance

Execute_Query;

Pre-Query

: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
DEFAULT_VALUE(' ', 'GLOBAL.customerid');
```
Linking by Parameter Lists

Parameters:
- Are form module objects
- Properties:
  - Name
  - Parameter Data Type
  - Maximum Length
  - Parameter Initial Value
- Can optionally receive a new value:

http://myhost:8889/forms90/f90servlet
?form=emp.fmx&otherparams=deptno=140
Linking by Parameter Lists

Example:

```
DECLARE
    pl_id     ParamList;
    pl_name  VARCHAR2(10) := 'tempdata';
BEGIN
    pl_id := GET_PARAMETER_LIST(pl_name);
    IF ID_NULL(pl_id) THEN
        pl_id := CREATE_PARAMETER_LIST(pl_name);
    ELSE
        DELETE_PARAMETER(pl_id,'deptno');
    END IF;
    ADD_PARAMETER(pl_id,'deptno',TEXT_PARAMETER,
                  to_char(:departments.department_id));
    OPEN_FORM('called_param',ACTIVATE,NO_SESSION,pl_id);
END;
```
Linking by Parameter Lists

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
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.
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 SHARE_LIBRARY_DATA` option.
6. Use variables in opened form.
Linking by Shared PL/SQL Variables

```sql
OPEN_FORM('called_lib', ACTIVATE, NO_SESSION, SHARE_LIBRARY_DATA);
```
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
Summary

- 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
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
Introduction to Query Builder
Query Builder Features

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

- Easy-to-use data access tool
- Point-and-click graphical user interface
- Distributed data access
- Powerful query building
Query Builder Window
Building a New Query
Datasource Components

Datasource name
Object type

Primary Key
Column name
Foreign Key

Column datatype
Recursive relationship
Comment
Refining a Query
Sorting Data

[Image of a sorting dialog window with columns and options]
Viewing and Saving Queries
Including Additional Tables
Viewing Comments

![Info dialog box showing details of EMPLOYEE_ID column]

- Kind: Column
- Table: EMPLOYEE
- Owner: DEMO
- Datatype: Number
- Comment: Employee Number: Unique 4-digit identification number assigned to every employee.
Including Related Tables
Creating a User-Defined Relationship
Unmatched Rows

Set Relationship

Relationship

A→: departments.manager_id
Operator: = <> < <= > >=

→B: emp_details_view.manager_id

Use Relationship in Query

Options:
- A not found in B
- B not found in A
- Suppress Mismatches

OK Cancel Help
Conditions
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
Multiple Conditions

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

- OR
  - SAL BETWEEN 1000 AND 2000
  - HIREDATE>=’23-jan-86’
  - DEPTNO=20
Deactivating a Condition
Defining Columns Using an Expression

Defined as:

\[ \text{SAL} \times 12 \]
Defining Columns Using a Function
Locking in Forms
Objectives

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

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

Table in (RX)

Row in (X)

Row in (X)

Insert, update, or delete

Query
Default Locking in Forms

- Insert record: No locks
- Update record: RS on table
- Delete record: RS on table
- Action ➔ Save: RX on above
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.
User A: Step 1

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

Count: 5
User B: Step 2

<table>
<thead>
<tr>
<th>Id</th>
<th>Last Name</th>
<th>Dept Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Duma</td>
<td>35</td>
</tr>
</tbody>
</table>

Oracle Forms

Could not reserve record [2 tries]. Keep trying?

Yes  No
### USER A: STEP 3

<table>
<thead>
<tr>
<th>Id</th>
<th>Last Name</th>
<th>First Name</th>
<th>Start Date</th>
<th>Title</th>
<th>Dept Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Magoo</td>
<td>Colin</td>
<td>14-May-90</td>
<td>Sales Representative</td>
<td>31</td>
</tr>
<tr>
<td>12</td>
<td>Gilmum</td>
<td>Henry</td>
<td>18-Jan-92</td>
<td>Sales Representative</td>
<td>32</td>
</tr>
<tr>
<td>13</td>
<td>Sedeghi</td>
<td>Yasmine</td>
<td>18-Feb-91</td>
<td>Sales Representative</td>
<td>33</td>
</tr>
<tr>
<td>14</td>
<td>Nguyen</td>
<td>Mai</td>
<td>22-Jan-92</td>
<td>Sales Representative</td>
<td>34</td>
</tr>
<tr>
<td>15</td>
<td>Dumas</td>
<td>Andre</td>
<td>09-Oct-91</td>
<td>Sales Representative</td>
<td>31</td>
</tr>
</tbody>
</table>
User B: Step 4

Sales Representatives

<table>
<thead>
<tr>
<th>Id</th>
<th>Last Name</th>
<th>Dept Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Dumas</td>
<td>35</td>
</tr>
</tbody>
</table>

FRM-40654: Record has been updated by another user. Re-query to see change.
Count: "1"
Locking in Triggers

Achieved by:

• SQL data manipulation language
• SQL explicit locking statements
• Built-in subprograms
• DML statements
Locking with Built-Ins

- `ENTER_QUERY (FOR_UPDATE)`
- `EXECUTE_QUERY (FOR_UPDATE)`
On-Lock Trigger

Example

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

• Default locking
  – Locks rows during update and delete
  – Informs user of concurrent update and delete

• Locking in triggers
  – Use SQL and certain built-ins
  – On-Lock trigger: LOCK_RECORD built-in available
Oracle Object Features
Objectives

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

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

- **Automatically converted:**
  - FLOAT
  - NLS types
    - NCHAR
    - NVARCHAR2
- **Unsupported:**
  - Timestamp
  - Interval
Object Tables

Object table based on object type
Object Columns

Object column based on object type
Object Views

Object-oriented application → Object view → Relational table

Object views based on object types
INSTEAD-OF Triggers

Nonupdatable view

INSTEAD-OF Trigger

DECLARE

BEGIN

END;

EXCEPTION
References to Objects
Object Types in Object Navigator

![Object Navigator Image](image-url)
Object Type Wizard

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

What are the Attributes of EMP_TYPE?

eno NUMBER
ename VARCHAR2(1)

Attribute Name: esal

Format:
- Datatype: NUMBER
- Length: 15
- Precision: 9
- Scale: 0

Add... Edit... OK Cancel Help

<Back Next> Finish
Object Tables and Columns in Object Navigator
Object Views in Object Navigator
INSTEAD-OF Trigger Dialog Box

```sql
BEGIN
  UPDATE emp e
  SET e.ename = :n.ename, e.sal = :n.esal, e.job = :n.ejob
  WHERE e.empno = :o.eno;
END;
```
Object REFs in Object Navigator
Summary

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

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

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

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

Common features:
- Moving and resizing objects and text
- Defining colors and fonts
- Importing and manipulating images and drawings
- Creating geometric lines and shapes
- Layout surface: Forms canvas view
Invoking the Layout Editor
Layout Editor: Components
Layout Editor: Components
Tool Palette
Selecting Objects
Manipulating Objects

Expand/contract in one direction

Expand/contract diagonally
Moving, Aligning, and Overlapping
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.
Edit and Layout Menus
Color and Pattern Tools
Importing Images
Summary

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