Oracle Developer: Build Forms I

Volume 3 • Student Guide

43112GC10
Production 1.0
April 1999
M08603

ORACLE®

Authors

Fergus Griffin
Ellen Gravina

Technical Contributors and Reviewers

Grant Anderson
David Ball
Christian Bauwens
Ruth Delaney
Brian Fry
Tushar Gadhia
Danae Hadjioannou
Daniel Maas
Jayne Marlow
Stella Misiulis
Mark Sullivan

Publisher

Tommy Cheung

Appendix A: Practice Solutions

Appendix B: Table Descriptions and Data

Appendix G: Using the Layout Editor in Oracle Developer

21

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**
- **Write code to reference objects indirectly**

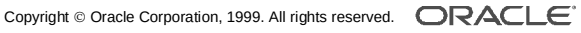
Introduction

Overview

The Oracle Developer Form Builder has a variety of features that enable you to write code in a flexible, reusable way.

Flexible Code

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



What Is Flexible Code?

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

Why Write Flexible Code?

Writing flexible code gives you the following advantages:

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

System Variables for Current Context

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

System Variables for Current Context

- **Trigger focus:**
  - **SYSTEM.TRIGGER_BLOCK**
  - **SYSTEM.TRIGGER_RECORD**
  - **SYSTEM.TRIGGER_ITEM**

Using System Variables for Flexible Coding

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

System Variables for Locating Current Input Focus

System Variable	Function
CURSOR_BLOCK	Determines which block has the input focus
CURSOR_RECORD	Determines which record has the input focus
CURSOR_ITEM	Determines which item in which block has the input focus
CURSOR_VALUE	Determines the value of the item with the input focus

System Variables for Locating Trigger Focus

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

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

System Status Variables

When-Button-Pressed

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

```
IF :SYSTEM.CURSOR_BLOCK = 'S_ORD' THEN
  GO_BLOCK('S_ITEM');
ELSIF :SYSTEM.CURSOR_BLOCK = 'S_ITEM' THEN
  GO_BLOCK('S_INVENTORY');
ELSIF :SYSTEM.CURSOR_BLOCK ='S_INVENTORY' THEN
  GO_BLOCK('S_ORD');
END IF;
```

System Variables for Determining the Current Status of the Form

You can use these system status variables to write code that performs one action for one particular status and a different action for another.

SYSTEM.RECORD_STATUS	Description
CHANGED	The record was fetched from the database, and a base table item on the record has been updated.
INSERT	The user (or a trigger) has entered a value into a base table item of a nonfetched record.
NEW	The user (or a trigger) has not yet entered any values into the base table items of the record.
QUERY	The record was fetched from the database, but no base table item on the record has been updated.

SYSTEM.BLOCK_STATUS	Description
CHANGED	The block contains at least one record with a status of CHANGED or INSERT.
NEW	The block contains only NEW records.
QUERY	The block contains only QUERY records.

SYSTEM.FORM_STATUS	Description
CHANGED	The form contains at least one record with a status of CHANGED or INSERT.
NEW	The form contains only NEW records.
QUERY	The form contains only QUERY records.

GET_*object*_PROPERTY Built-ins

- GET_APPLICATION_PROPERTY
- GET_FORM_PROPERTY
- GET_BLOCK_PROPERTY
- GET_RELATION_PROPERTY
- GET_RECORD_PROPERTY
- GET_ITEM_PROPERTY
- GET_ITEM_INSTANCE_PROPERTY

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

Using Built-in Subprograms for Flexible Coding

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

GET_APPLICATION_PROPERTY

The GET_APPLICATION_PROPERTY built-in returns information about the current Form Builder application.

Example

The following example captures the username and operating system information:

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

GET_BLOCK_PROPERTY

The GET_BLOCK_PROPERTY built-in returns information about a specified block.

Example

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

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

GET_ITEM_PROPERTY

The GET_ITEM_PROPERTY built-in returns information about a specified item.

Example

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

```
DECLARE
 cv_name varchar2(15);
BEGIN
cv_name := GET_ITEM_PROPERTY(:SYSTEM.CURSOR_ITEM,item_canvas);
...
```

SET_*object*_PROPERTY Built-ins

- SET_APPLICATION_PROPERTY
- SET_FORM_PROPERTY
- SET_BLOCK_PROPERTY
- SET_RELATION_PROPERTY
- SET_RECORD_PROPERTY
- SET_ITEM_PROPERTY
- SET_ITEM_INSTANCE_PROPERTY

SET_*object*_PROPERTY Built-ins

- SET_LOV_PROPERTY
- SET_RADIO_BUTTON_PROPERTY
- SET_MENU_ITEM_PROPERTY
- SET_CANVAS_PROPERTY
- SET_TAB_PAGE_PROPERTY
- SET_VIEW_PROPERTY
- SET_WINDOW_PROPERTY

SET_ITEM_INSTANCE_PROPERTY

The SET_ITEM_INSTANCE_PROPERTY built-in modifies the specified instance of an item in a block by changing the specified item property.

Example

The following example sets the visual attribute to VA_CURR for the current record of the current item:

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

SET_MENU_ITEM_PROPERTY

The SET_MENU_ITEM_PROPERTY built-in modifies the given properties of a menu item.

Example

To enable the save menu item in a file menu:

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

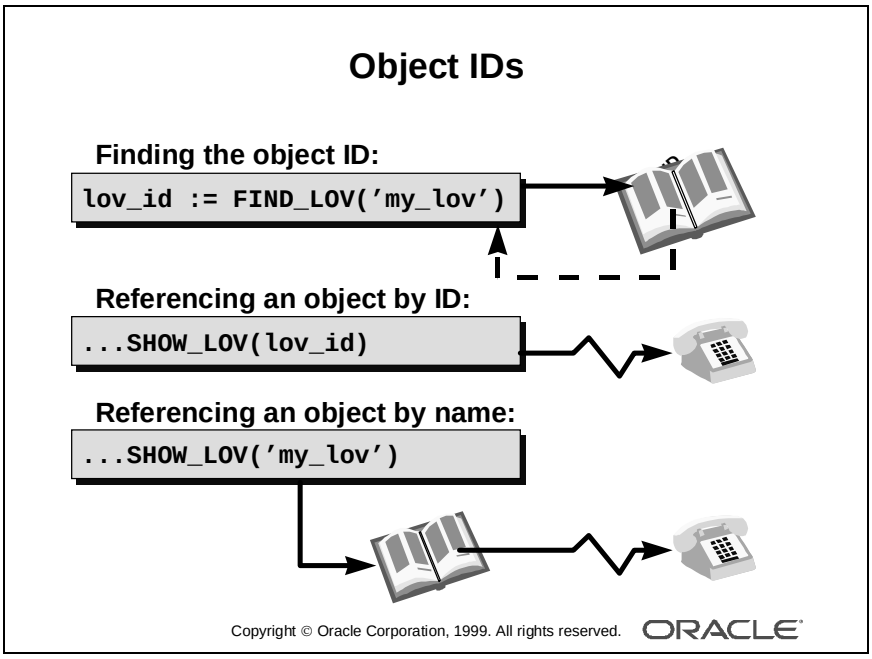
SET_TAB_PAGE_PROPERTY

The SET_TAB_PAGE_PROPERTY built-in sets the tab page properties of the specified tab canvas page.

Example

To enable tab_page_1, if it is already disabled, use:

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


Object IDs
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')
Copyright © Oracle Corporation, 1999. All rights reserved.
ORACLE

Referencing Objects by Internal ID

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

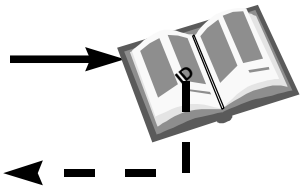
The return values of the FIND_ subprograms (the object IDs) are of a specific type. The types for object IDs are predefined in Form Builder. There is a different type for each object.

Three Reasons for Using Object IDs

- Improving performance (Form Builder looks up the object only once when you initially call the FIND_ subprogram to get the ID. When you refer to an object by name in a trigger, Form Builder must look up the object ID each time.)
- Writing more generic code
- Testing whether an object exists (using the ID_NULL function and FIND_*object*)

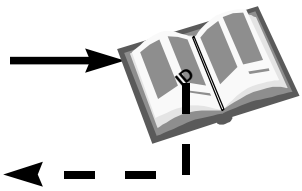
FIND_ Built-ins

- FIND_FORM
- FIND_BLOCK
- FIND_ITEM
- FIND_RELATION
- FIND_LOV



FIND_ Built-ins

- FIND_WINDOW
- FIND_VIEW
- FIND_CANVAS
- FIND_ALERT
- FIND_EDITOR



Form Builder Class Types

The table below lists some of the FIND_ subprograms, along with the object classes that use them and the return types they produce:

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

Using Object IDs

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

Using Object IDs

Example:

```
DECLARE
 item_var item;
BEGIN
 item_var := FIND_ITEM(:SYSTEM.CURSOR_ITEM);
 SET_ITEM_PROPERTY(item_var,position,30,55);
 SET_ITEM_PROPERTY(item_var,prompt_text,'Current');
END;
```

Declaring Variables for Object IDs

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

The following example uses the FIND_ITEM built-in to assign the ID of the item that currently has input focus to the variable *id_var*.

Once you assign an object ID to a variable in a trigger or PL/SQL program unit, you can use that variable to reference the object, rather than referring to the object by name.

```
DECLARE
 id_var item;
BEGIN
 id_var := FIND_ITEM(:SYSTEM.CURSOR_ITEM);
. . .
END;
```

The two examples below show that you can pass either an item name or an item ID to the SET_ITEM_PROPERTY built-in subprogram. The following calls are logically equivalent:

```
SET_ITEM_PROPERTY('S_ITEM.price',position,50,35);


SET_ITEM_PROPERTY(id_var,position,50,35);
```

You can use either object IDs or object names in the same argument list, provided that each individual argument refers to a distinct object.

You cannot, however, use an object ID and an object name to form a fully qualified object_name (`blockname.itemname`). The following call is illegal:

```
GO_ITEM(block_id.'item_name');
```

Note: Use the FIND_ built-in subprograms only when referring to an object more than once in the same trigger or PL/SQL program unit.

Using 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**

Using Object IDs Outside the Initial PL/SQL Block

You have seen how object IDs are referenced within the trigger or program unit by means of PL/SQL variables. You can only reference these PL/SQL variables in the current PL/SQL block; however, you can increase the scope of an object ID.

To reference an object ID outside the initial PL/SQL block, you need to convert the ID to a numeric format using an `.id` extension for your declared PL/SQL variable, then assign it to a global variable.

Example

The following example of trigger code assigns the object ID to a local PL/SQL variable (`item_var`) initially, then to a global variable (`global.item`):

```
DECLARE
 item_var item;
BEGIN
 item_var := FIND_ITEM(:SYSTEM.CURSOR_ITEM);
 :GLOBAL.item := item_var.id;
END;
```

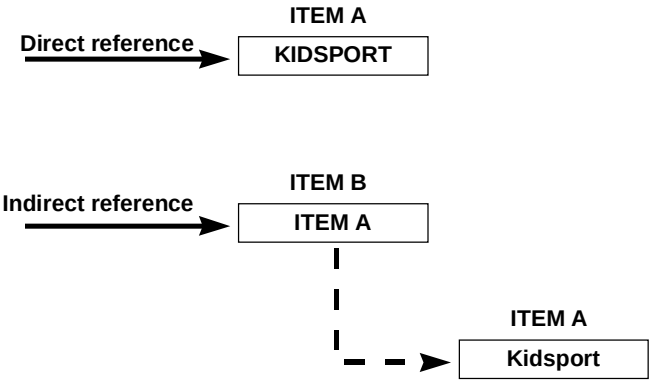
You can pass the global variable around within the application. To be able to reuse the object ID, you need to convert it back to its original data type.

Example

The following example shows the conversion of the global variable back to its original PL/SQL variable data type:

```
DECLARE
 item_var item;
BEGIN
 item_var.id := TO_NUMBER(:GLOBAL.item);
 GO_ITEM(item_var);
END;
```

Referencing Objects Indirectly



Referencing Items Indirectly

By referencing items indirectly, you can write more generic, reusable code. Using variables instead of actual item names, you can write a PL/SQL program unit to use any item whose name is assigned to the indicated variable.

You can reference items indirectly with the NAME_IN and COPY built-in subprograms.

Note: Use indirect referencing when you create procedures and functions in a library module, because direct references cannot be resolved.

Referencing 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**

```
COPY('Kidsport','S_CUSTOMER.name');
```

- **Indirect copy**

```
COPY('Kidsport',:GLOBAL.customer_name);
```

Using the NAME_IN Built-in Function

The NAME_IN function returns the contents of an indicated variable. The following statements are equivalent. The first one uses a direct reference to `customer.name`, whereas the second uses an indirect reference:

```
IF :S_CUSTOMER.name = 'Kidsport'...
```

In a library, you could avoid this direct reference by using:

```
IF NAME_IN('S_CUSTOMER.name') = 'Kidsport'...
```

The return value of NAME_IN is always a character string. To use NAME_IN for a date or number item, convert the string to the desired data type with the appropriate conversion function. For instance:

```
date_var := TO_DATE(NAME_IN('S_ORD.date_ordered'));
```

Using the COPY Built-in Procedure

The COPY built-in assigns an indicated value to an indicated variable or item. Unlike the standard PL/SQL assignment statement, using the COPY built-in enables you to indirectly reference the item whose value is being set. The first example below shows direct referencing and the second shows indirect referencing:

```
COPY('Kidsport','S_CUSTOMER.name');
COPY('Kidsport',:GLOBAL.customer_name);
```

Use the COPY built-in subprogram with the NAME_IN built-in to assign a value to an item whose name is stored in a global variable, as in the following example:

```
COPY('Kidsport',NAME_IN('GLOBAL.customer_name'));
```

Summary

- **Use system variables:**
  - **To avoid hard-coding object names**
  - **To return information about the current state of the form**
- **Use GET_*object*_PROPERTY built-ins to return current property values for Form Builder objects.**

Summary

- **Use object IDs to improve performance.**
- **Use indirect referencing to allow form module variables to be referenced in library and menu modules.**

Summary

Use the following to write flexible code:

- System variables:
  - To avoid hard-coding object names
  - To return information about the current state of the form
- GET_*object*_PROPERTY built-ins, to return current property values for Form Builder objects
- Object IDs, to improve performance
- Indirect referencing, to allow form module variables to be referenced in library and menu modules

Practice 21 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**

Note

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

Practice 21 Overview

In this practice, you use properties and variables in the ORDG*XX* form to provide flexible use of its code. You also make use of object IDs.

- Populating product images only when the image item is displayed
- Modifying the When-Button-Pressed trigger of the Image_Button in order to use object IDs instead of object names

Practice 21

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

   Add a test in the code to check Product_Image. Only perform the trigger actions if the image is currently displayed. Use the GET_ITEM_PROPERTY built-in function.

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

   Use a FIND_*object* function to obtain the IDs of each item referenced by the trigger. Declare variables for these IDs, and use them in each item reference in the trigger.

3. Save, compile, and run the form to test these features.

22

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**
- **Reuse objects from an object library**
- **Reuse PL/SQL code**

Introduction

Overview

The Oracle Developer Form Builder includes some features specifically for object and code reuse. In this lesson, you learn how to share objects between form modules using the Object Library. You also learn how to share code using the PL/SQL Library.

Sharing and Reusing Code

- **Increases productivity**
- **Decreases maintenance**
- **Increases modularity**
- **Maintains standards**

Reusable Objects and Code Overview

When you are developing applications, you should share and reuse objects and code wherever possible in order to:

- Increase productivity
- Decrease maintenance
- Increase modularity
- Maintain standards

Increase Productivity

You can develop applications much more effectively and efficiently if you are not trying to “start over” each time you write a piece of code. By sharing and reusing frequently used objects and code, you can cut down development time and increase productivity.

Decrease Maintenance

By creating applications that use or call the same object or piece of code several times, you can decrease maintenance time.

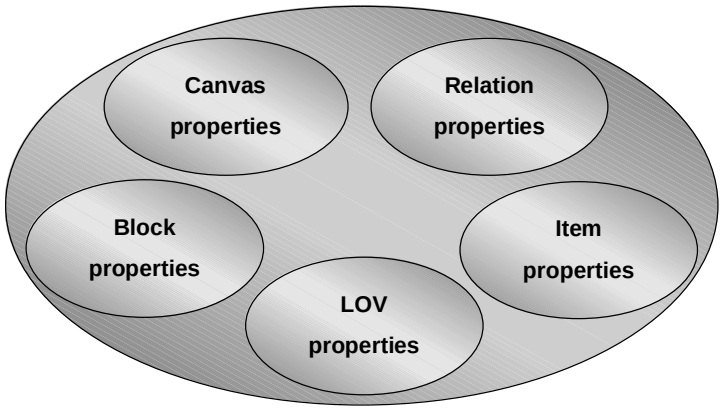
Increase Modularity

Sharing and reusing code increases the modularity of your applications.

Maintain Standards

You can maintain standards by reusing objects and code. If you create an object once and copy it again and again, you do not run the risk of introducing minor changes. In fact, you can create a set of standard objects and some pieces of standard code and use them as a starting point for all of your new form modules.

Property Classes



Property Class

What Is a Property Class?

A *property class* is a named object that contains a list of properties and their settings.

Why Use Property Classes?

Use property classes to increase productivity by setting standard or frequently used values for common properties and associating them with several Form Builder objects. Property classes enable you to define standard properties not just for one particular object, but for several at a time. This results in increased productivity, because it eliminates the time spent on setting identical properties for several objects.

Property Class Icons

Add Property

Inherit Property

Delete Property

Property Class

Creating a Property Class

When you create a property class, you have all the properties from every Form Builder object available. You choose the properties and their values for the property class. You can create a property class in two ways:

- Using the Create button in the Object Navigator
- Using the Create Property Class button

How to Create a Property Class from the Object Navigator

1. Click the Property Class node.
2. Click Create. A new property class entry displays.
3. Add the required properties and their values using the Add Property button in the Property Palette.

How to Create a Property Class from the Property Palette

1. In the Object Navigator, click the object whose properties you want to copy into a property class.
2. Move to the Property Palette, select the properties you want to copy into a property class, and click the Property Class icon. An information alert is displayed.
3. Use the Object Navigator to locate the property class and change its name.

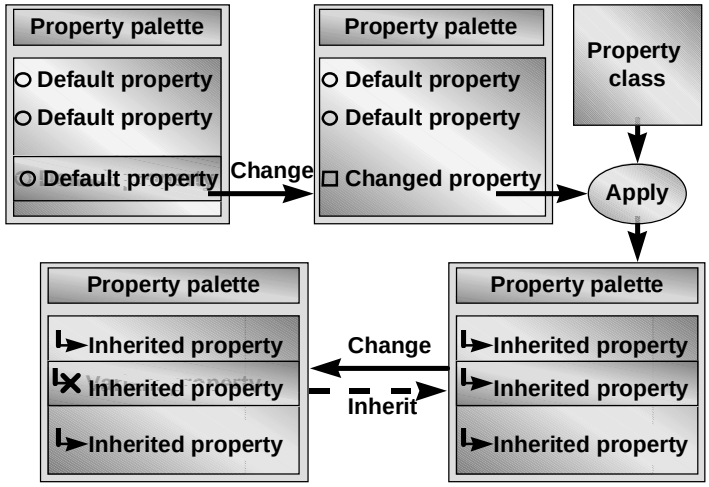
Adding a Property

Once you create a property class, you can add a property by clicking the Add Property button and selecting a property from the list. Set the value for that property using the Property Palette.

Deleting a Property

You can remove properties from a property class using the Delete Property button.

Inherited and Variant Properties



Inheriting Properties

- **Set the Subclass Information property.**
- **Convert an inherited property to a variant property.**
- **Convert a variant property to an inherited property.**

Inheriting a Property Class

Once you create a property class and add properties, you can use the property class. To apply the properties from a property class to an object, use the Subclass Information property in the Property Palette.

What Is an Inherited Property?

An *inherited property* is one that takes its value from the property class that you associated with the object. An inherited property is displayed with an arrow to the left of the property name.

What Is a Variant Property?

A *variant property* is one that has a modified value even though it is inherited from the property class associated with the object. You can override the setting of any inherited property to make that property variant. Variant properties are displayed with a red cross over an arrow.

How to Inherit Property Values from a Property Class

1. Click the object to which you want to apply the properties from the property class.
2. Click the Subclass Information property icon in the Property Palette.
3. Select the property class whose properties you want to use. The object takes on the values of that property class. Inherited properties are displayed with an arrow symbol.

Converting a Variant Property to an Inherited Property

To convert a variant property to an inherited property, click the Inherit property icon in the Property Palette.

Converting an Inherited Property to a Variant Property

To convert an inherited property to a variant property, simply enter a new value over the inherited one.

Object Group

- **Is a logical container**
- **Enables you to:**
  - **Group related objects**
  - **Copy multiple objects in one operation**

Creating an Object Group

What Is an Object Group?

An *object group* is a logical container for a set of Form Builder objects.

Why Use Object Groups?

You define an object group when you want to package related objects for copying or subclassing in another module. You can use object groups to bundle numerous objects into higher-level building blocks that you can use again in another application.

Example

Your application can include an appointment scheduler that you want to make available to other applications. You can package the various objects in an object group and copy the entire bundle in one operation.

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**
  - **An object affects the object group**

How to Create an Object Group

1. Click the Object Group node in the Object Navigator.
2. Click the Create icon. A new object group entry is displayed.
3. Rename the new object group.
4. Click the form module and expand all.
5. Control-click all the objects of one type that you want to include in the object group.
6. Drag the selected objects into the new object group entry. The objects are displayed as object group children.
7. Repeat steps 5 and 6 for different object types.

The objects in the object group are still displayed in their usual position in the Object Navigator, as well as within the object group. The objects in the object group are not duplicates, but pointers to the source objects.

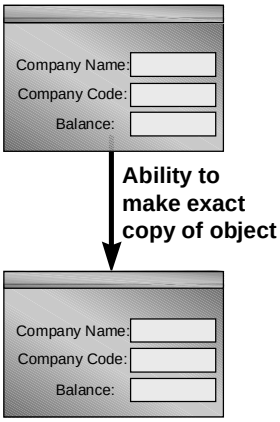
Things to Consider When Using Object Groups

- Including a block in an object group also includes its items, the item-level triggers, the block-level triggers and the relations. You cannot use any of these objects in an object group without the block.
- It is not possible to include another object group.
- Deleting an object from a module automatically deletes the object from the object group.
- Deleting an object group from a module does not delete the objects it contains from the module.

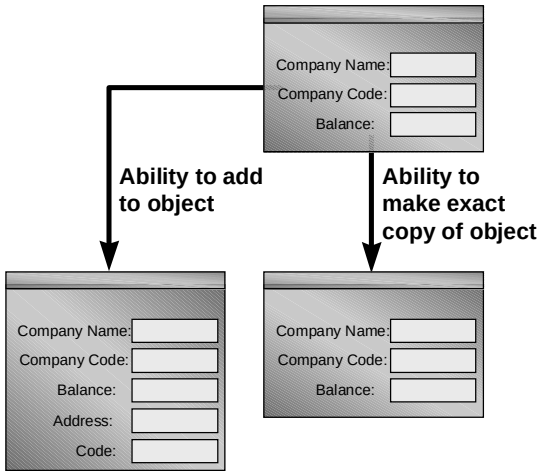
Subclass Information Dialog Box

The Subclass Information property of a form object shows a dialog box that provides information about the origins of the object. You can see whether an object is local to the form document or foreign to it. If the object is foreign to the current form, the Module field shows the module from which the object originates. The original object name is shown in the Object Name field.

Copying Objects



What Is Subclassing?



Copying and Subclassing Objects and Code

When you drag and drop objects between modules in the Object Navigator, the Subclass or Copy alert is displayed. The copy and the subclass mechanisms enable you to reuse objects in more than one module. Specifically, the copy mechanism enables you to create an object in a form or menu source module and then copy that object to another form or menu target module.

In contrast, the subclass mechanism enables you to create an object in a form or menu source module and then subclass that object definition in another form or menu target module.

Note: Subclassing is an object-oriented term that refers to the following capabilities:

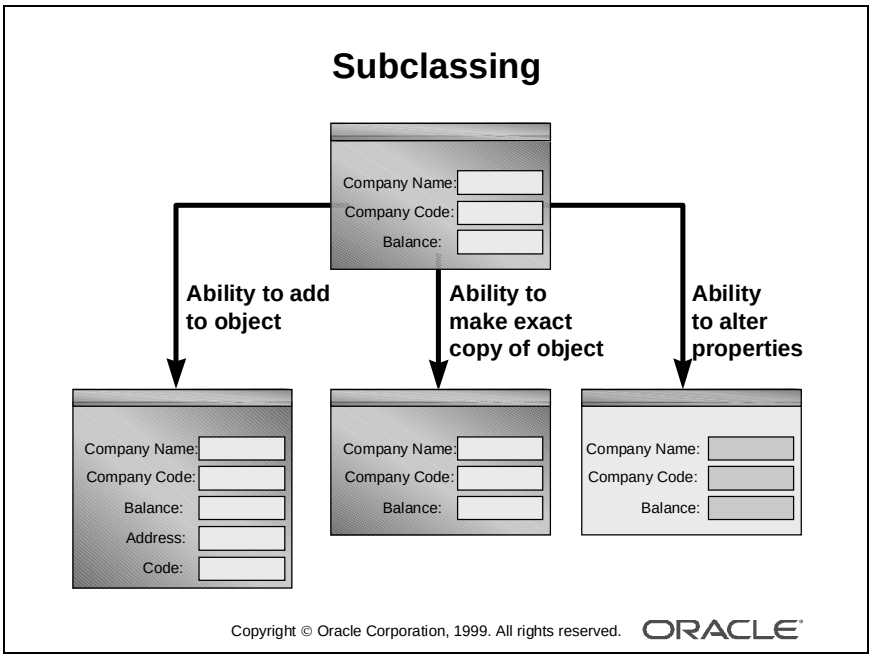
- Inheriting the characteristics of a base class (Inheritance)
- Overriding properties of the base class (Specialization)

Copying an Object

Copying an object creates a separate, unique version of that object in the target module. Any objects owned by the copied object are also copied.

Points to Remember

- Use copying to export the definition of an object to another module.
- Changes made to a copied object in the source module do not affect the copied object in the target module.


Subclassing
Company Name:
Company Code:
Balance:
Ability to add to object
Ability to make exact copy of object
Ability to alter properties
Company Name:
Company Code:
Balance:
Address:
Code:
Company Name:
Company Code:
Balance:
Company Name:
Company Code:
Balance:
Copyright © Oracle Corporation, 1999. All rights reserved.
ORACLE

Subclassing an Object

Earlier versions of Form Builder provided only the ability to inherit from a base class, and this severely limited the usefulness of the reference mechanism. Subclassing is similar to referencing in Oracle Developer Release 1, but is more powerful and reflects true object subclassing.

Ability to Add to an Object

You can still create an exact copy of an object, as with referencing, but you can add to the subclassed object. For example, you can add additional items to a subclassed block.

Ability to Alter Properties

With subclassing, you can make an exact copy and then alter the properties of some objects. If you change the parent class, the changes also apply to the properties of the subclassed object that you have not altered. However, any properties that you override remain overridden.

This provides a powerful object inheritance model. With Property Palette, you can identify inherited or overridden properties.

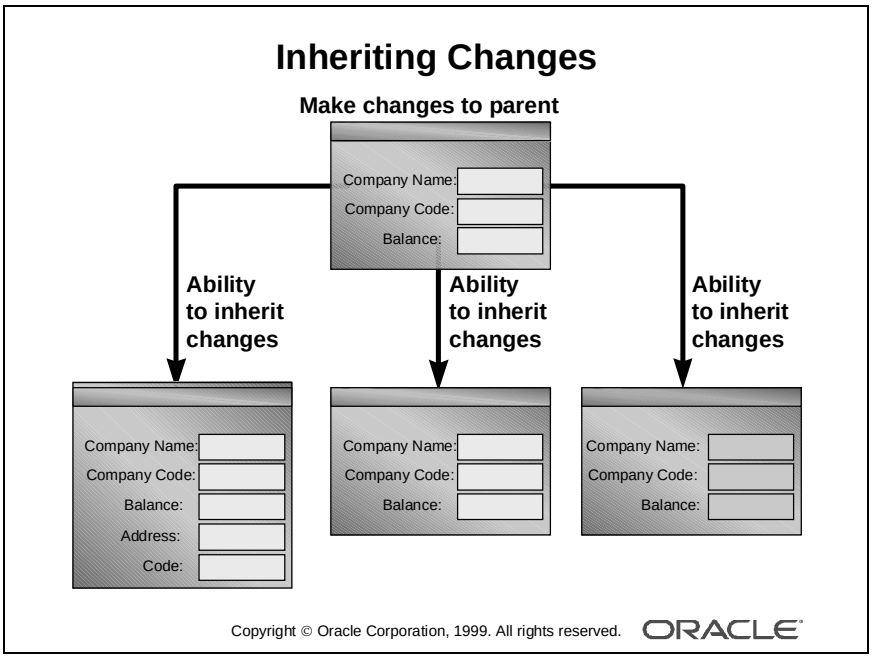
Property Palette Icon	**Meaning**
Circle	The value for the property is the default.
Square	The value for the property was changed from the default.
Arrow	The value for the property was inherited.
Arrow with a red cross over it	The value for the property was inherited but overridden (variant property).

Ability to Inherit Changes

When you change the properties of a parent object, all child objects inherit those properties if they are not already overridden.

The child inherits changes:

- Immediately, if the parent and child objects are in the same form
- When you reload the form containing a child object


Inheriting Changes
Make changes to parent
Company Name:
Company Code:
Balance:
Ability to inherit changes
Ability to inherit changes
Ability to inherit changes
Company Name:
Company Code:
Balance:
Address:
Code:
Company Name:
Company Code:
Balance:
Company Name:
Company Code:
Balance:
Copyright © Oracle Corporation, 1999. All rights reserved.
ORACLE

Example

You can subclass an object within a form module by selecting the object in the Object Navigator, pressing [Control], and dragging it to create the new object. A dialog box appears that asks whether you want to copy or subclass the object.

When you subclass a data block, you can*:*

- Change the structure of the parent, automatically propagating the changes to the child
- Add or change properties to the child to override the inheritance

When you subclass a data block you cannot:

- Delete items from the child
- Change the order of items in the child
- Add items to the child unless you add them to the end

Points to Remember

- You can only add items to the end of the subclassed block; you cannot sequence new items before or between existing, subclassed items. This is intentional behavior that enables the layout frame to control and arrange the original items.
- The major benefit of subclassing is that you can now specialize as well as inherit. This makes reuse much more useful.
- Another improvement, mainly one of convenience, is that you do not need to close and reopen a form to see changes to inherited objects.
- Inheritance also works within a single form. For example, if you want to create another button just like an existing one but with a different label, you simply subclass the button. In the past you had to create a property class to do this.
- There is no limit to the depth of your subclass hierarchy, other than your ability to manage the complexity.

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**

What Is an Object Library?

Object libraries are convenient containers of objects for reuse. They simplify reuse in complex environments, and they support corporate, project, and personal standards.

An object library can contain simple objects, property classes, object groups, and program units, but they are protected against change in the library. Objects can be used as standards (classes) for other objects.

Object libraries simplify the sharing of reusable components. Reusing components enables you to:

- Apply standards to simple objects, such as buttons and items, for a consistent look and feel
- Reuse complex objects such as a Navigator

In combination with SmartClasses, which are discussed later, object libraries support both of these requirements.

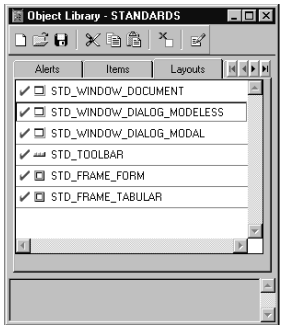
Note: Form Builder opens all libraries that were open when you last closed Form Builder.

Why Object Libraries Instead of Object Groups?

- Object libraries can contain individual items; for example, iconic buttons. The smallest unit accepted in an object group is a block.
- Object libraries accept PL/SQL program units.
- If you change an object in an object library, all forms that contain the subclassed object reflect the change.

Object Libraries

- Appear in the Navigator if they are open
- Are used with a simple tabbed interface
- Are saved to `.olb` file or to database



Benefits of the Object Library

- Simplifies the sharing and reuse of objects
- Provides control and enforcement of standards
- Eliminates the need to maintain multiple referenced forms

Working with Object Libraries

Object libraries appear in the Navigator if they are open. You can create, open, and close object libraries like other modules. Object libraries are automatically reopened when you start up Form Builder, making your reusable objects immediately accessible. It is easy to use object libraries with a simple tabbed interface. This interface has Add and Remove tab pages that help you to create your own groups. Double-clicking on the tab and dragging an object to it from the object library creates a subclassed object. To create a copy of the object, press [Control] and drag the object. To move the object out of the object library and into a form, press [Shift] and drag the object. You can save object libraries to a file system as `.olb` files, or to the database.

How to Populate an Object Library

1. Select Tools—>Object Library to display the object library.
2. Drag objects from the Object Navigator into the object library. The descriptive comment comes from the object Comment property, but it can be edited independently of that property.

Benefits of the Object Library

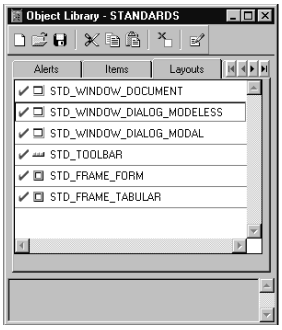
There are several advantages to using object libraries to develop applications:

- Simplifies the sharing and reuse of objects
- Provides control and enforcement of standards
- Eliminates the need to maintain multiple referenced forms

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**

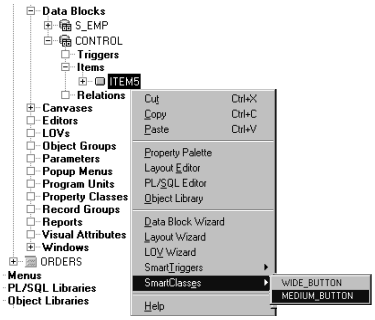
You can have many SmartClasses of a given object type.



Check indicates a SmartClass.

Working with SmartClasses

1. **Select an object in the Layout Editor or Navigator.**
2. **From the pop-up menu, select SmartClasses.**
3. **Select a class from the list.**



What Is a SmartClass?

A SmartClass is a special member of an Object Library. Unlike other Object Library members, it can be used to subclass existing objects in a form using the SmartClass option from the right mouse button popup menu. Object Library members which are not SmartClasses can only be used to create new objects in form modules into which they are added.

If you frequently use certain objects as standards, such as standard buttons, date items, and alerts, you can mark them as SmartClasses by selecting each object in the object library and choosing Object—>SmartClass.

You can mark many different objects that are spread across multiple object libraries as SmartClasses.

You can also have many SmartClasses of a given object type; for example:

- Wide_Button
- Narrow_Button
- Small_Iconic_Button
- Large_Iconic_Button

How to Work with SmartClasses

1. Select an object in the Layout Editor or Navigator.
2. From the pop-up menu, select SmartClasses. The SmartClasses pop-up menu lists all the SmartClasses from all open object libraries that have the same type as the object. When you select a class for the object, it becomes the parent class of the object. You can see its details in the Subclass Information dialog box, just like any other subclassed object.

This mechanism makes it very easy to apply classes to existing objects.

Reusing PL/SQL

- **Triggers:**
  - **Copy and paste text**
  - **Copy and paste within a module**
  - **Copy to or subclass from another module**
  - **Move to an object library**
- **PL/SQL program units:**
  - **Copy and paste text**
  - **Copy and paste within a module**
  - **Copy to or subclass in another module**
  - **Create a library module**
  - **Move to an object library**

Reusing PL/SQL

PL/SQL in Triggers

You can reuse the PL/SQL in your triggers by:

- Copying and pasting, using the Edit menu
- Copying to another area of the current form module, using Copy and Paste on the menu of the right mouse button
- Copying to or subclassing from another form module, using drag and drop in the Object Navigator
- Moving the trigger to an object library

PL/SQL Program Units

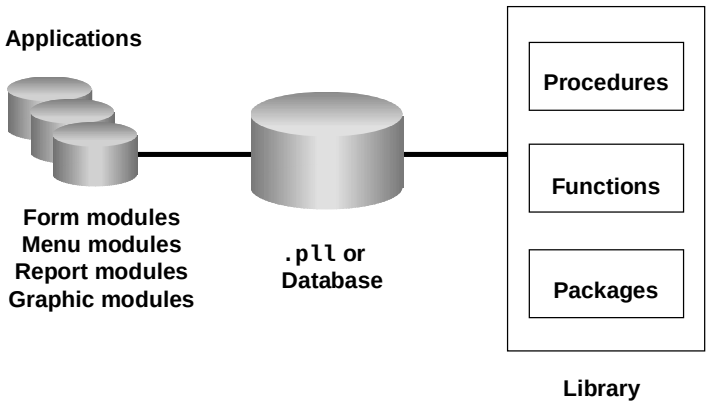
Although triggers are the primary way to add programmatic control to a Form Builder application, using PL/SQL program units supplement triggers, you can reuse code without having to retype it.

With Form Builder, you can create PL/SQL program units to hold commonly used code. These PL/SQL program units can use parameters, which decrease the need to hard-code object names within the procedure body.

You can reuse PL/SQL program units by:

- Copying and pasting, using the Edit menu
- Copying or subclassing to another form module, using drag and drop in the Object Navigator
- Creating a library module
- Moving the program unit to an object library

PL/SQL Libraries



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.**

PL/SQL Libraries

A *library* is a collection of PL/SQL program units, including procedures, functions, and packages. A single library can contain many program units that can be shared among the Oracle Developer modules and applications that need to use them.

A library:

- Is produced as a separate module and stored in either a file or the database
- Provides a convenient means of storing client-side code and sharing it among applications
- Means that a single copy of program units can be used by many form, menu, report, or graphic modules
- Supports dynamic loading of program units

Scoping of Objects

Because libraries are compiled independently of the Oracle Developer modules that use them, bind variables in forms, menus, reports, and displays are outside the scope of the library. This means that you cannot directly refer to variables that are local to another module, because the compiler does not know about them when you compile the library program units.

There are two ways to avoid direct references to bind variables:

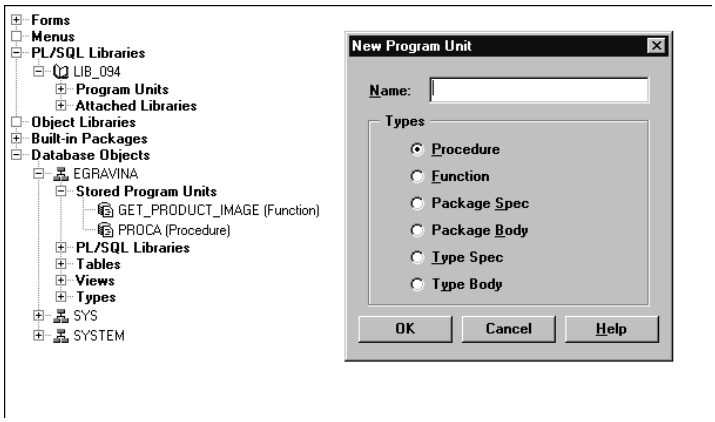
- You can refer to global variables and system variables in forms indirectly as quoted strings by using certain built-in subprograms.
- Write program units with IN and IN OUT parameters that are designed to accept references to bind variables. You can then pass the names of bind variables as parameters when calling the library program units from your Oracle Developer applications.

Example

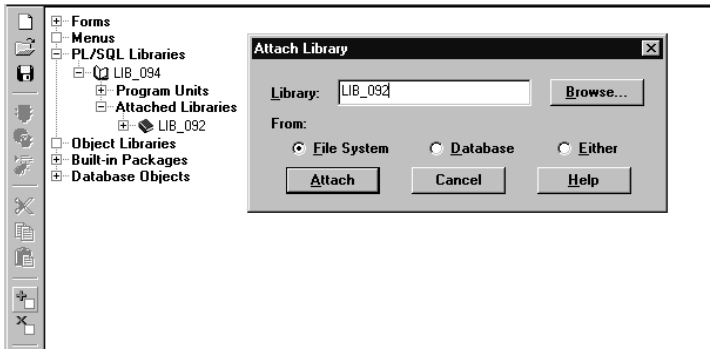
Consider the second method listed above in the following library subprogram:

```
FUNCTION locate_emp(bind_value IN NUMBER) RETURN VARCHAR2 IS
   v_ename VARCHAR2(15);
BEGIN
   SELECT ename INTO v_ename FROM emp WHERE empno = bind_value;
   RETURN(v_ename);
END;
```

Creating Library Program Units



Attach Library Dialog Box



Working with PL/SQL Libraries

Creating a Library

You must first create libraries in the builder before you add program units. To do this, you can either:

- Select File—>New—>Library from the menus (An entry for the new library then appears in the Navigator)
- Select the Libraries node in the Navigator, and select the Create tool from the tool bar

There is a Program Units node within the library's hierarchy in the Navigator. From this node, you can create procedures, functions, package bodies, and specifications in the same way as in other modules.

How to Save the Library

1. With the context set on the library, select the Save option in Form Builder.
2. Select the destination for the library (file system or database), if required, and the name by which the library is to be saved.

How to Attach a Library

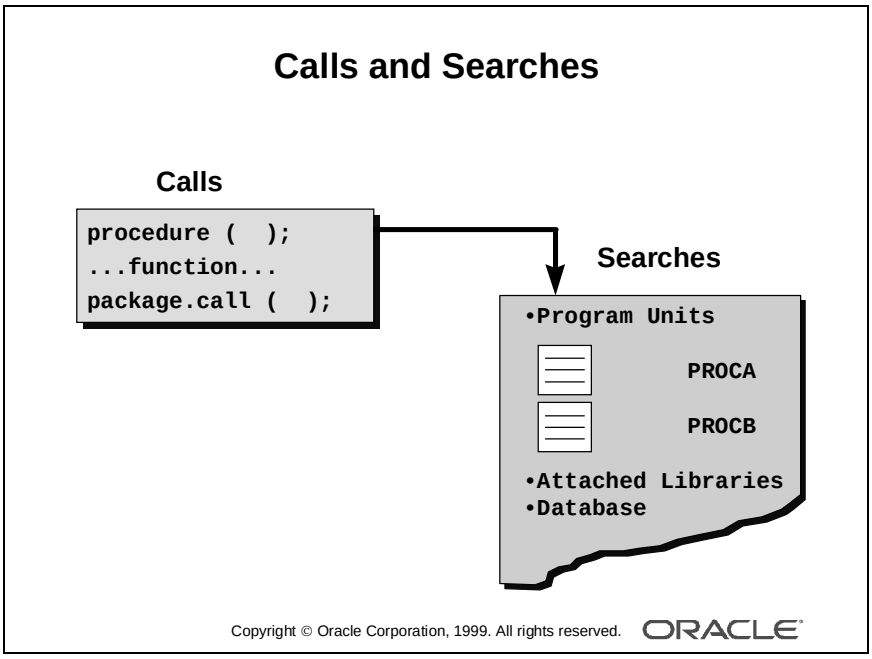
Before you can refer to a library program unit from a form, menu, report, or graphics, you must attach the library to the modules.

To attach a library to a module:

1. Open the module that needs to be attached to the library. This may be a form, menu, report, or graphics module, or another library module.
2. Expand the module and select the Attached Libraries node in the Navigator. When you select Create, the Attach Library dialog box appears.
3. In the Attach Library dialog box, specify the library's name and select File System or Database.
4. Click Attach.
5. Save the module to which you have attached the library. This permanently records the library attachment in the definition of this module.

Detaching a Library

To later detach a library, simply delete the library entry from the list of Attached Libraries for that module, in the Navigator. That module will then no longer be able to reference the library program units, either in the designer or at run time.


Calls and Searches
Calls
procedure ( );
...function...
package.call ( );
Searches
•Program Units
PROCA
PROCB
•Attached Libraries
•Database
Copyright © Oracle Corporation, 1999. All rights reserved.
ORACLE

Referencing Attached Library Program Units

You refer to library program units in the same way as those that are defined locally, or stored in the database. Remember that objects declared in a package must be referenced with the package name as a prefix, whether or not they are part of a library.

Program units are searched for first in the calling module, then in the libraries that are attached to the calling module.

Example

Assume that the program units report_totals, how_many_people, and pack5.del_emps are defined in an attached library:

```
report_totals(:sub1);      --library procedure
v_sum := how_many_people;  --library function
pack5.del_emps;            --library package procedure
```

When Several Libraries Are Attached

You can attach several libraries to the same Oracle Developer module. References are resolved by searching through libraries in the order in which they occur in the attachment list.

If two program units of the same name and type occur in different libraries in the attachment list, the one in the “higher” library will be executed, since it is located first.

Creating `.PLX` Files

The library `.PLX` file is a platform-specific executable that contains no source.

When you are ready to deploy your application, you will probably want to generate a version of your library that contains only the compiled p-code, without any source. You can generate a `.PLX` file from Form Builder or from the command line.

Example

The following command creates a run-time library named `runlib1.plx` based on the open library `mylib.pll`:

```
GENERATE LIB mylib FILE runlib1;
```

Summary

- **Reasons to share objects and code:**
  - **Increased productivity**
  - **Increased modularity**
  - **Decreased maintenance**
  - **Maintaining standards**

Summary

- **Methods of sharing objects and code:**
  - **Using property classes**
  - **Using object groups**
  - **Copying**
  - **Subclassing**
  - **Creating a library module**
  - **Using object libraries**

Summary

Forms provides a variety of methods for reusing objects and code. This lesson described how to use these methods.

Reasons to share objects and code:

- Increased productivity
- Increased modularity
- Decreased maintenance
- Maintaining standards

Methods of sharing objects and code:

- Using property classes: Defines standard properties for several objects at a time
- Using object groups: Bundles numerous objects into higher-level building blocks that can be used again in another application
- Copying: Exports the definition of an object to another module
- Subclassing: Creates an object in one module and then subclasses the current definition of that object in any number of target modules at build time; extends referencing to include object-oriented capabilities
- Creating a library module
- Using object libraries: Enables drag and drop reuse, and provides SmartClasses for default objects

Practice 22 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**
- **Setting and using SmartClasses**

Note

For solutions to this practice, see Practice 22 in Appendix A, “Practice Solutions.”

Practice 22 Overview

In this practice, you use an object group and an object library to copy Form Builder objects from one form to another. You will also create a property class and use it to set multiple properties for several objects. You set SmartClasses in the object library and use these classes in the form module.

- Creating an object group and using this object group in a new form module
- Using property classes
- Creating an object library and using this object library in a new form module
- Setting and using SmartClasses

Practice 22

1. In the ORDG*XX* form, create an object group called Stock_Objects, consisting of the S_INVENTORY block, CV_INVENTORY and WIN_INVENTORY.
2. Save the form.
3. Create a new form module and copy the Stock_Objects object group into it.
4. In the new form module, create a property class called ClassA.

   Include the following properties and settings:

Property	**Setting**
Font Name	Arial
Format Mask	99,999
Font Size	8
Justification	Right
Delete Allowed	No
Background Color	DarkRed

5. Apply ClassA to CV_INVENTORY, the Restock_Date item and the Max_In_Stock item.
6. Save the form module as `STOCK`*`XX`*`.fmb`, compile, run the form, and note the error.
7. Make the Restock_Date format mask a variant property.

   Change the format mask for S_INVENTORY.Restock_Date to MM/DD/YYYY.
8. Correct the error. Save, compile, and run the form again.
9. Create an object library and name it `summit`.

   Create two tabs in the object library called Personal and Corporate.

   Add the CONTROL block, the Toolbar, and the Question_Alert to the personal tab of the object library.

   Save the object library as `summit.olb`.

Practice 22 (continued)

10 Create a new form, and create a data block based on the S_DEPT table.

Drag the Toolbar canvas, CONTROL block, and Question_Alert from the object library into the new form. For proper behavior, the S_DEPT block must be before the CONTROL block in the Object Navigator. Subclass the objects.

Some items are not applicable to this form. Set the Canvas property for the following items to NULL: Image_Button, Stock_Button, Show_Help_Button, Product_Lov_Button, Hide_Help_Button.

Use Toolbar as the form horizontal toolbar canvas for this form.

Set the Window property to WINDOW1 for the Toolbar canvas.

Set the Horizontal Toolbar Canvas property to TOOLBAR for the window.

Save this form as DEPTG*XX*, compile, and run the form to test it.

11 Try to delete items on the Null canvas. What happens and why?

12 Create two sample buttons, one for wide buttons and one for medium buttons, by means of width.

Create a sample date field. Set the width and the format mask to your preferred standard.

Drag these items into your object library.

Mark these items as SmartClasses.

Create a new form and a new data block in the form. Apply these SmartClasses in your form. Place the Toolbar canvas in the new form.

23

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**

Introduction

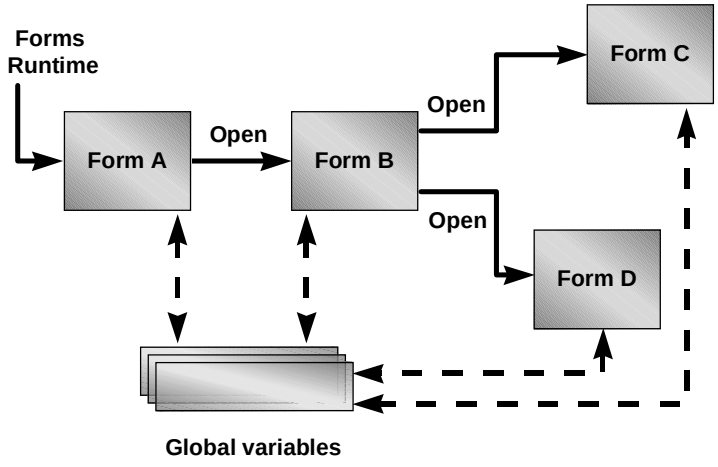
Overview

Oracle Developer applications rarely consist of a single form document. This lesson introduces you to the ways in which you can link two or more forms.

Multiple Form Applications

- **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 or parameter lists**
  - **Code is shared as required, through libraries and the database**

Multiple Form Session



Multiple Form Applications

At the beginning of the course, we discussed the ability to design Form Builder applications where blocks are distributed over more than one form, producing a modular structure. A modular structure indicates the following:

- Component forms are only loaded in memory if they are needed.
- One form can be called from another, providing flexible combinations, as required.

How Does the Application Behave?

The first form module to run is specified before the Form Builder session begins, using Forms Runtime. Other form modules can be opened in the session by calling built-ins from triggers.

You can design forms to appear in separate windows, so the user can work with several forms concurrently in a session (when forms are invoked by the OPEN_FORM built-in). Users can then navigate between visible blocks of different forms, much as they can in a single form.

You can design forms for a Forms Runtime session according to the following conditions:

- Forms share the same database session, or open their own separate sessions.
- Database transactions are continued across forms, or ended before control is passed to another form. The commit sequence starts from the current form and follows the opening order of forms.
- Form Builder provides the same menus across the application, or each form provides its own separate menus when it becomes the active form.

What Links the Forms Together?

Each form runs within the same Forms Runtime session, and Form Builder remembers the form that invoked each additional form. This chain of control is used when you exit a form or commit transactions.

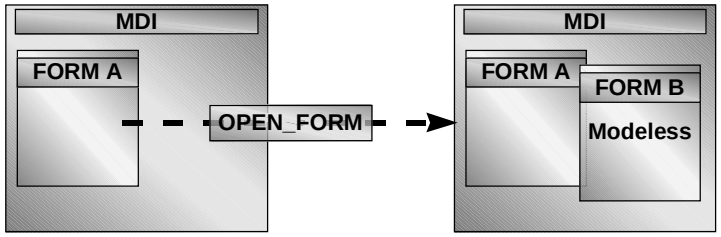
Data can be exchanged between forms as follows:

- Through global variables, which span sessions
- Through parameter lists, for passing values between specific forms

Code can be shared through the following:

- Library modules, by attaching them to each form as required
- Stored program units in the database

OPEN_FORM



How to Start Another Form Module

When the first form in a Forms Runtime session has started, it can provide the user with facilities for starting additional forms. This can be done by one of two methods:

- Calling a built-in procedure from a trigger in the form
- Calling a built-in procedure from a menu item in an attached menu

In either case, the available built-ins are those described below.

Built-in Procedures for Starting Another Form

There are three procedures that you can use to start another form module from one that is already active: OPEN_FORM, CALL_FORM, and NEW_FORM.

OPEN_FORM This is a restricted procedure, and cannot be called in Enter Query mode. OPEN_FORM enables you to start another form in a modeless window, so the user can work in other running forms at the same time. This built-in is normally the preferred way of providing multiple form applications.

You can start another form using OPEN_FORM without passing control to it immediately, if required. This built-in also gives you the option to begin a separate database session for the new form.

```
OPEN_FORM('form_name', activate_mode, session_mode,
          data_mode, paramlist);
```

Parameter	Description
Form_Name	Filename of the executable module (without the .FMX suffix)
Activate_Mode	Either ACTIVATE (the default), or NO_ACTIVATE
Session_Mode	Either NO_SESSION (the default) or SESSION
Data_Mode	Either NO_SHARE_LIBRARY_DATA (the default) or SHARE_LIBRARY_DATA (Use this parameter to enable Form Builder to share data among forms that have identical libraries attached.)
Paramlist	Either the name (in quotes) or internal ID of a parameter list

OPEN_FORM and the 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.**

OPEN_FORM and the Summit Application

Actions:

1. Define windows and positions for each form.

2. Plan global variables and their names.

3. Implement triggers to:

- **Open other forms**
- **Initialize globals from calling forms**
- **Use globals in opened forms**

Defining Multiple Form Functionality

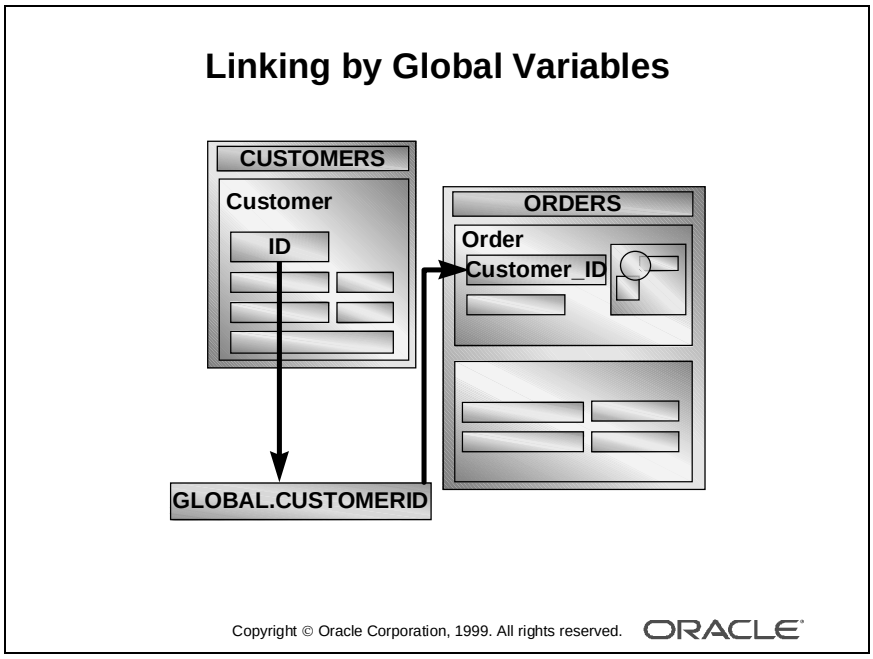
Using OPEN_FORM to Provide Forms in Multiple Windows

You can use OPEN_FORM to link form modules in an application and enable the user to work in them concurrently. Consider these requirements for the Summit application:

- The CUSTOMERS form must provide an option to start the ORDERS form in the same transaction, and orders for the current customer can be viewed, inserted, updated, and deleted.
- The user can see all open forms at the same time, and freely navigate between them to apply changes.
- Changes in all forms can be saved together.

To provide this kind of functionality, perform the following steps:

1. Create each of the form modules. Plan where the windows of each module will appear in relation to those of other modules.
2. Plan names for global variables. You need one for each item of data that is to be accessible across all the forms in the application. Note that each form must reference a global variable by the same name.
3. Plan and implement triggers to:
   - Open another form (You can do this from item interaction triggers, such as When-Button-Pressed, or from When-New-Object-Instance triggers, or from a Key- trigger that fires on a keystroke or equivalent menu selection.)
   - Initialize global variables in calling forms (This is so that values such as unique keys are accessible to other forms when they open. This might need to be done in more than one trigger, if the reference value changes in the calling form.)
   - Make use of received global variables in opened forms (For example, a Pre-Query trigger can use the global variable's contents of the global variable as query criteria.)



Defining Multiple Form Functionality

Note: It is possible to start up several instances of the same form, using OPEN_FORM, unless the application does appropriate tests before calling this built-in. For example, test a flag (global variable) set by an opened form at startup, which the opened form could reset on exit, or use the FIND_FORM built-in.

Planning Global Variables and Their Names

You need a global variable for each item of data that is used across the application. Reminders:

- Global variables contain character data values, with a maximum of 255 characters.
- Each global variable is known by the same name to each form in the session.
- Global variables can be created by a PL/SQL assignment, or by the DEFAULT_VALUE built-in, which has no effect if the variable already exists.
- Reading from a nonexistent global variable causes an error.

The scenario in the slide on the opposite page shows one global variable: GLOBAL.CUSTOMERID ensures that orders queried at the startup of the ORDERS form apply to the current customer.

Opening the ORDERS Form from the CUSTOMERS Form

This When-Button-Pressed trigger on :CONTROL.Orders_Button opens the ORDERS form, and passes control immediately to it. ORDERS will use the same database session and transaction.

```
:GLOBAL.customerid := :S_CUSTOMER.id;
OPEN_FORM('ORDERS');
```

Opening Another Form

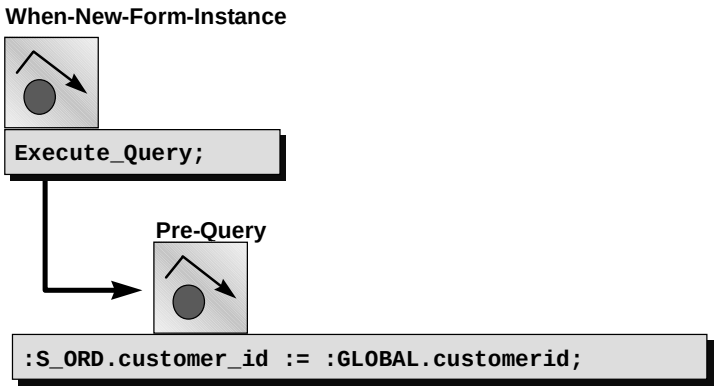
Example:

```
:GLOBAL.customerid := :S_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.**

Restricted Query at Startup



Opening Another Form

- When you default the Activate_Mode argument in OPEN_FORM, control is passed immediately to the specified form, and any remaining statements after OPEN_FORM are not executed.
- If you set Activate_Mode to NO_ACTIVATE, control remains in the calling form, although the specified form starts up and the rest of the trigger is processed. Users can then navigate to the other form when they choose.
- If you want to end the current transaction before opening the next form, call the COMMIT_FORM built-in before OPEN_FORM. (Alternatively, you can just post changes to the database with POST, then open the next form in the same transaction.)

Performing a Restricted Query on Startup

To display a query automatically in the opened form, with data in context to the calling form, you produce two triggers:

- When-New-Form-Instance: This form-level trigger fires when the form is opened (regardless of whether control is passed to this form immediately or not). A query can be initiated by the EXECUTE_QUERY built-in procedure. This, in turn, fires a Pre-Query trigger if one is defined. This trigger is in the ORDERS form:

  ```
  EXECUTE_QUERY;
  ```

- Pre-Query: This is usually on the master block of the opened form. Because this trigger fires in Enter Query mode, it can populate items with values from global variables, which are then used as query criteria. This restriction applies for every other query performed on the block thereafter.

  This trigger is on the S_ORD block of the ORDERS form:

  ```
  :S_ORD.customer_id := :GLOBAL.customerid;
  ```

Assigning Global Variables in the Opened Form

- **DEFAULT_VALUE ensures the existence of globals.**
- **You can use globals to communicate that the form is running.**

Pre-Form example:

```
DEFAULT_VALUE('', 'GLOBAL.customerid');
```


Conditional Opening

Example:

```
:GLOBAL.customerid := :S_CUSTOMER.id;
IF  ID_NULL(FIND_FORM('ORDERS')) THEN
    OPEN_FORM('ORDERS');
ELSE
    GO_FORM('ORDERS');
END IF;
```

Assigning Global Variables in the Opened Form

If, for some reason, a global variable has not been initialized before it is referenced in a called form, an error is reported:

```
FRM-40815: Variable GLOBAL.productid does not exist.
```

You can provide independence, and ensure that global variables exist by using the DEFAULT_VALUE built-in when the form is opening.

Example

This Pre-Form trigger in the ORDERS form assigns a NULL value to `GLOBAL.CUSTOMERID` if it does not exist when the form starts. Because the Pre-Form trigger fires before record creation, and before all of the When-New-Object-Instance triggers, it ensures existence of global variables at the earliest point.

```
DEFAULT_VALUE('', 'GLOBAL.customerid');
```

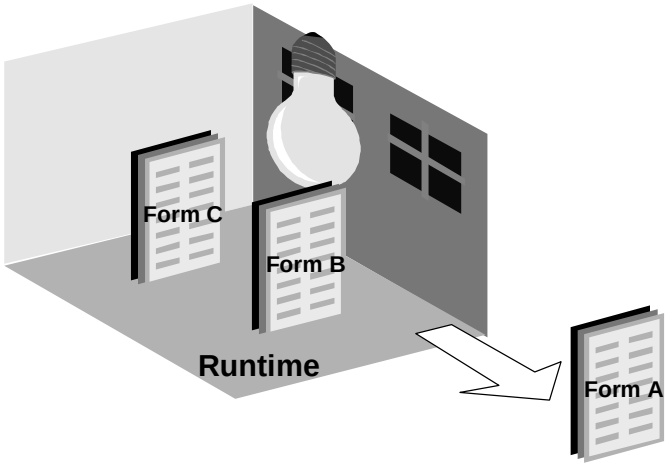
The ORDERS form can now potentially be called without the CUSTOMERS form.

Conditional Opening

Here is a variation of the When-Button-Pressed trigger on Orders_Button in the CUSTOMERS form. If the ORDERS form is already running, it simply passes control to it, using GO_FORM.

```
:GLOBAL.customerid := :S_CUSTOMER.id;
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.**

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

Closing Forms and Forms Runtime Sessions

A form may close down and pass control back to its calling form under the following conditions:

- The user presses [Exit] or selects Exit from the Action menu.
- The EXIT_FORM built-in is executed from a trigger.

If the closing form is the only form still running in the Forms Runtime session, the session will end as a result. When a multiple form session involves the OPEN_FORM built-in, it is possible that the last form to close is not the one that began the session.

Closing a Form with EXIT_FORM

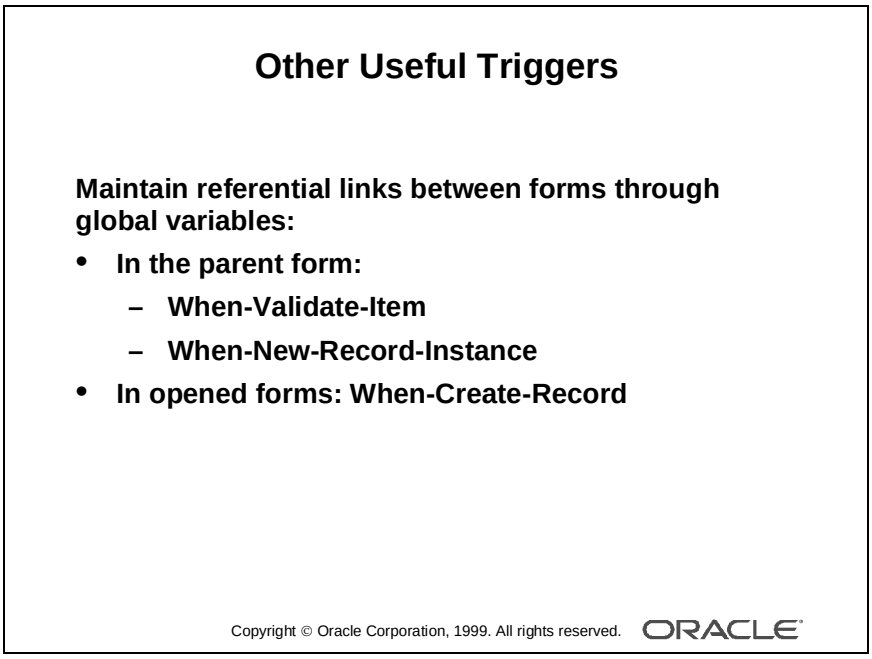
When a form is closed, Form Builder checks to see whether there are any uncommitted changes. If there are, the user is prompted with the standard alert:

```
Do you want to save the changes you have made?
```

If you are closing a form with EXIT_FORM, the default functionality is the same as described above. You can, however, make the decision to commit (save) or roll back through the EXIT_FORM built-in, so the user is not asked. Typically, you might use this built-in from a Key-Exit or When-Button-Pressed trigger.

```
EXIT_FORM(commit_mode)
```

Parameter	Description
Commit_Mode	Defines what to do with uncommitted changes in the current form: • ASK_COMMIT gives the decision to the user (the default). • DO_COMMIT posts and commits changes across all forms for the current transaction. • NO_COMMIT validates and rolls back uncommitted changes in the current form. • NO_VALIDATE is the same as NO_COMMIT, but without validation.



Other Useful Triggers When Using OPEN_FORM

Note: The system variable SYSTEM.FORM_STATUS contains the value CHANGED if there are uncommitted changes in the current form. You can use this variable to decide whether to attempt a commit in a trigger.

Other Useful Triggers When Using OPEN_FORM

Because OPEN_FORM enables the user to navigate among open forms, potentially changing and inserting records, these triggers can help keep referential key values in step across forms.

Example

In the parent form (CUSTOMERS), this assignment to GLOBAL.CUSTOMERID can be performed in a When-Validate-Item trigger on :S_CUSTOMER.Id, so that the global variable is kept up-to-date with an applied change by the user. The statement can also be issued from a When-New-Record-Instance trigger on the S_CUSTOMER block, in case the user navigates to a line item record for a different customer.

```
:GLOBAL.customerid := :S_CUSTOMER.id;
```

Example

In the opened form (ORDERS), a When-Create-Record trigger on the S_ORD block ensures that new records use the value of GLOBAL.CUSTOMERID as their default.

When items are assigned from this trigger, the record status remains NEW, so that the user can leave the record without completing it.

```
:S_ORD.customer_id := :GLOBAL.customerid;
```

Task List

- **Starting another form module**
- **Implementing multiple form functionality**
  - **Defining the details of calling and opening forms**
  - **Passing data between forms**
  - **Performing multiple form transactions**

Task List

This lesson is an introduction lesson to multiple form applications. In the course *Oracle Developer Build Forms II*, there is another lesson that covers this subject in detail. This lesson covered the following topics:

- Starting another form module: OPEN_FORM built-in
- Multiple form functionality:
  - Linking forms by using global variables
  - Closing a form by using the EXIT_FORM built-in

The course *Oracle Developer Build Forms II* covers the following topics in the lesson "Building Multiple Form Applications":

- Relevant details of calling and opening forms:
  - OPEN_FORM, CALL_FORM, NEW_FORM built-ins
  - Closing forms
  - Navigating between forms
- Passing data between forms:
  - Form parameters
  - Parameter lists
- Multiple form transactions

Summary

- **The OPEN_FORM built-in provides multiple concurrent forms in a session.**
- **Forms communicate through global variables:**
  - **Load key values in the parent form**
  - **Use global values in opened forms for When-New-Form-Instance and Pre-Query**

Summary

In this lesson, you should have learned how to open more than one form module in a Forms Runtime session, and how to pass information among forms.

The OPEN_FORM built-in provides multiple concurrent forms in a session.

Forms communicate through global variables:

- Load key values in the parent form.
- Use global values in opened forms, particularly for When-New-Form-Instance and Pre-Query.

Practice 23 Overview

This practice covers the following topics:

- **Linking ORDERS and CUSTOMERS forms by using a global variable**
- **Using built-ins to check whether the ORDERS form is running**
- **Using global variables to restrict a query in the ORDERS form**

Note

For solutions to this practice, see Practice 23 in Appendix A, “Practice Solutions.”

Practice 23 Overview

In this practice, you produce a multiple form application by linking the CUSTG*XX* and the ORDG*XX* form modules.

- Linking ORDERS and CUSTOMERS forms by using a global variable
- Using built-ins to check whether the ORDERS form is running
- Using global variables to restrict a query in the ORDERS form

Practice 23

1. In the ORDG*XX* form, create a Pre-Form trigger to ensure that a global variable called Customer_Id exists.

   Use the DEFAULT_VALUE built-in, and set the variable to NULL if it does not yet exist.

2. Add a trigger to ensure that queries on the S_ORD block are restricted by the value of GLOBAL.Customer_Id.

   Write a Pre-Query trigger on the S_ORD block that assigns the value of the global variable to the Customer_Id item.

3. Save, compile, and run the form to test that it works as a stand-alone.

4. In the CUSTG*XX* form, create a CONTROL block button called Orders_Button.

5. Define a trigger for CONTROL.Orders_Button that initializes GLOBAL.Customer_Id with the current customer's ID, then opens the ORDG*XX* form, passing control to it.

   Use OPEN_FORM. Make sure that the internal name of the ORDG*XX* form (in the Object Navigator) matches the filename of the form.

6. Save and compile each form, then run the application to test them.

   Run the CUSTG*XX* form, then open the ORDG*XX* form from the new button. Are the relative positions of the two forms adequate?

7. Change the window location of the ORDG*XX* form, if required.

8. Alter the Orders_Button trigger in CUSTG*XX* so that it uses GO_FORM to pass control to ORDG*XX* if the form is already running. Use the FIND_FORM built-in for this purpose.

   Remember that you need to use the module name in the GO_FORM built-in, and the filename in the OPEN_FORM built-in.

9. Write a When-Create-Record trigger on the S_ORD block that uses the value of GLOBAL.Customer_Id as the default value for S_ORD.Customer_Id.

10. Add code to the CUSTG*XX* form so that GLOBAL. Customer_Id is updated when the current Customer_Id changes.

A

Practice Solutions

Practice 1 Solutions

1. Invoke Project Builder and select Go to the Project Navigator.

   No formal solution.

2. Launch Form Builder, and select “Open an existing form” from the Welcome page.

   No formal solution.

3. Open the `Orders.fmb` form module from the Open Dialog window.

   No formal solution.

4. Change your preferences so that when you open or save a file Form Builder gives you the option of saving the file to the filesystem or the database.

   Select Tools—>Preferences from the default menu system. Click the Access tab in the Preferences dialog box.

   Click the Ask option. Click OK.

5. Try to open the `Customers.fmb` form module. Notice that the module access dialog box displays. Press Cancel.
   Modify your preferences so that Form Builder will access the filesystem only.

   Select Tools—>Preferences from the default menu system. Click the Access tab in the Preferences dialog box.

   Click the File option. Click OK.

6. Close the ORDERS form.

   No formal solution.

7. Open the `Summit.fmb` form module.

   No formal solution.

8. Expand the Data Blocks node.

   No formal solution.

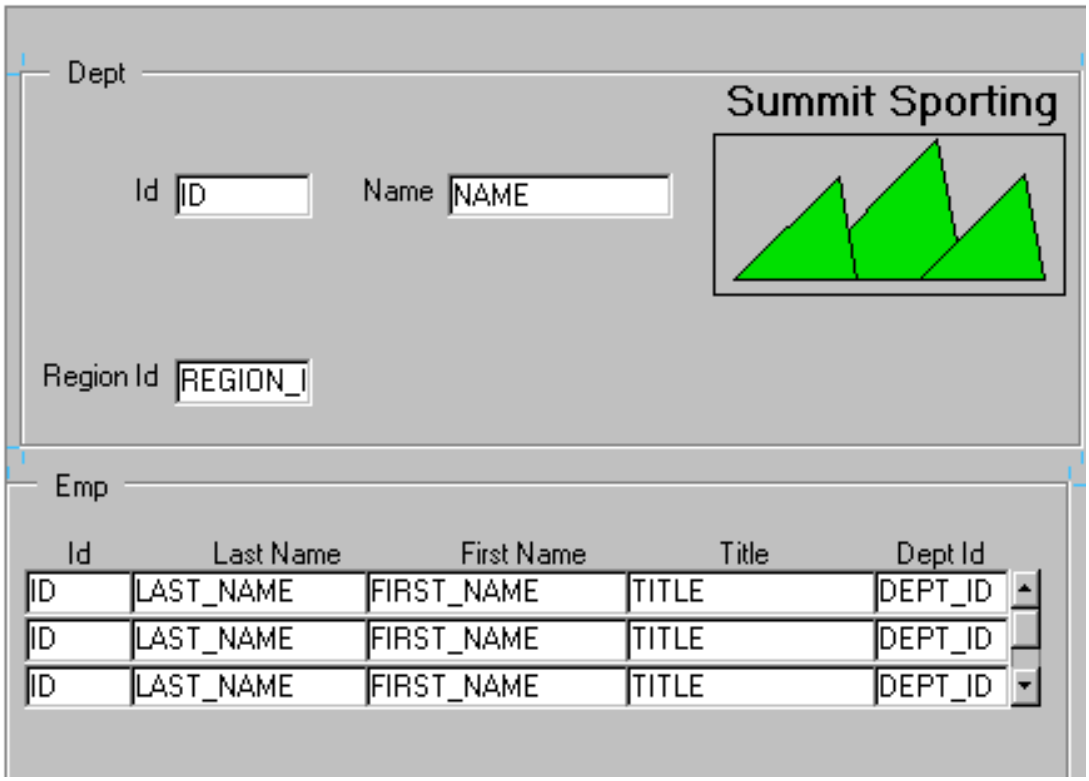
9. Expand the Database Objects node. If you cannot expand the node, connect to the database and try again. What do you see below this node?

   No formal solution.

10. Collapse the Data Blocks node.

    No formal solution.

11 Change the layout of the `Summit.fmb` form module to match the screenshot shown below. At the end, save your changes, and exit Form Builder.



a Invoke the Layout Editor.
Select Tools—>Layout Editor from the default menu system.

b Move the three summit shapes to the top-right corner of the layout. Align the objects along the bottom edge.
Shift-click each of the three shapes to select them together. Move them to the top-right corner of the layout.

c Select the summit shape in the middle and place it behind the other two shapes.
Select the middle summit shape, and select Arrange—>Send to Back.

d Draw a box with no fill around the summit shapes.
Select the Rectangle tool from the Tool Palette and draw a rectangle around the three summit shapes. With the rectangle still selected, click the Fill Color tool and select No Fill.

e Add the text Summit Sporting on top of the box.
Select the text tool from the Tool Palette and enter the text above the rectangle. Choose a suitable font size and style.

f Move the Name, Id, and Region_Id items to match the screenshot.
Select and move the Name item to the right with the mouse. Select and move the Id item up. Shift-click the Id and Region_Id items to select them together. Click the Align Left button to align these two items.

g Move the First_Name item up to align it at the same level as the Last_Name item.
Select the First_Name and Last_Name items together, and click the Align Top button.

h Resize the scroll bar to make it the same height as the three records in the Emp block.
Select the scroll bar and resize it with the mouse.

i Save the form module, and exit Form Builder.
In the Object Navigator, select File—>Save, then File—>Exit.

Optional Practice

12 Your instructor may ask you to do the following exercise to prepare some forms for the next practice session:

a Launch Project Builder.
No formal solution.

b Select “Open an existing project” from the Welcome page.
No formal solution.

c Select the `Summit.upd` project in the Lab folder of the directory.
No formal solution.

d In the Project Builder - Project View window, expand the Projects node.
No formal solution.

e Expand the Oracle Developer: Build Forms I node.
No formal solution.

f Open the Property Palette, and specify the Project Directory as well as the user ID and password.
With the Oracle Developer: Build Forms I node still selected, select Tools—>Property Palette from the default menu.
Set the Project Directory property to the Lab directory.
Your instructor will give you the pathname.
Specify values for the Username, Password, and Database properties. Your instructor will give you this information.
Close the Property Palette.

g Expand the Summit Application node in the Project Builder - Project View window.
No formal solution.

h Expand the Form Builder document node.
No formal solution.

i Select the `Orders.fmb` entry.
No formal solution.

j Select Project—>Build All from the default menu system. This creates the run-time files you need for the next practice session.
No formal solution.

k Exit Project Builder.
Select File—>Exit.

Practice 2 Solutions

Queries

1. Start Forms Runtime by using the form module Customers.
   No formal solution.
2. Select Help—>Keys from the menu.
   No formal solution.
3. Execute an unrestricted query.
   Select Query—>Execute, or press [Execute Query], or click the Execute Query Toolbar button.
   Press [Up] and [Down] to browse through the records returned.
4. Execute a restricted query to retrieve the “Womansport” record.
   Put the form module in Enter Query mode (press [F7] or select Query—>Enter from menu).
   Notice that the status line displays the words `ENTER QUERY`.
   Move to the Name item and enter the search value `Womansport`.
   Execute the query (press [F8] or select Query—>Execute from menu).
   Notice that only one record is retrieved.
5. Execute a restricted query to retrieve customers with a Sales Rep ID greater than 13.
   Put the form module in Enter Query mode.
   Click the Billing Tab to move to the Sales_Rep_ID item and enter the search criteria `>13`.
   Execute the query.
6. Try each of these restricted queries:
   - Retrieve all cities starting with San.
     Select Query—>Enter.
     Type `San%` in the City item.
     Select Query—>Execute.
   - Retrieve all those customers based in the USA with a credit rating of Excellent.
     Select Query—>Enter.
     Enter `Excellent` in the Credit Rating item.
     Select Query—>Execute.

7 Display the customer details for Big John's Sports Emporium and click the Orders button to move to the Orders form module.
Press [Next Record] until you see Big John's Sports Emporium.
Click the Orders button.

8 Click the Image Off button and notice that the image item no longer displays. Click the Image On button and notice that the image item displays.
No formal solution.

9 Query only those orders with a payment type of Credit.
Select Query—>Enter.
Select the Credit radio button.

10 Move to the first record in the Item block and click the Stock button. The Inventory block displays in a separate window. Execute a query to get stock information.
No formal solution.

Inserting, Updating, and Deleting Records

11 Insert a new record in the ORDER block, as detailed below.
Move to the ORDER block and select Record—>Insert, or press insert button of the Toolbar.
Notice that some items are already populated with default values. Enter the following:

Item	**Value**
Date Shipped	Today's date (*DD-MON-YYYY*)
Payment Type	Cash (Radio group button)
Order Filled	No (Unchecked)

12 Insert a new record in the ITEM block.
Move to the ITEM block and enter the following:

Item	**Value**
Product ID	`50530`
Quantity	`2`

13 Save the new records.
Select Action—>Save or click the Save button.

14 Update the order that you have just placed and save the change.
Change the Date Shipped to next Monday and save the change.

15 Attempt to delete the order that you have just placed. What happens?
Move to the ORDER block and select Record—>Remove.

16 Delete the line item for your order and save the change**.**
Move to the Item block and select Record—>Remove. Click the Save button.

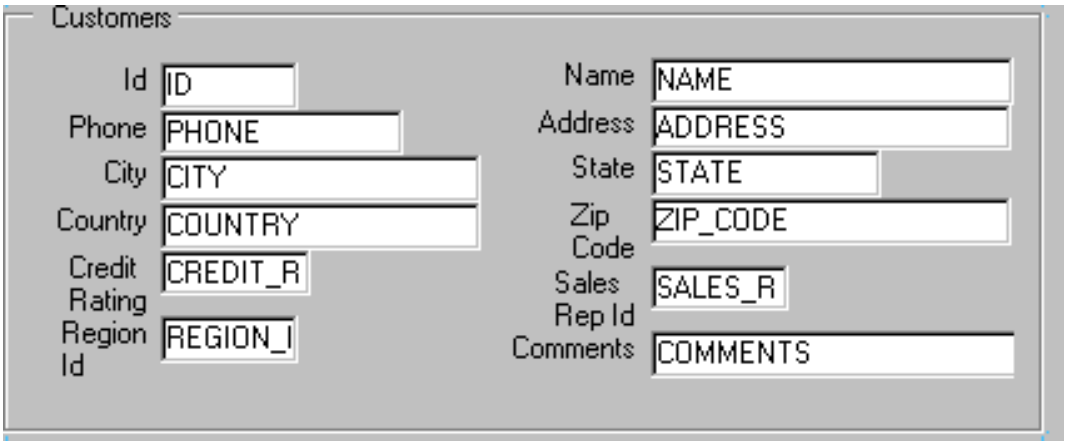
17 Now attempt to delete your order and save the change.
Move to the ORDER block and select Record—>Remove. Click the Save button.

18 Exit the run-time session.
No formal solution.

Practice 4 Solutions

1. Create a new form module.
   Create a new single block by using the Data Block Wizard.
   Base it on the S_CUSTOMER table and include all columns.
   Display the S_CUSTOMER block on a new content canvas called CV_CUSTOMER and show just one record at a time. Set the frame title to Customers.
   If you are not already in Form Builder, run Form Builder and create a new form module by using Welcome Wizard. If you are already in Form Builder, then create a new form module by selecting File—>New—>Form or by clicking the Create Toolbar button.
   Use Tools—>Data Block Wizard to create a block.
   Select the block type as Table or View.
   Set the Table or View field to S_CUSTOMER.
   Click the Refresh button and click the >> button to include all columns.
   Click the Next button, and select "Create the data block, then call the Layout Wizard" option, and click the Finish button.
   In the Layout Wizard, select a new canvas and make sure the Type field is set to "Content."
   Include all items.
   Set the Style to Form.
   Set the Frame Title to Customers, and click Finish.
   In Object Navigator, rename the canvas as CV_CUSTOMER:
   - **Select the canvas.**
   - **Click the name.**
   - **The cursor changes to an I-beam; edit the name.**
   - **Press [Return].**
2. Save the new module to a file called CUSTG*XX*, where *XX* is the group number that your instructor has assigned to you.
   Run your form module and execute a query.
   Navigate through the fields. Exit run time and return to Form Builder.
   No formal solution.
3. Change the form module name in the Object Navigator to CUSTOMERS.
   Select the form module. Click the name. The cursor changes to an I-beam. Edit the name, then press [Return].

4 In the Layout Editor, reposition the items so that the canvas resembles the screenshot below.
Hint: First resize the canvas and the frame.
Reposition the items by dragging and dropping them.



5 Create a new form module.
Create a new block by using the Data Block Wizard.
Base it on the S_ORD table and include all columns except TOTAL.
Display the S_ORD block on a new content canvas called CV_ORDER and show just one record at a time. Use a form style layout. Set the frame title to Orders.
Create a new form module by selecting File—>New—>Form or by clicking the Create Toolbar button.
Use Tools—>Data Block Wizard to create a block.
Select the block type as Table or View.
Set the Table or View field to S_ORD.
Click the Refresh button and include all columns except Total.
Click the Next button, and select "Create the data block, then call the Layout Wizard" option, and click Finish.
In the Layout Wizard select a new canvas and make sure the Type filed is set to "Content."
Include all items.
Set Style to Form.
Set Frame Title to Orders, and click Finish.
In Object Navigator rename the canvas as CV_ORDER.

6. Create a new block by using the Data Block Wizard.
   Base the block on the S_ITEM table and include all columns.
   Create a relationship and select the master block as S_ORD.
   Display all items except ORD_ID on the CV_ORDER canvas.
   Display six records in this detail block on the same canvas as the master block.
   Use a tabular style layout and include a scroll bar.
   Change the order of the blocks in the Object Navigator, moving the S_ITEM block after the S_ORD block. Set the frame title to Items.
   In the same module, create a new block by using Tools—>Data Block Wizard.
   Select block type as Table or View.
   Set the Base Table to S_ITEM.
   Include all columns.
   Click the Create Relationship button.
   Select S_ORD block as the master block.
   Use the Layout Wizard to create a layout.
   Select Canvas as CV_ORDER.
   Include all items except Ord_ID.
   Set the Style to Tabular.
   Set the Frame Title to Items.
   Set the Records Displayed to 6.
   Select the Display Scrollbar check box.
   In the Object Navigator, drag and drop the S_ITEM block to a position below the S_ORD block.
7. Save the new module to a file called ORDG*XX*, where *XX* is the group number that your instructor has assigned to you.
   No formal solution.
8. Create a new block based on S_INVENTORY (do not create any relationships with other blocks at this time) to display on a different canvas.
   Base it on the S_INVENTORY table, exclude the OUT_OF_STOCK_EXPLANATION column from the definition.
   Display four records in this block and ensure that they display on a new content canvas called CV_INVENTORY.
   Use a tabular style layout, and include a scroll bar.
   In the Object Navigator move the S_INVENTORY block after the S_ITEM block. Set the frame title to Stock.
   Do not create any relationships between blocks at this stage.

In the same module, create a new block by using Tools—>Data Block Wizard.
Select block type as Table or View.
Set the Base Table to S_INVENTORY.
Include all columns except the OUT_OF_STOCK_EXPLANATION column.
Use the Layout Wizard to create a layout.
Select a New Canvas.
Include all items.
Set the Style to Tabular.
Set the Frame Title to Stock.
Set the Displayed Records to 4.
Select the Display Scrollbar check box.
In Object Navigator rename the canvas to CV_INVENTORY.
In the Object Navigator move the S_INVENTORY block after the S_ITEM block.

9 Create a relation called S_Item_S_Inventory explicitly between the S_Item and S_Inventory blocks.
Ensure that line item records can be deleted independently of any related inventory.
Set the coordination so that the Inventory block is not queried until you explicitly execute a query.
Create the relation by selecting the word "RELATIONS" in the S_ITEM block in the Object Navigator, and clicking the Create button. The New Relation dialog box appears. Select S_INVENTORY as the detail block. Select the Isolated radio button. Select the Deferred check box, and then uncheck the Auto Query check box. Enter the join condition `s_item.product_id = s_inventory.product_id` and click the OK button.

10 On the S_ITEM block change the prompt for the Quantity Shipped item to Shipped by using the reentrant Layout Wizard. First select the relevant frame in the Layout Editor, then use the Layout Wizard.
Select the frame for the S_ITEM block under the CV_ORDER canvas in the Object Navigator or in the Layout Editor, and select Tools—>Layout Wizard from the menu.
Select the Items tab page.
Change the prompt for the Quantity Shipped item to Shipped, and click Finish.

11 In the S_INVENTORY data block, change the prompt for Amount in Stock to In Stock by using the Layout Wizard.
In Object Navigator or in the Layout Editor, select the frame that is associated with S_INVENTORY data block.
Select Tools—>Layout Wizard from the menu.
Select the Items tab page and change the prompt for Amount in Stock to In Stock, and click Finish.

12 Run your form module.
Execute a query.
Navigate through the blocks so that you see the S_INVENTORY block.
Exit run time and return to Form Builder.
No formal solution.

13 Change the form module name in the Object Navigator to ORDERS and save.
No formal solution.

Practice 5 Solution

CUSTG*XX* Form

1. Create a control block in the CUSTG*XX* form.
   Create a new block manually, and rename this block CONTROL.
   Set the Database Data Block, Query Allowed, Insert Allowed, Update Allowed, and Delete Allowed Database properties to No. Set the Query Data Source Type property to None. Leave other properties as default.
   Move the CONTROL block after the S_CUSTOMER block.
   Select Data Blocks node in the Object Navigator.
   Click the Create icon in Object Navigator, or select
   Navigator—>Create option from menu to create a new Data Block.
   Select the "Build a new data block manually" option.
   Rename this new data block as CONTROL.
   Click the right mouse button on this block, and open the Property Palette.
   Find Database category in Property Palette.
   Set the Database Data Block, Query Allowed, Insert Allowed, Update Allowed, and Delete Allowed properties to No.
   Set the Query Data Source Type property to None.
   Leave other properties as default.
   In the Object Navigator move the CONTROL block after the S_CUSTOMER block.
2. Ensure that the records retrieved in the S_CUSTOMER block are sorted by the customer's ID.
   In the Property Palette for the S_CUSTOMER block, set the ORDER BY Clause property to ID.
3. Set the frame properties for the S_CUSTOMER block as follows:
   Remove the frame title, and set the Update Layout property to Manually.
   In the Layout Editor select the frame that covers S_CUSTOMER block and open Property Palette. Remove the Title property value and set the Update Layout property to Manually.
4. Save and run the CUSTG*XX* form.
   Test the effects of the properties that you have set.
   No formal solution.

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

ORDG*XX* Form

5 Create a CONTROL block in the ORDG*XX* form.
Create a new block manually, and rename this block CONTROl.
Set the Database Data Block, Query Allowed, Insert Allowed, Update Allowed, and Delete Allowed database properties to No. Set the Query Data Source Type property to None. Leave other properties as default.
Position the CONTROL block after the S_INVENTORY block in the Object Navigator.
Select Data Blocks node in the Object Navigator.
Click Create icon in Object Navigator, or select Navigator—>Create option from menu to create a new Data Block.
Select the "Build a new data block manually" option.
Rename this new data block CONTROL.
Click the right mouse button on this block, and open the Property Palette.
Find the Database category in Property Palette.
Set the Database Data Block, Query Allowed, Insert Allowed, Update Allowed, and Delete Allowed properties to No.
Set the Query Data Source Type property to None.
Leave other properties as default.
In the Object Navigator move the CONTROL block after the S_INVENTORY block.

6 Ensure that the current record is displayed differently from the others in both the S_ITEM and S_INVENTORY blocks.
Create a visual attribute called Current_Record.
Using the Color Picker, set the Foreground Color to White and the Background Color to Dark Cyan. (If these colors are not available on your window manager, use other colors instead.) Using the Pattern Picker, choose any fill pattern. Using the Font Picker, set the font to MS Serif italic 10 point. (If that font is not available on your window manager, use any available font.)
Use the multiple selection feature on both data blocks to set the relevant block property to use this visual attribute.
In the Object Navigator, select the Visual Attributes node, and create a new visual attribute.
In the Property Palette set the Name property to CURRENT_RECORD.
Select the Foreground Color property and press the button. The Color Picker dialog box is displayed. Set Foreground Color to White. Repeat the process to set the Background Color to Dark Cyan.
Select the Fill Pattern property and press the button. The Pattern Picker dialog box is displayed. Select any pattern.
In the Property Palette, select the Font category heading. (Do not select any of the properties under the Font category heading.) Press the More button and the Font dialog box appears. Select MS Serif italic 10 point, and click the OK button.
In the Object Navigator to use multiple selection feature, select both of the S_ITEM and the S_IVENTORY blocks by pressing the [Shift] key and the left mouse button, then open the Property Palette.
Set the Current Record Visual Attribute Group property to CURRENT_RECORD.

7 For the S_ITEM block change the number of records displayed to 4 and resize the scroll bar accordingly.
In the Object Navigator select S_ITEM block and open Property Palette. Set the Number of Records Displayed property to 4.

8 Ensure that the records retrieved in the S_ITEM block are sorted by the ITEM_ID.
For the S_ITEM data block, set the ORDER By Clause property to ITEM_ID.

9 Set the property that causes automatic navigation to Next Record, when the user uses [Next Item] to exit the last item of a record in the S_ITEM block.
For the S_ITEM block, set the Navigation Style to Change Record.

10 Set the frame properties for all blocks as follows:
Remove the frame title and set the Update Layout property to Manually.
In the Object Navigator, select frames under the Canvases node and open Property Palette. Remove the Title property value and set the Update Layout property to Manually.

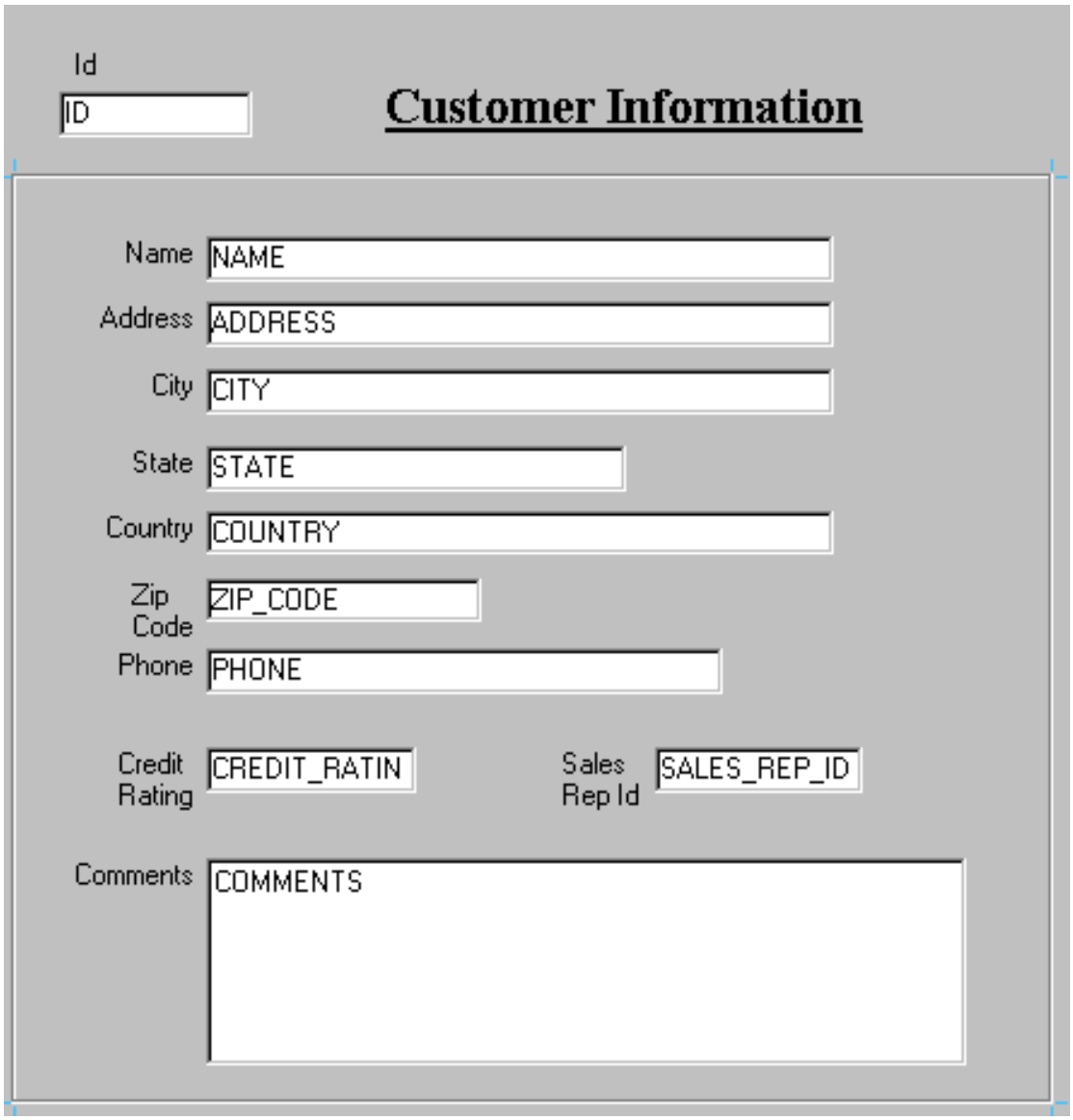
11 Save, compile, and run the ORDG*XX* form.
Test the effects of the properties that you have set.
No formal solution.

Practice 6 Solutions

CUSTG*XX* Form

1. Remove the Region ID item.
   In the Layout Editor, select and delete Region_Id item.
2. Make sure that the Comments item accepts multiline text to display.
   For the Comments item, set Multi-line to Yes and set Height to 65.
3. Automatically display a unique, new customer number for each new record and ensure that it cannot be changed**.**
   Use the S_CUSTOMER_ID sequence.
   For ID, set Initial Value to :sequence.s_customer_id.nextval
4. In the CUSTG*XX* form, resize and reposition the items. Reorder the items in the Object Navigator. Use the screenshot and the table below as a guide.
   Resize items by setting the width according to the following property table.

Item	**Suggested Width**
ID	60
NAME	195
ADDRESS	195
CITY	195
STATE	130
COUNTRY	195
ZIP_CODE	85
PHONE	160
CREDIT_RATING	65
SALES_REP_ID	65
COMMENTS	236



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

ORDG*XX* Form

6 In the S_ORD block, create a new text item called Customer_Name. Ensure that Customer_Name is not associated with the S_ORD table. Do not allow insert, update, or query operations on this item, and make sure that navigation is possible only by means of the mouse. Set the Prompt text to Customer Name. Display this item on CV_ORDER canvas.
Create a text item and name it Customer_Name.
Item Type should be set to Text Item.
Set the Database Item, Insert Allowed, Update Allowed, Query Allowed, and Keyboard Navigable to No.
Set the Prompt to Customer Name.
Set the Canvas property to CV_ORDER.

7 In the S_ORD block, create a new text item called Sales_Rep_Name. Ensure that Sales_Rep_Name is not associated with the S_ORD table. Do not allow insert, update, or query operations on this item and make sure that navigation is possible only by means of the mouse. Set the Prompt text to Sales Rep Name. Display this item on the CV_ORDER canvas.
Create a text item and name it Sales_Rep_Name.
Item Type should be set to Text Item.
Set the Database Item, Insert Allowed, Update Allowed, Query Allowed, and Keyboard Navigable to No.
Set the Prompt to Sales Rep Name.
Set the Canvas property to CV_ORDER.

8 Set the relevant property for Date_Ordered, so that it displays the current date whenever a new record is entered.
For Date_Ordered, set Initial Value to $$date$$.

9 In the S_ITEM block, create a new text item called Item_Total. Ensure that Item_Total is not associated with the S_ITEM table. Do not allow insert, update, or query operations on this item and make sure that navigation is possible only by means of the mouse. Allow numeric data only and display it by using a format of 999G990D99.
Set the Prompt text to Item Total. Display this item on the CV_ORDER canvas**.**
Item Type should be set to Text Item.
Set the Database Item, Insert Allowed, Update Allowed, Query Allowed, and Keyboard Navigable to No.

Set the Data Type to Number.
Set the Format Mask to 999G990D99.
Set the Prompt to Item Total.
Set the Canvas property to CV_ORDER.

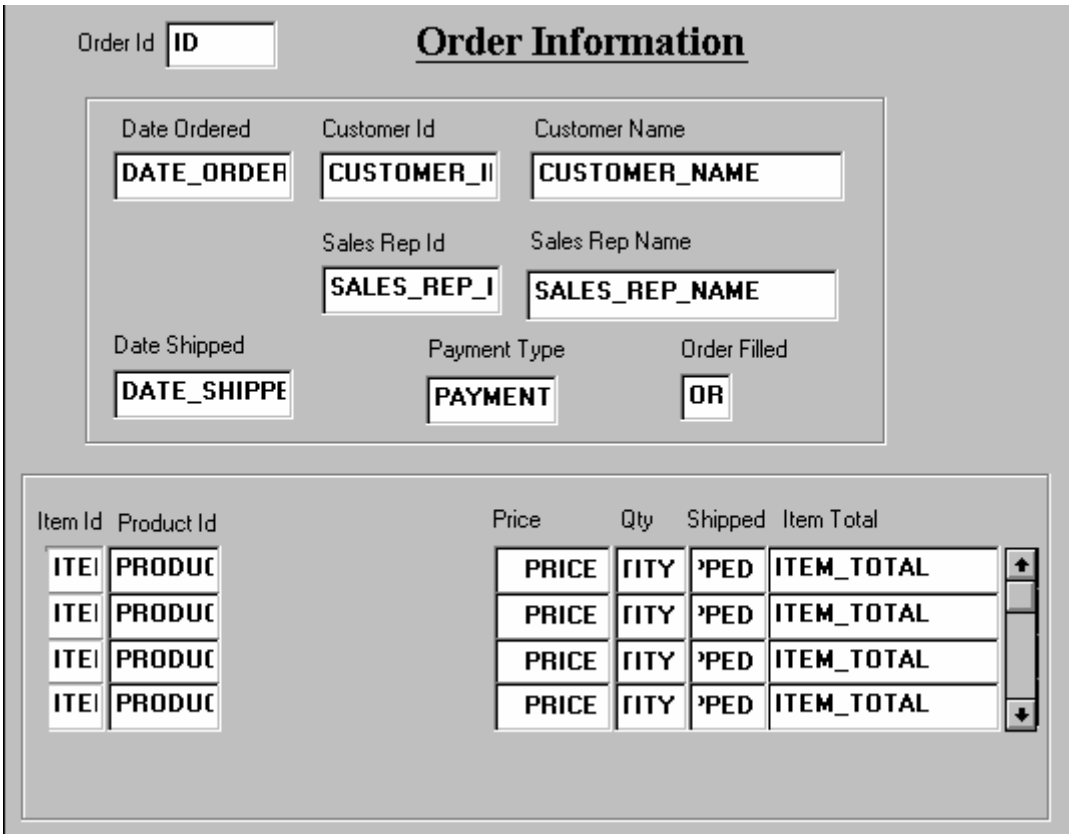
10 Justify the values of Price, Quantity, and Quantity_Shipped to the right.
For each of the items, set Justification to Right.

11 Alter the Quantity_Shipped item, so that navigation is possible only by means of the mouse, and updates are not allowed.
For Quantity_Shipped, set Keyboard Navigable and Update Allowed to No.

12 In the ORDG*XX* form, resize and reposition the items according to the screenshot and the table below.
Resize items by setting the width in the corresponding property sheet.
Drag and drop items to reposition.

S_ORD Block Items	**Suggested Width**
ID	40
DATE_ORDERED	66
CUSTOMER_ID	66
CUSTOMER_NAME	116
SALES_REP_ID	66
SALES_REP_NAME	116
DATE_SHIPPED	66
PAYMENT_TYPE	48
ORDER_FILLED	18

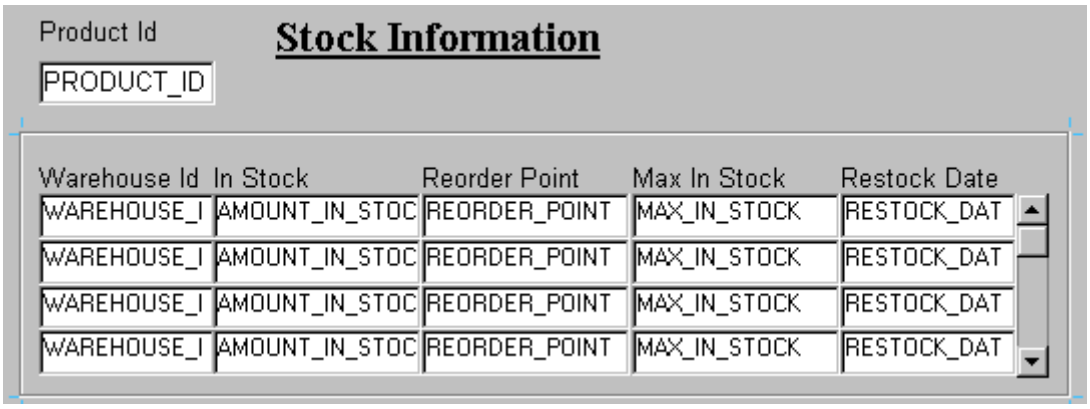
S_ITEM Block Items	**Suggested Width**
ITEM_ID	20
PRODUCT_ID	40
PRICE	42
QUANTITY	26
QUANTITY_SHIPPED	26
ITEM_TOTAL	86



13 In the S_INVENTORY block, alter the number of instances of the Product_ID, so that it is displayed just once.
In the property sheet for Product_ID, set Number of Items Displayed to 1.

14 Arrange the items and boilerplate on CV_INVENTORY, so that it resembles the screenshot.

Hint: Set the Update Layout property for the frame to Manually.
No formal solution.



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

Practice 7 Solution

1 In the ORDG*XX* form, create an LOV to display product numbers and descriptions to be used with the Product_Id item in the S_ITEM block. Use the S_PRODUCT table, Id, and Name columns.
Assign a title of Products to the LOV. Assign a column width of 25 for ID, and assign the LOV a width of 200 and a height of 250. Position the LOV 30 pixels below and to the right of the upper lefthand corner. For the ID column, set the return item to S_ITEM.PRODUCT_ID. Attach the LOV to the Product_Id item in the S_ITEM block. Change the name of the LOV to PRODUCTS.
Create a new LOV. Press the OK button to use the LOV Wizard. Select the "New Record Group based on a query" radio button, and click the Next button.
In the SQL Query Statement, enter:

```
SELECT      id, name
FROM        s_product
ORDER BY    name
```

and click the Next button.

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

Set the display width for ID to 25 and click the Next button.

Enter the title Products. Assign the LOV a width of 200 and a height of 250. Select the "No, I want to position it manually" radio button, and set both Left and Top to 30. Click the Next button.

Click the Next button to accept the default advanced properties.

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

In the Object Navigator, change the name of the new LOV to PRODUCTS_LOV.

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

3 In the CUSTG*XX* form, create an LOV to display sales representatives' numbers and their names, using the LOV Wizard. Use the S_EMP table, Id, First_Name, and Last_Name columns. Concatenate the First_Name and the Last_Name columns and give an alias such as Name.
Assign a title of Sales Representatives to the LOV. Assign a column width of 20 for ID, and assign the LOV a width of 200 and a height of 250. Position the LOV 30 pixels below and to the right of the upper lefthand corner. For the ID column, set the return item to S_CUSTOMER.SALES_REP_ID. Attach the LOV to the Sales_Rep_Id item in the S_CUSTOMER block.
Change the name of the LOV to SALES_REP_LOV.
Create a new LOV. Press the OK button to use the LOV Wizard. Select the "New Record Group based on a query" radio button, and click the Next button.
In the SQL Query Statement, enter:

```
Select     id, first_name || ' ' || last_name Name
From       s_emp
Where title = 'Sales Representative'
Order By   last_name
```

and click the Next button.
Click the >> button and then the Next button to select both of the record group values.
With the ID column selected, click the "Look up return item" button. Select S_CUSTOMER.SALES_REP_ID and click the OK button. Set the display width for ID to 20 and click the Next button. Enter the title Sales Representatives. Assign the LOV a width of 200 and a height of 250. Select the "No, I want to position it manually" radio button, and set both Left and Top to 30. Click the Next button.
Click the Next button to accept the default advanced properties.
Click the > button and then the Finish button to create the LOV and attach it to the Sales_Rep_Id item.
In the Object Navigator, change the name of the new LOV to SALES_REP_LOV.

4. In the CUSTG*XX* form, create an editor and attach it to the Comments item. Set the title to Comments, the background color to gray, and the foreground color to yellow.
   Create a new editor. Change the name to Comments_Editor.
   Set the X Position and Y Position properties to 175.
   Set the Width property to 450 and the Height property to 250.
   Set the title to Comments, Background Color property to gray, and the Foreground Color property to yellow.
   In the Property Palette of the Comments item, set the Editor property to Comments_Editor.
5. Save, compile, and run the forms to test the changes. Resize the window if necessary.
   No formal solution.

Practice 8 Solutions

1. In the CUSTG*XX* form, convert the Credit_Rating text item into a pop-up list item.
   Add list elements of Poor, Good, and Excellent to represent database values of POOR, GOOD, and EXCELLENT.
   Display any other values as Poor.
   Ensure that new records display the initial value GOOD.
   Resize the poplist in the Layout Editor, so that the elements do not truncate at run time.
   For Credit_Rating, set Item Type to List Item.
   Set the Initial Value to GOOD.
   Set the List Style to Poplist.
   Set the Mapping of Other Values to Poor.
   Select the Elements in List property and click the More button to invoke the List Elements dialog box.
   Enter the elements Poor, Good, and Excellent.
   Enter the corresponding database values in the List Item Value box.
   Click OK to accept and close the dialog.
   Open the Layout Editor and resize the item so that elements do not truncate.
2. Save, compile, and run the form to test the changes.
   No formal solution.
3. In the ORDG*XX* form, convert the Order_Filled text item into a check box.
   Set the checked state to represent the base table value of Y and the unchecked state to represent N.
   Ensure that new records are automatically assigned the value N.
   Allow only those records with Order_Filled values of Y or N to display.
   Remove the existing prompt and set label as Order Filled.
   In the Layout Editor resize the check box so that its label is displayed to the right.
   For Order_Filled, set Item Type to Check Box.
   Set the Value when Checked to Y and Value when Unchecked to N.
   Set the Check Box Mapping of Other Values to Not Allowed.
   Set the Initial Value to N.
   Delete the Prompt property, and set Label property to Order Filled.
   Open the Layout Editor and resize the check box so that its label displays to the right.

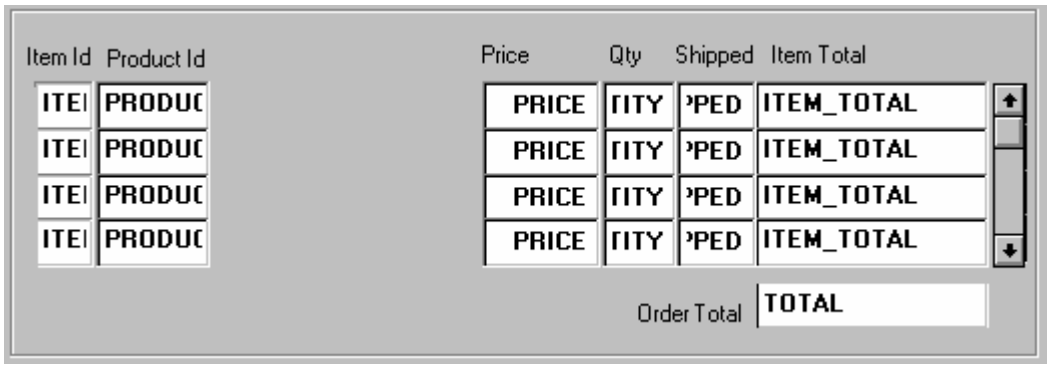
4. Convert the Payment_Type text item into a radio group.
   Add radio buttons for Cash and Credit to represent database values of CASH and CREDIT.
   Define access keys of S for cash and T for credit.
   Add text Payment type to describe the radio group's purpose.
   Set Label to Cash for Cash radio button and Credit for Credit radio button.
   Ensure that new records display the default of Cash.
   For Payment_Type, set Item Type to Radio Group.
   Set the Initial Value to CASH.
   Expand the Payment_Type node.
   The node Radio Buttons is displayed.
   Create two buttons; name them Cash_Button and Credit_Button.
   For the first button, set Access Key to S, Label to Cash, and Radio Button Value to CASH. For the second button, set Access Key to T, Label to Credit, and Radio Button Value to CREDIT.
   Use the Layout Editor to position the radio buttons so that both can be seen.
   In the Layout Editor, create boilerplate text to identify radio buttons as Payment Type.
5. Reorder the items of the S_ORD block in the Object Navigator. Use the order of the items in the Layout Editor as a guide.
   Drag-and-drop the items of the S_ORD block according to the visual order of this block.
6. Save, compile, and run the forms to test the changes.
   No formal solution.

Practice 9 Solutions

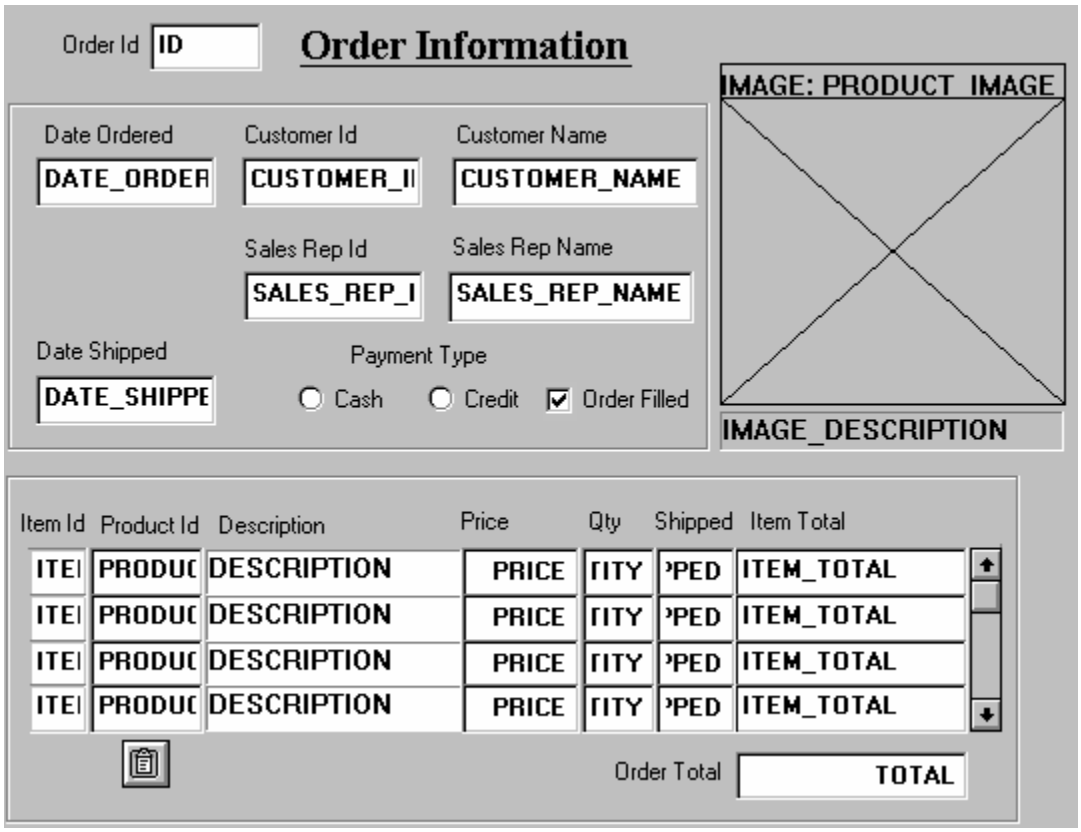
1. In the S_ITEM block of the ORDG*XX* form, create a display item called Description. Set the Prompt property to Description and display the prompt above the item.
   Open the Layout Editor, ensure the block is set to S_ITEM, and select the Display Item tool.
   Place the Display Item to the right of the Product_Id.
   Set the Name property to Description.
   Set the Maximum length to 50.
   Set the Width to 133.
   Set the Database Item to No.
   Set the Prompt property to Description and the Prompt Attachment Edge property to Top.
2. Create a single-record image item called Product_Image in the S_ITEM block of the ORDG*XX* form.
   Display the Layout Editor.
   Ensure the block is set to S_ITEM.
   Select the Image Item tool.
   Click and drag a box and place it so it matches the picture below.
   Display the properties window for the image.
   Change the name to Product_Image.
   Include these properties for the image item:
   Bevel: Lowered, Keyboard Navigable: No, Sizing Style: Adjust, Database Item: No, Number of Items Displayed: 1.
3. Create another display item, Image_Description, in the S_ITEM block. This should synchronize with the Description item. Set the Maximum Length property to the same value as the Description item.
   Display the Layout Editor.
   Ensure the block is set to S_ITEM.
   Select the Display Item tool.
   Place the Display Item to just below the Product_Image.
   Set the Synchronize with Item to Description.
   Set the Maximum Length property to 50.
   Set the Database Item to No.
   Set the Number of Items Displayed to 1.
   Set the Width to 133.
   Change the name to Image_Description.

4. In the CONTROL block of the ORDG*XX* form, create an iconic button called Product_LOV_Button. Use the `list.ico` file (do not include the `.ico` extension). Set the Keyboard Navigable property and the Mouse Navigate property to No.
   Display the Layout Editor and ensure the block is set to CONTROL.
   Select the Push Button tool.
   Create a push button and place it close to the Product_Id.
   Resize the push button.
   Set the Name to Product_LOV_Button.
   Set the Iconic to Yes, and set the Icon File Name to List.
   Set the Keyboard Navigable property to No.
   Set the Mouse Navigate property to No.
5. To display item total information, set the following properties for the Item_Total item in the S_ITEM block:
   Set the Justification property to right.
   Set the Calculation Mode property to Formula.
   Set the Formula property to :S_ITEM.quantity_shipped * :S_ITEM.price.
   Set the Keyboard Navigable property to No.
   In the Object Navigator select the Item_Total item in the S_ITEM block, and open the Property Palette.
   Set the Justification property to right.
   Set the Calculation Mode property to Formula.
   Set the Formula property to :S_ITEM.quantity_shipped * :S_ITEM.price.
   Set the Keyboard Navigable property and the Mouse Navigate property to No.
6. To display total of the item totals create a new nondatabase item in the S_ITEM block.
   Set the position, size and prompt properties according to the screenshot.
   Set the format mask property to 9G999G990D99.
   Set the Justification property to right.
   Set the Number of Items Displayed property to 1.
   Make S_ITEM.total a summary item and display summaries of the item_total values in the S_ITEM block. Ensure that you have to set the Query All Records property to Yes for the S_ITEM block.

Set the Keyboard Navigable property to No.
In the Object Navigator create a new text item in the S_ITEM block.
Set the Name property to TOTAL, and the Prompt property to Order Total.
Set X Position, Y Position, and Width properties according to the screenshot.
Set the Canvas property to CV_ORDER.
Set the Data Type property to Number.
Set the format mask property to 9G999G990D99.
Set the Justification property to right.
Set the Number of Items Displayed property to 1.
Set the Query All Records property to Yes for the S_ITEM block; that is necessary for summary items.
For S_ITEM.total item, set Database item property to No, Calculation Mode property to Summary, and Summary Function property to Sum.
Set the Summarized Block property to S_ITEM and Summarized Item property to item_total.
Set the Keyboard Navigable property and the Mouse Navigate property to No.



7 Save, compile, and run the forms to test the changes. Change the window size if necessary.
No formal solution.



8 Perform a query in the ORDG*XX* form to ensure that the new items do not cause an error. Did you remember to switch off the Database Item property for items that do not correspond to columns in the base table?
No formal solution.

9 Create an iconic button similar to the one created in Question 4, in the CONTROL block of form CUSTG*XX*. Use the `list.ico` file (do not include the `.ico` extension). Name the push button Sales_Rep_Lov_Button, and place it next to Sales_Rep_ID.
Display the Layout Editor and ensure the block is set to CONTROL.
Select the Push Button tool.
Create a push button and place it close to Sales_Rep_ID.
Resize the push button.
Set the Name to Sales_Rep_LOV_Button.
Set the Keyboard Navigable to No.
Set the Mouse Navigate to No.
Set the Iconic to Yes.
Set the Icon File Name to List.

10 Save, compile, and run the forms to test the changes.
No formal solution.

Practice 10 Solutions

1. Modify the window in the CUSTG*XX* form. Change the name of the window to WIN_CUSTOMER, and change its title to Customer Information. Check that the size and position are suitable.
   In the Layout Editor, look at the lowest and right-most positions of objects on the canvas, and plan the height and width for the window.
   Change the height, width, and title of the window in the Property Palette.
   Change the Title property to Customer Information.
   The suggested size is Width 360, Height 360.
   The suggested X, Y positions are 10, 10.
2. Save, compile, and run the form to test the changes.
   No formal solution.
3. Modify the window in the ORDG*XX* form. Ensure that the window is called WIN_ORDER. Also change its title to Orders and Items.
   Set the Name property of the existing window to WIN_ORDER.
   Set the Title property to Orders and Items.
4. In the ORDG*XX* form, create a new window called WIN_INVENTORY suitable for displaying the CV_INVENTORY canvas. Use the rulers in the Layout Editor to help you plan the height and width of the window. Set the window title to Stock Levels and Hide on Exit property to Yes. Place the new window in a suitable position relative to WIN_ORDER.
   Create a new window called WIN_INVENTORY.
   Set the size, position, and title in the Property Palette.
   The suggested size is Width 460, Height 340.
   The suggested position is X 60, Y 30.
   Set Hide on Exit to Yes.
5. Associate the CV_INVENTORY canvas with the window WIN_INVENTORY. Run the form to ensure that the S_INVENTORY block displays in WIN_INVENTORY when you navigate to this block.
   Set the canvas Window property to WIN_INVENTORY.
   Run the form.
6. Save the form.
   No formal solution.

Practice 11 Solutions

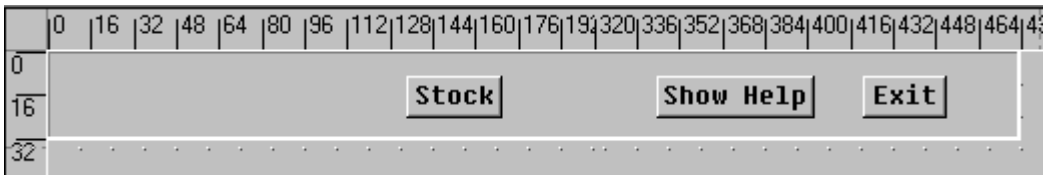
Toolbar Canvases

1. In the ORDG*XX* form, create a horizontal toolbar canvas called Toolbar in the WIN_ORDER window, make it the standard toolbar for that window. Suggested height is 30.
   Create a new canvas. Set Name to Toolbar and Height to 30. Set Canvas Type to Horizontal Toolbar and Window to WIN_ORDER. In the WIN_ORDER Property Palette of the window, set Horizontal Toolbar Canvas to Toolbar.
2. Save, compile, and run the form to test.
   Notice that the toolbar now uses part of the window space. Adjust the window size accordingly.
   No formal solution.

Create three push buttons in the Control block, as detailed below, and place them on the Toolbar canvas.
No formal solution.

Push Button Name	**Details**
Stock_Button	Label: Stock Mouse Navigate: No Keyboard Navigable: No Canvas: Toolbar
Show_Help_Button	Label: Show Help Mouse Navigate: No Keyboard Navigable: No Canvas: Toolbar
Exit_Button	Label: Exit Mouse Navigate: No Keyboard Navigable: No Canvas: Toolbar

Suggested positions for the push buttons are shown in the following illustration:

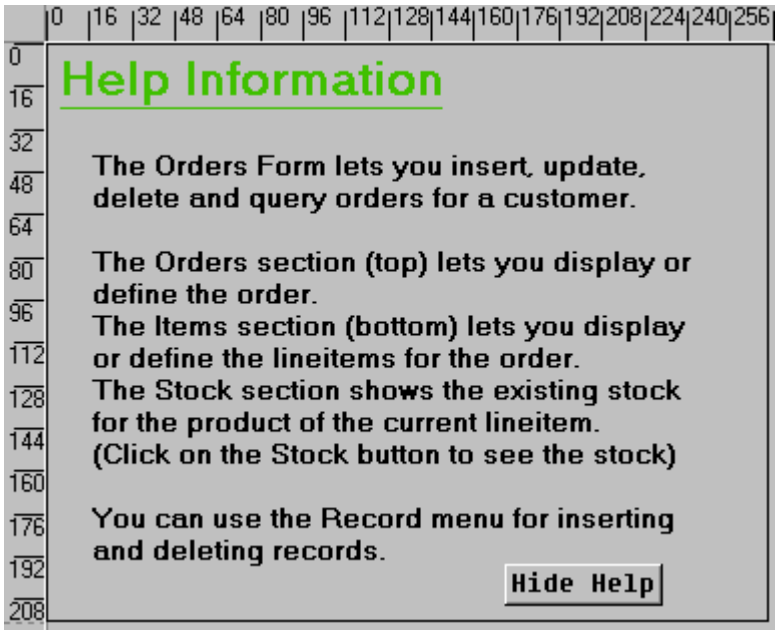


Stacked Canvases

1. Create a stacked canvas named CV_HELP to display help in the WIN_ORDER window of the ORDG*XX* form. Suggested *visible* size is Viewport Width 270, Viewport Height 215 (points). Place some application help text on this canvas.
   Create a new canvas.
   If the Property Palette is not already displayed, click the new canvas object in the Object Navigator and select Tools—>Property Palette.
   Set the Canvas Type to Stacked.
   Display the stacked canvas in the Layout Editor, and create some boilerplate text objects with help information about the form.
2. Position the view of the stacked canvas so that it appears in the center of WIN_ORDER. Make sure it will not obscure the first enterable item. Do this by planning the top-left position of the view in the Layout Editor, while showing CV_ORDER. Define the Viewport X and Viewport Y Positions in the Property Palette. Do not move the view in the Layout Editor.
   In the Layout Editor display the CV_ORDER canvas and select View—>Stack Views from the menu, and select CV_HELP from the list. You can see both canvases in this way. Change Viewport X and Viewport Y properties in the Property Palette for CV_Help Canvas.
3. Organize CV_HELP so that it is the last canvas in sequence.
   Do this in the Object Navigator. (This ensures the correct stacking order at run time.)
   Drag the Help_Canvas so that it is the last canvas displayed under the Canvases node.
4. Save, compile, and run the form to test. Note that the stacked canvas displays all the time, providing that it does not obscure the current item in the form.
   No formal solution.

5 Switch off the Visible property of CV_HELP, then create a push button in the CONTROL block to hide the Help information when it is no longer needed. We will add the code later. Display this push button on the CV_HELP canvas.
Set the Visible Property to No for the CV_HELP.
Create a push button in the CONTROL block with the following properties:

Push Button Name	**Details**
Hide_Help_Button	Label: Hide Help, Canvas: CV_HELP Mouse Navigate: No

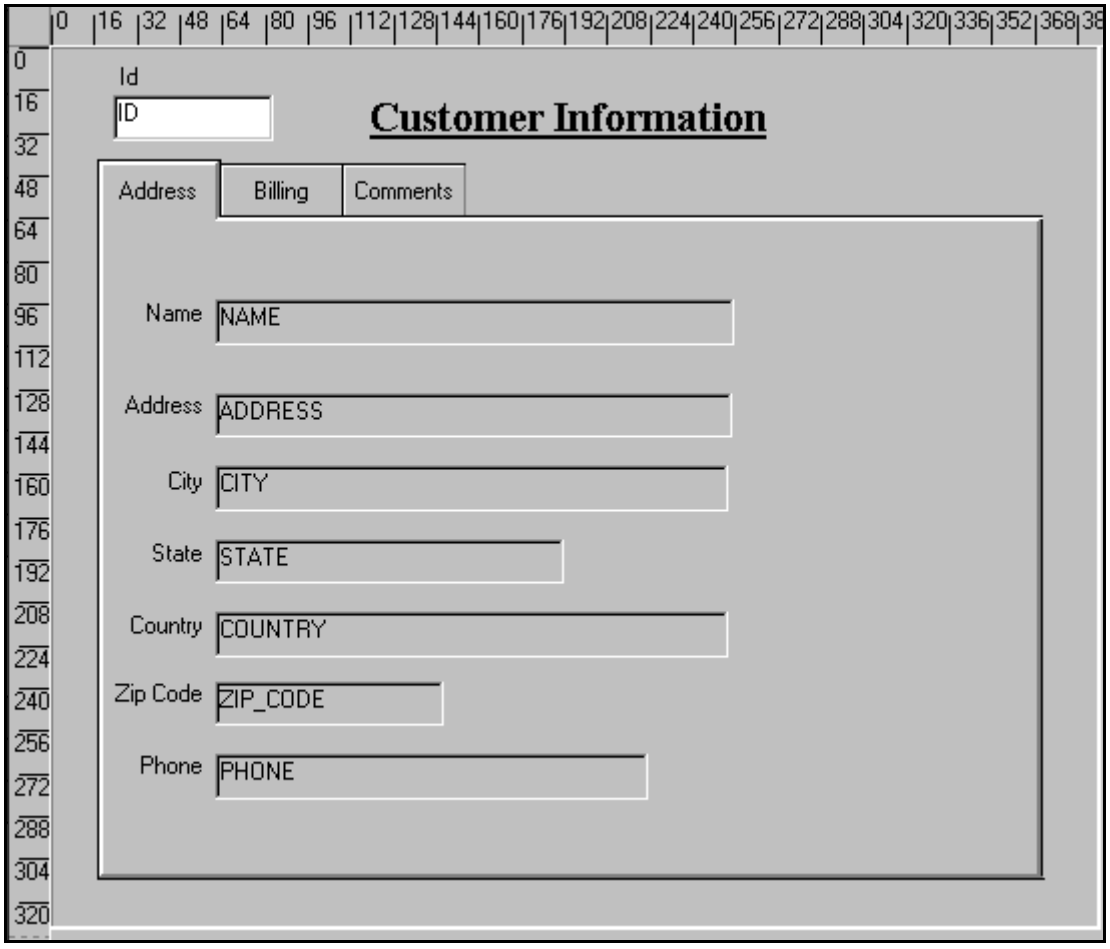


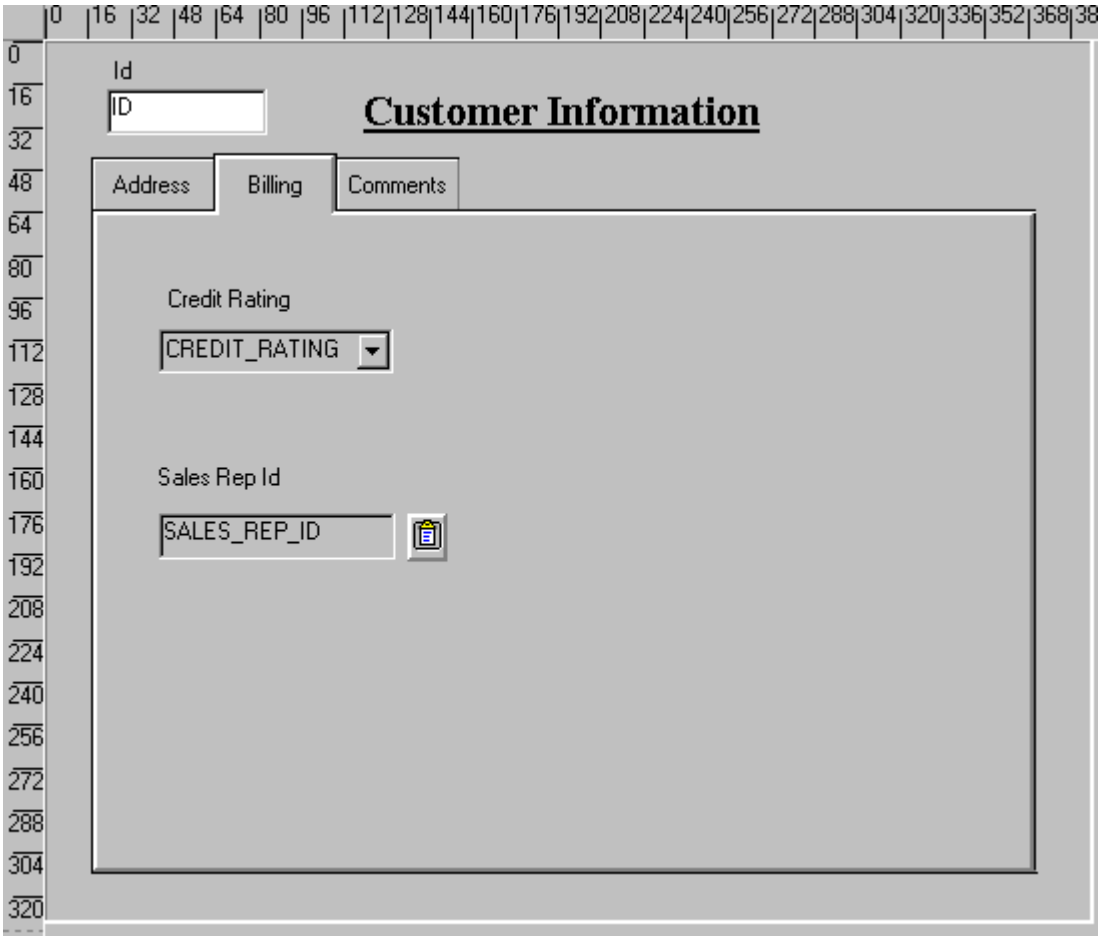
Tab Canvases

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

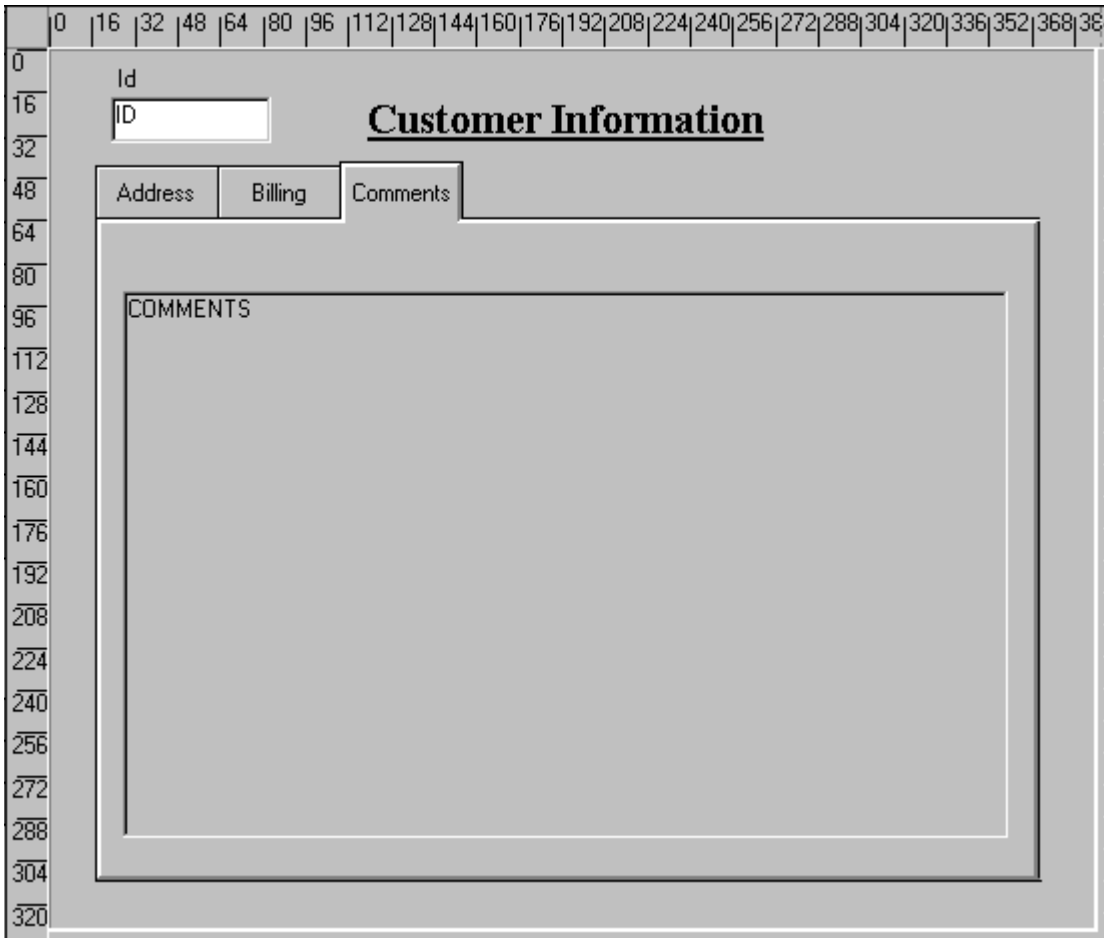
1. In the Layout Editor, delete the frame object that covers S_CUSTOMER block. Create a tab canvas. In the Layout Editor set the Background Color property to gray, Tab style property to Square, and Bevel property to None.
   In the Layout Editor, select the frame that covers S_CUSTOMER block and delete.
   Click the Tab canvas button in the toolbar and create a Tab canvas.
   In the Property Palette set the Background Color property to gray, Corner Style property to Square, and Bevel property to None.
2. Rename this tab canvas TAB_CUSTOMER. Create three tab pages and label them as Address, Billing, and Comments.
   In the Object Navigator select this Tab canvas and open the Property Palette. Set the Name property to TAB_CUSTOMER.
   In the Object Navigator expand this Tab canvas and create three Tab pages.
   Set Tab pages label properties as Address, Billing, and Comments.
3. Design the tab pages according to the following screenshots. Set the item properties to make them visible on the relevant tab pages.

   In the Object Navigator select NAME, ADDRESS, CITY, STATE, COUNTRY, ZIP_CODE, and PHONE items of the S_CUSTOMER block and open the Property Palette for this multiple selection.
   Set the Canvas property to TAB_CUSTOMER.
   Set the Tab Page property to the Tab page that is labeled Address.
   In the Layout Editor, arrange the items according to the following screenshot.


Id
ID
Customer Information
Address
Billing
Comments
Name
NAME
Address
ADDRESS
City
CITY
State
STATE
Country
COUNTRY
Zip Code
ZIP_CODE
Phone
PHONE



In the Object Navigator select CREDIT_RATING, SALES_REP_ID items of the S_CUSTOMER block, and SALES_REP_LOV button of the CONTROL block. Open the Property Palette for this multiple selection.
Set the Canvas property to TAB_CUSTOMER.
Set the Tab page property to the tab page that is labeled Billing.
In the Layout Editor, arrange the items according to the screenshot.



In the Object Navigator select the COMMENTS item of the S_CUSTOMER block, and open the Property Palette.
Set the Canvas property to TAB_CUSTOMER.
Set the Tab page property to the tab page that is labeled Comments.
In the Layout Editor resize the COMMENTS item according to the screenshot.

4 Reorder the items according to the tab page sequence. Ensure that the user does not move from one tab page to another when tabbing through items. Set Next Navigation Item and Previous Navigation Item properties according to the order of items in the tab pages.
Set the Previous Navigation Item and Next Navigation Items properties for items in the S_CUSTOMER block according to their order in the tab pages. For example, for the Phone item, set the Previous Navigation Item Property to ZIP_CODE and the Next Navigation Item Property to CREDIT_RATING.

5 Save, compile, and run the form.
No formal solution.

Practice 13 Solutions

1. In the CUSTG*XX* form, write a trigger to display the Sales_Rep_Lov when the Sales_Rep_Lov_Button is pressed. To create When-Button-Pressed trigger, use the Smart triggers feature. Find the relevant built-in in the Object Navigator under built-in packages, and use the "Paste Name and Arguments" feature.
   Select Sales_Rep_Lov_Button in the Object Navigator, press right mouse button, select Smart triggers in the pop-up menu, and select the When-Button-Pressed trigger from the list.
   When-Button-Pressed on CONTROL.Sales_Rep_Lov_Button:

```
IF SHOW_LOV('sales_rep_lov') THEN
   NULL;
END IF;
```

2. Create a When-Window-Closed trigger at the form level in order to exit form.
   When-Window-Closed a the form level:

```
EXIT_FORM;
```

3. Save, compile, and run the form.
   No formal solution.
4. In the ORDG*XX* form, write a trigger to display the Products_Lov when the Products_Lov_Button is selected.
   When-Button-Pressed on CONTROL.Products_Lov_Button:

```
IF SHOW_LOV('products_lov') THEN
   NULL;
END IF;
```

5. Write a trigger that exits the form when the Exit_Button is selected.
   When-Button-Pressed on CONTROL.Exit_Button:

```
EXIT_FORM;
```

6. Save, compile, and run the form.
   No formal solution.

7. Create a When-Button-Pressed trigger on CONTROL.Show_Help_Button that uses the SHOW_VIEW built-in to display the CV_HELP.
   When-Button-Pressed on CONTROL.Show_Help_Button:
   ```
   SHOW_VIEW('CV_HELP');
   ```
8. Create a When-Button-Pressed trigger on CONTROL.Hide_Help_Button that hides the CV_HELP. Use the HIDE_VIEW built-in to achieve this.
   When-Button-Pressed on CONTROL.Hide_Help_Button
   ```
   HIDE_VIEW('CV_HELP');
   ```
9. Save, compile, and run the ORDG*XX* form to test.
   The stacked canvas, CV_HELP, displays only if the current item will not be obscured. Ensure, at least, that the first entered item in the form is one that will not be obscured by CV_HELP.
   You might decide to advertise Help only while the cursor is in certain items, or move the stacked canvas to a position that does not overlay enterable items.
   The CV_HELP canvas, of course, could also be shown in its own window, if appropriate.
   No formal solution.
10. Create a When-Button-Pressed trigger on CONTROL.Stock_button that uses the GO_BLOCK built-in to display the S_INVENTORY block.
    When-Button-Pressed on CONTROL.Stock_Button:
    ```
    GO_BLOCK('S_INVENTORY');
    ```

Practice 14 Solutions

1 Open your CUSTG*xx*.FMB file. In this form, create a procedure that is called List_Of_Values. Import code from the pr14_1.txt file:

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

Modify the When-Button-Pressed trigger of CONTROL.Sales_Lov_Button in order to call this procedure.

When-Button-Pressed on CONTROL.Sales_Lov_Button:

```
LIST_OF_VALUES('SALES_REP_LOV', 'Sales Representative');
```

Compile and run your form in Debug mode. Set a breakpoint in one of your triggers, and investigate the call stack. Try stepping through the code to monitor its progress.
No formal solution. You can use CUSTWK14.fmb for compiled form.

Practice 15 Solutions

1. In the ORDG*XX* form write a trigger that fires when the Payment Type changes, allowing only those customers with a good or excellent Credit Rating to pay for orders on credit.You can import the `pr15_1.txt` file.

   Open the PL/SQL Editor and select File—>Import Text.

   When-Radio-Changed on S_ORD.Payment_Type:

```
DECLARE
   v_credit s_customer.credit_rating%type;
BEGIN
   IF :S_ORD.payment_type = 'CREDIT' THEN
      SELECT  credit_rating
      INTO v_credit
      FROM S_CUSTOMER
      WHERE :S_ORD.customer_id = id;
      IF v_credit NOT IN('GOOD','EXCELLENT') THEN
         :S_ORD.payment_type := 'CASH';
         MESSAGE('This customer must pay cash');
      END IF;
   END IF;
END;
```

2. In the CONTROL block, create a new button called Image_Button and position it on the Toolbar. Set Label property to Image Off.

   Display the Layout Editor.
   Select the Button tool.
   Create a button and place it on the Toolbar.
   Set the Name to Image_Button.
   Set the Canvas property to Toolbar.
   Set the Keyboard Navigable property to No.
   Set the Mouse Navigate property to No.
   Set the Label property to Image Off.

3 Import the file `pr15_3.txt` into a trigger that fires when the Image_Button is clicked.The file contains code that determines the current value of the visible property of the Product Image item. If the current value is True, the visible property toggles to False for both the Product Image item and the Image Description item. Finally the label changes on the Image_Button to reflect its next toggle state. However, if the visible property is currently False, the visible property toggles to True for both the Product Image item and the Image Description item.
Open the PL/SQL Editor and select File—>Import Text.

When-Button-Pressed on Control.Image_Button:

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

4 Save, compile, and run the form.
No formal solution.

Practice 16 Solutions

1. Create an alert in ORDG*XX* called Payment_Type_Alert with a single OK button. The message should read “This customer must pay cash!”
   Create an alert.
   Set the Name to PAYMENT_TYPE_ALERT.
   Set the Title to Payment Type.
   Set the Alert Style to Caution.
   Set the Button1 Label to OK.
   Set the Message to “This customer must pay cash!”
   Remove the labels for the other buttons.

2. Alter the When-Radio-Changed trigger on Payment_Type to show the Payment_Type_Alert instead of the message when a customer must pay cash.
   When-Radio-Changed on S_ORD.Payment_Type:

```
DECLARE
   n NUMBER;
   v_credit S_CUSTOMER.credit_rating%type;
BEGIN
   IF :S_ORDER.payment_type = 'CREDIT' THEN
      SELECT credit_rating
      INTO v_credit
      FROM S_CUSTOMER
      WHERE :S_ORD.customer_id = id;
      IF v_credit not in('GOOD','EXCELLENT') THEN
         :S_ORD.payment_type := 'CASH';
         n := SHOW_ALERT('payment_type_alert');
      END IF;
   END IF;
END;
```

3. Create a generic alert called Question_Alert that allows Yes and No replies.
   Create an alert.
   Set the Name to QUESTION_ALERT.
   Set the Title to Question.
   Set the Alert Style to Stop.
   Set the Button1 Label to Yes.
   Set the Button2 Label to No.

4 Alter the When-Button-Pressed trigger on CONTROL.Exit_Button that uses Question_Alert to ask the operator to confirm that the form should terminate.

When-Button-Pressed on CONTROL.Exit_Button:

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

5 Save, compile, and run the form to test.

No formal solution.

Practice 17 Solutions

1. In the ORDG*XX* form, write a trigger that populates the Customer_Name and the Sales_Rep_Name for every row fetched by a query on the S_ORD block.
   Post-Query on S_ORD block:

```
SELECT C.name, E.last_name
INTO :S_ORD.customer_name, :S_ORD.sales_rep_name
FROM S_CUSTOMER C, S_EMP E
WHERE E.id = C.sales_rep_id AND :S_ORD.customer_id = C.id;
```

2. Write a trigger that populates the Description for every row fetched by a query on the S_ITEM block.
   Post-Query on S_ITEM block:

```
SELECT name
INTO :S_ITEM.description
FROM S_PRODUCT
WHERE :S_ITEM.product_id = id;
```

3. Ensure that the Exit_Button has no effect in Enter Query mode.
   Set Fire in Enter Query Mode property to No for the When-Button-Pressed trigger.

4. Adjust the default query interface. Open the CUSTOMERS form module. Add a check box called CONTROL.Case_Sensitive to the form so that the user can specify whether or not a query for a customer name should be case sensitive. You can import the `pr17_4.txt` file into the When-Checkbox-Changed trigger. Set the initial value property to Y. In the CONTROL block, add a check box (called CONTROL.Case_Sensitive as shown below) to it, and create the following trigger. Set the Mouse Navigate Property to No.

   When-Checkbox-Changed trigger on the CONTROL.Case_Sensitive item (checkbox):

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

5 Add a check box called CONTROL.Exact_Match to the form so that the user can specify whether or not a query condition for a customer name should exactly match the table value. (If a nonexact match is allowed, the search value can be part of the table value.) You can import the `pr17_5.txt` file into the Pre-Query Trigger. Set the initial value property to Y. Add another check box (called CONTROL.Exact_Match as shown below) to the CONTROL block and create the following trigger. Set the Mouse Navigate Property to No.

Pre-Query trigger on the S_CUSTOMER block:

```
IF NVL( :CONTROL.exact_match, 'Y' ) = 'N' THEN
   :S_CUSTOMER.name := '%' || :S_CUSTOMER.name || '%';
END IF;
```

Practice 18 Solutions

1. In the CUSTG*XX* form, cause the Sales_Rep_Lov to display whenever the user enters a Sales_Rep_Id that does not exist in the database.
   Set the Validate from List property to Yes for the Sales_Rep_Id item in the S_CUSTOMER block.
2. Save, compile, and run the form to test.
   No formal solution.
3. In the ORDG*XX* form, write a validation trigger to check that the Date_Shipped is not before the Date_Ordered.
   When-Validate-Record on S_ORD block:

```
IF  :S_ORD.date_shipped < :S_ORD.date_ordered THEN
   MESSAGE('Ship date is before order date!');
   RAISE form_trigger_failure;
END IF;
```

4. In the ORDG*XX* form, create a trigger to write the correct values to the Customer_Name, Sales_Rep_Name, and Sales_Rep_Id items whenever validation occurs on Customer_Id.

   Fail the trigger if the customer is not found.

   When-Validate-Item on S_ORD.Customer_Id:

```
SELECT C.name,
       C.sales_rep_id,
       E.last_name
INTO  :S_ORD.customer_name,
      :S_ORD.sales_rep_id,
      :S_ORD.sales_rep_name
FROM S_CUSTOMER C, S_EMP E
WHERE E.id = C.sales_rep_id AND :S_ORD.customer_id = C.id;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      MESSAGE('Invalid Customer Id');
      RAISE form_trigger_failure;
```

5 Create another validation trigger on S_ITEM.Product_Id to derive the name of the product, and write it to the Description item.
Fail the trigger and display a message if the product is not found.

When-Validate-Item on S_ITEM.Product_Id:

```
SELECT name,
       suggested_whlsl_price
INTO  :S_ITEM.description,
      :S_ITEM.price
FROM S_PRODUCT
WHERE :S_ITEM.product_id = id;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      MESSAGE('Invalid Product Id!');
      RAISE form_trigger_failure;
```

Practice 19 Solutions

1. Write a When-New-Form-Instance trigger on the ORDG*XX* form to execute a query at form startup.
   When-New-Form-Instance at form level:

```
EXECUTE_QUERY;
```

2. Write a trigger that fires as the cursor arrives in each record of the S_ITEM block, and populate the Product_Image item with a picture of the product, if one exists.
   When-New-Record-Instance on S_ITEM block:

```
DECLARE
   filename VARCHAR2(20);
BEGIN
   filename := GET_PRODUCT_IMAGE(:S_ITEM.product_id);
   IF filename = 'No file' THEN
      null;
   ELSE
      READ_IMAGE_FILE(filename,'tiff','S_ITEM.product_image');
   END IF;
END;
```

   Get_Product_Image function is already created for you. This function returns the image file name for the given product number.

```
FUNCTION get_product_image (product_number IN NUMBER) RETURN
VARCHAR2 IS
   v_filename VARCHAR2(20);
BEGIN
   SELECT s_image.filename INTO v_filename
      FROM   s_image, s_product
      WHERE  s_image.id = s_product.image_id
      AND s_product.id = product_number;
   IF v_filename is null THEN
      v_filename := 'No file';
   END IF;
   RETURN v_filename;
EXCEPTION
   WHEN no_data_found THEN return('No file');
END;
```

3. Define the same trigger type and code on the S_ORD block.
   When-New-Record-Instance on S_ORD block. Use the same code above.
4. Is there another trigger where you might also want to place this code?
   When-Validate-Item on S_ITEM.Product_Id is a candidate for the code.
   You could use a procedure to avoid duplicating the trigger code.
5. Save, compile, and run the form to test.
   No formal solution.

Practice 20 Solutions

1. In the ORDG*XX* form write a transactional trigger on the S_ORD block that populates S_ORD.Id with the next value from the S_ORD_ID sequence.
   Pre-Insert on S_ORD block:

```
SELECT S_ORD_ID.nextval
INTO :S_ORD.id
FROM SYS.DUAL;
EXCEPTION
   WHEN OTHERS THEN
      MESSAGE('Failed to assign Order Id');
      RAISE form_trigger_failure;
```

2. In the S_ORD block, set the Enabled property for the ID item to No.
   In the Property Palette, set Enabled property to No for S_ORD.Id.
3. Save, compile, and run the form to test.
   No formal solution.
4. Create a similar trigger on the S_ITEM block that assigns the Item_Id when a new record is saved.
   Pre-Insert on S_ITEM block:

```
SELECT NVL(MAX(item_id),0) + 1
INTO :S_ITEM.item_id
FROM S_ITEM
WHERE :S_ITEM.ord_id = ord_id;
```

   Set the Required and Enabled properties to No for S_ITEM.Item_Id.
5. Save, compile, and run the form to test.
   No formal solution.

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

Post-Insert at form level:

```
DEFAULT_VALUE('0', 'GLOBAL.insert');
:GLOBAL.insert := TO_CHAR( TO_NUMBER( :GLOBAL.insert ) + 1 );
```

Post-Update at form level:

```
DEFAULT_VALUE('0', 'GLOBAL.update');
:GLOBAL.update := TO_CHAR( TO_NUMBER( :GLOBAL.update ) + 1 );
```

Post-Delete at form level:

```
DEFAULT_VALUE('0', 'GLOBAL.delete');
:GLOBAL.delete := TO_CHAR( TO_NUMBER( :GLOBAL.delete ) + 1 );
```

7 Create a procedure called HANDLE_MESSAGE. Import the `pr20_10.txt` file. This procedure receives two arguments. The first one is a message number, and the second is a Boolean error indicator. This procedure uses the three global variables to display a customized commit message and then erases the global variables.

```
PROCEDURE handle_message( message_number IN NUMBER, IS_ERROR IN
BOOLEAN ) IS
BEGIN
   IF message_number IN ( 40400, 40406, 40407 ) THEN
      DEFAULT_VALUE( '0', 'GLOBAL.insert' );
      DEFAULT_VALUE( '0', 'GLOBAL.update' );
      DEFAULT_VALUE( '0', 'GLOBAL.delete' );
      MESSAGE('Save Ok: ' ||
      :GLOBAL.insert || ' records inserted, ' ||
      :GLOBAL.update || ' records updated, ' ||
      :GLOBAL.delete || ' records deleted !!!' );
   ELSIF is_error = TRUE THEN
      MESSAGE('ERROR: ' || ERROR_TEXT );
   ELSE
      MESSAGE( MESSAGE_TEXT );
   END IF;
END ;
```

Call the procedure when an error occurs. Pass the error code and TRUE. Call the procedure when a message occurs. Pass the message code and FALSE.

On-Error at form level:

```
handle_message( error_code, TRUE );
```

On-Message at form level:

```
handle_message( message_code, FALSE );
```

8 Open the CUSTG*XX* form module. Write an On-Logon trigger to control the number of connection tries. Use the LOGON_SCREEN built-in to simulate the default login screen and LOGON to connect to the database. You can import the `pr20_11.txt` file.

On-Logon at form level:

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

Practice 21 Solutions

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

When-New-Record-Instance on S_ORD and S_ITEM blocks:

```
DECLARE
   filename VARCHAR2(20);
BEGIN
   IF GET_ITEM_PROPERTY('S_ITEM.product_image',VISIBLE)='TRUE'
   THEN
      filename := GET_PRODUCT_IMAGE(:S_ITEM.product_id);
      IF filename = 'No file' THEN
         null;
      ELSE
         READ_IMAGE_FILE(filename,'tiff',
                         'S_ITEM.product_image');
      END IF;
   END IF;
END;
```

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

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

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

Practice 22 Solutions

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

Property	**Setting**
Font Name	Arial
Format Mask	99,999
Font Size	8
Justification	Right
Delete Allowed	No
Background Color	DarkRed

   Select the Property Classes node and click the Create icon.
   In the Property Palette for the property class, set the Name to ClassA.
   Add the properties listed by clicking the Add Property icon and selecting from the list displayed.
   Set the properties to the values listed above.
5. Apply ClassA to CV_INVENTORY, the Restock_Date item and the Max_In_Stock item.
   In the Property Palette for each of the items, select the Subclass Information property, and click the More button. In the Subclass Information dialog box, select the Property Class radio button and set the Property Class Name list item to ClassA.

6 Save the form module as STOCK*XX*.fmb, compile, and run the form and note the error.
You should receive the following error:
Invalid format mask for given datatype for S_INVENTORY.Restock_Date.

7 Make the Restock_Date format mask a variant property.
Change the Format Mask for S_INVENTORY.Restock_Date to MM/DD/YYYY.

8 Save, compile, and run the form. Corret the error. Save, compile, and run the form again.
You should receive the following error:
Cannot resolve reference item reference s_ITEM.Product_Id
Delete s_ITEM.Product_Id From the Copy Value From Item property.

9 Create an object library and name it summit.
Create two tabs in the Object Library called Personal and Corporate.
Add the CONTROL block, the Toolbar, and the Question_Alert to the personal tab of the object library.
Save the object library as summit.olb.
Select the Object Libraries node in the Object Navigator, and click the Create button. Rename this object library summit.
Expand the Library tabs node in the Object Navigator.
Set their label properties as Personal and Corporate.
Drag the CONTROL block, the Toolbar, and the Question_Alert to the Personal tab of the object library.
Save the summit Object library.

10 Create a new form, and create a data block based on the S_DEPT table.
Drag the Toolbar canvas, CONTROL block, and Question_Alert from the object library into the new form. For proper behavior, the S_DEPT block must be before the CONTROL block in the Object Navigator.
Subclass the objects.
Some items are not applicable to this form. Set the Canvas property for the following items to NULL: Image_Button, Stock_Button, Show_Help_Button, Product_Lov_Button, Hide_Help_Button.
Use Toolbar as the form horizontal toolbar canvas for this form.
Set the Window property to WINDOW1 for the Toolbar canvas.
Set the Horizontal Toolbar Canvas property to TOOLBAR for the window.
Save this form as DEPTG*XX*, compile, and run the form to test it.
Follow the practice steps, see the DEPTWK22 file for a solution.

11 Try to delete items on the Null canvas. What happens and why?
You cannot delete the objects because the Toolbar and contents are subclassed from another object.

12 Create two sample buttons, one for wide buttons and one for medium buttons, by means of width.
Create a sample date field. Set the width and the format mask to your preferred standard.
Drag these items into your object library.
Mark these items as SmartClasses.
Create a new form and a new data block in the form. Apply these SmartClasses in your form. Place the Toolbar canvas in the new form.
No formal solution.

Practice 23 Solutions

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

8 Alter the Orders_Button trigger in CUSTG*XX* so that it uses GO_FORM to pass control to ORDG*XX* if the form is already running. Use the FIND_FORM built-in for this purpose.

When-Button-Pressed on Orders_Button:

```
:GLOBAL.customer_id := :S_CUSTOMER.id;
IF ID_NULL(FIND_FORM('ORDERS'))THEN
   OPEN_FORM('ORDG XX');
ELSE
   GO_FORM('ORDERS');
END IF;
```

Remember that you need to use the module name in the GO_FORM built-in, and the filename in the OPEN_FORM built-in.

9 Write a When-Create-Record trigger on the S_ORD block that uses the value of GLOBAL.Customer_Id as the default value for S_ORD.Customer_Id.

When-Create-Record on S_ORD block:

```
:S_ORD.customer_id := :GLOBAL.customer_id;
```

10 Add code to the CUSTG*XX* form so that GLOBAL.Customer_Id is updated when the current Customer_Id changes.

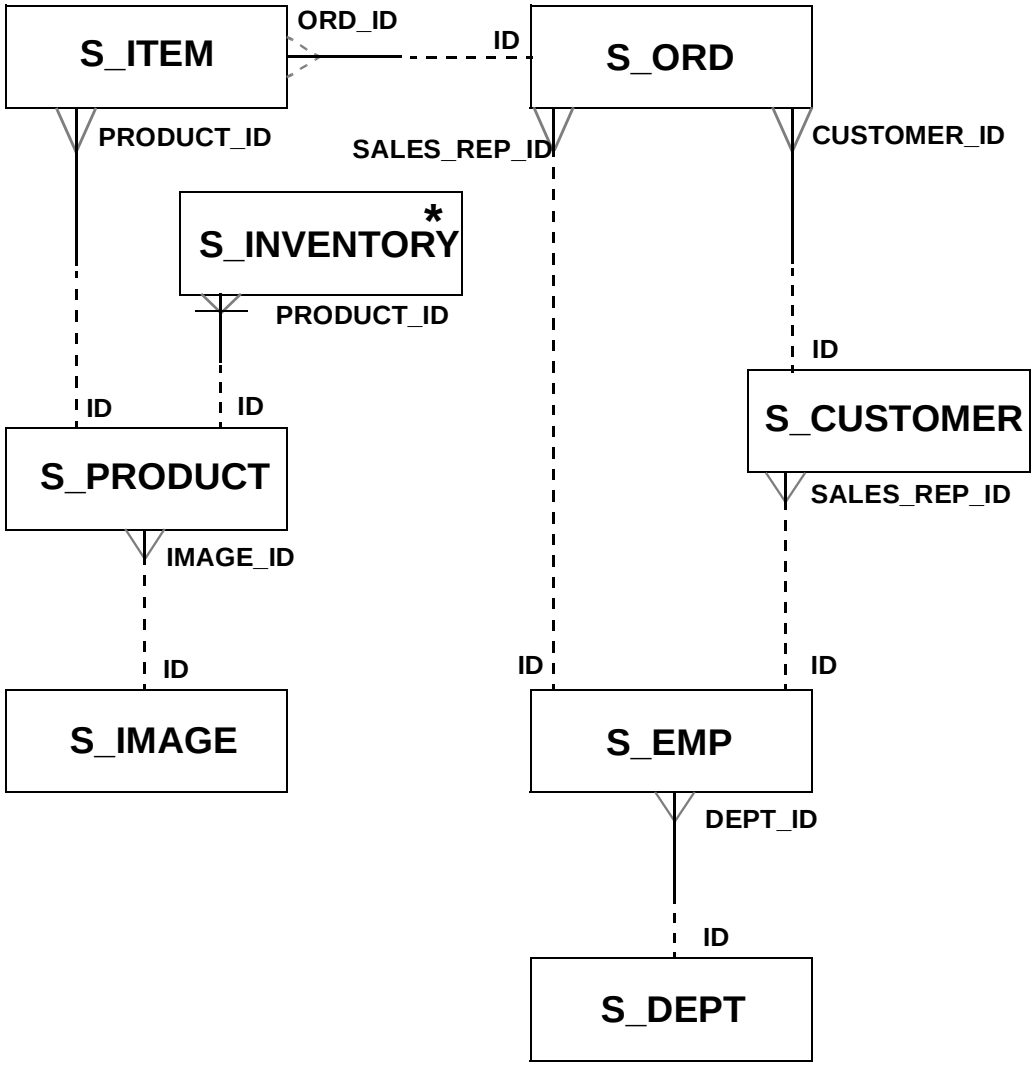
When-Validate-Item on S_CUSTOMER.Id:

```
:GLOBAL.customer_id := :S_CUSTOMER.id;
```

B

Table Descriptions and Data

Summit Sporting Goods Database Diagram



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

S_CUSTOMER Description

Column Name	Null?	Datatype
ID	NOT NULL	NUMBER(7)
NAME	NOT NULL	VARCHAR2(50)
PHONE		VARCHAR2(25)
ADDRESS		VARCHAR2(400)
CITY		VARCHAR2(30)
STATE		VARCHAR2(20)
COUNTRY		VARCHAR2(30)
ZIP_CODE		VARCHAR2(75)
CREDIT_RATING		VARCHAR2(9)
SALES_REP_ID		NUMBER(7)
REGION_ID		NUMBER(7)
COMMENTS		VARCHAR2(255)

S_CUSTOMER Data

```
SQL> SELECT * FROM s_customer;

   Id    Name                 Phone          Address
------   ------------------   ------------   -----------
City                State                   Country
------------------   ----------------       -----------
Zip_code       Credit_ra      Sales_rep_id   Region_id
----------     ---------      ------------   ---------
Comments
-------------------------------------------------
   201   Unisports            55-2066101     72 Via Bahia
Sao Paolo                                   Brazil
              Excellent      12             2
Customer Usually Orders Large Amounts And Has A High Order
Total. This Is Okay As Long As The Credit Rating Remains
Excellent.

   202   Oj Atheletics        81-20101       6741 Takashi Blvd.
Osaka                                       Japan
              Poor           14             4
Customer Should Always Pay By Cash Until His Credit Rating
Improves.

   203   Delhi Sports         91-10351       11368 Chanakya
New Delhi                                   India
              Good           14             4
Customer Specializes In Baseball Equipment And Is The Largest
Retailer In India.
```

S_CUSTOMER Data (continued)

```
   Id    Name                 Phone          Address
------   ------------------   ------------   -----------
City                State                   Country
------------------   ----------------       -----------
Zip_code       Credit_ra      Sales_rep_id   Region_id
----------     ---------      ------------   ---------
Comments
--------------------------------------------------
   204   Womansport           1-206-104-0103 281 King Street
Seattle              Washington              USA
98101          EXCELLENT      11             1

   205      Kam’s Sporting Goods 852-3692888 15 Henessey Road
Hong Kong      EXCELLENT      15             4

   206   Sportique            33-2257201     172 Rue de Rivoli
Cannes                                      France
              EXCELLENT      15             5
Customer specializes in Soccer. Likes to order accessories in
bright colors.

   207   Sweet Rock Sports    234-603620     6 Saint Antoine
Lagos                                       Nigeria
```

S_CUSTOMER Data (continued)

```
   Id    Name                Phone          Address
------   ------------------  ------------   -----------
City                State                  Country
-----------------   ----------------       -----------
Zip_code       Credit_ra     Sales_rep_id   Region_id
----------     ---------     ------------   ---------
Comments
-------------------------------------------------
   208   Muench Sports       49-527454      435 Gruenestrasse
Stuttgart                                   Germany
              GOOD          15             5
Customer usually pays small orders by cash and large orders on
credit.

   209   Beisbol Si!         809-352689     789 Playa Del Mar
San Pedro de Macon's                        Dominican Republic
              EXCELLENT     11             1

   210   Futbol Sonora       52-404562      3 Via Saguaro
Nogales
              EXCELLENT     12             2
Customer is difficult to reach by phone. Try mail.

   211   Kuhn's Sports       42-111292      7 Modrany
Prague                                      Czechoslovakia
              EXCELLENT     15             5
```

S_CUSTOMER Data (continued)

```
   Id    Name                 Phone          Address
------   ------------------   ------------   -----------
City                State                   Country
------------------   ----------------       -----------
Zip_code       Credit_ra      Sales_rep_id   Region_id
----------     ---------      ------------   ---------
Comments
-------------------------------------------------

   212   Hamada Sport         20-1209211     57A Corniche
Alexandria                              Egypt
              EXCELLENT      13             3
Customer orders sea and water equipment.

   213   Big John’s Sports    1-415-555-6281 4783 18th Street
Emporium
San Francisco        CA                     USA
94117          EXCELLENT      11             1
Customer has a dependable credit record.

   214   Ojibway Retail       1-716-555-7171 415 Main Street
Buffalo              NY                     USA
14202          POOR           11             1

   215   Sporta Russia        7-3892456      6000 Yekatamina
St. Petersburg                              Russia
              POOR           15             5
This customer is very friendly, but has difficulty paying bills.
Insist upon cash.
```

Note: This display has been formatted.

S_DEPT Description and Data

```
Column Name                    Null?            Datatype
-----------------------        -------------    --------------
ID                             NOT NULL         NUMBER(7)
NAME                           NOT NULL         VARCHAR2(25)
REGION_ID                                       NUMBER(7)
```

```
SQL> SELECT * FROM s_dept;

   ID   NAME                                REGION_ID
------   ----------------                 -----------------
   10   Finance                             1
   31   Sales                               1
   32   Sales                               2
   33   Sales                               3
   34   Sales                               4
   35   Sales                               5
   41   Operations                          1
   42   Operations                          2
   43   Operations                          3
   44   Operations                          4
   45   Operations                          5
   50   Administration                      1

12 rows selected.
```

S_EMP Description

Column Name	Null?	Datatype
ID	NOT NULL	NUMBER(7)
LAST_NAME	NOT NULL	VARCHAR2(25)
FIRST_NAME		VARCHAR2(25)
USERID		VARCHAR2(8)
START_DATE		DATE
COMMENT		VARCHAR2(255)
MANAGER_ID		NUMBER(7)
TITLE		VARCHAR2(25)
DEPT_ID		NUMBER(7)
SALARY		NUMBER(11,2)
COMMISSION_PCT		NUMBER(4,2)

S_EMP Data

```
SQL> SELECT * FROM s_emp;

   ID LAST_NAME         FIRST_NAME           USERID   START_DAT
----  ----------------- ----------------     -------- ---------
COMMENTS
-------------------------------------------------------------
MANAGER_ID  TITLE                           DEPT_ID  SALARY
----------  ------------------------------  -------  ------
COMMISSION_PCT
---------------
   1  Velasquez         Carmen               cvelasqu 03-MAR-90
         President                          50 2500
   2  Ngao              LaDoris              lngao    08-MAR-90
      1  VP, Operations                      41    1450

   3  Nagayama          Midori               mnagayam 17-JUN-91
      1  VP, Sales                           31    1400

   4  Quick-To-See      Mark                 mquickto 07-APR-90
      1  VP, Finance                         10    1450

   5  Ropeburn          Audry                aropebur 04-MAR-90
      1  VP, Administration                  50    1550

   6  Urguhart          Molly                murguhar 18-JAN-91
      2  Warehouse Manager                   41    1200

   7  Menchu            Roberta              rmenchu  14-MAY-90
      2  Warehouse Manager                   42    1250
   8  Biri              Ben                  bbiri    07-APR-90
      2  Warehouse Manager                   43    1100

   9  Catchpole         Antoinette           acatchpo 09-FEB-92
      2  Warehouse Manager                   44    1300
```

S_EMP Data (continued)

```
   ID LAST_NAME         FIRST_NAME           USERID   START_DAT
----  ----------------- ----------------     -------- ---------
COMMENTS
-------------------------------------------------------------
MANAGER_ID  TITLE                           DEPT_ID  SALARY
----------  ------------------------------  -------  ------
COMMISSION_PCT
---------------
   10 Havel             Marta                mhavel   27-FEB-91
         2  Warehouse Manager                45    1307

   11 Magee             Colin                cmagee   14-MAY-90
         3  Sales Representative             31    1400
             10

   12 Giljum            Henry                hgiljum  18-JAN-92
         3  Sales Representative             32    1490
            12.5

   13 Sedeghi           Yasmin               ysedeghi 18-FEB-91
         3  Sales Representative             33    1515
             10

   14 Nguyen            Mai                  mnguyen  22-JAN-92
         3  Sales Representative             34    1525
             15

   15 Dumas             Andre                adumas   09-OCT-91
         3  Sales Representative             35    1450
            17.5

   16 Maduro            Elena                emaduro  07-FEB-92
         6  Stock Clerk                      41    1400
```

S_EMP Data (continued)

```
   ID LAST_NAME         FIRST_NAME           USERID   START_DAT
----  ----------------- ----------------     -------- ---------
COMMENTS
-------------------------------------------------------------
MANAGER_ID  TITLE                           DEPT_ID  SALARY
----------  ------------------------------  -------  ------
COMMISSION_PCT
---------------
   17 Smith             George               gsmith   08-MAR-90
         6  Stock Clerk                      41    940

   18 Nozaki            Akira                anozaki  09-FEB-91
         7  Stock Clerk                      42    1200

   19 Patel             Vikram               vpatel   06-AUG-91
         7  Stock Clerk                      42    795

   20 Newman            Chad                 cnewman  21-JUL-91
         8  Stock Clerk                      43    750

   21 Markarian         Alexander            amarkari 26-MAY-91
         8  Stock Clerk                      43    850

   22 Chang             Eddie                echang   30-NOV-90
         9  Stock Clerk                      44    800

   23 Patel             Radha                rpatel   17-OCT-90
         9  Stock Clerk                      34    795

   24 Dancs             Bela                 bdancs   17-MAR-91
         10 Stock Clerk                      45    860

   25 Schwartz          Sylvie               sschwart 09-MAY-91
         10 Stock Clerk                      45    1100
```

Note: This display has been formatted.

S_ITEM Description

Column Name	Null?	Datatype
ORD_ID	NOT NULL	NUMBER(7)
ITEM_ID	NOT NULL	NUMBER(7)
PRODUCT_ID	NOT NULL	NUMBER(7)
PRICE		NUMBER(11,2)
QUANTITY		NUMBER(9)
QUANTITY_SHIPPED		NUMBER(9)

S_ITEM Data

```
SQL> SELECT * FROM s_item;

ORD_ID   ITEM_ID  PRODUCT_ID  PRICE QUANTITY QUANTITY_SHIPPED
------   -------- ---------- ------ -------- -----------------
   100   1        10011       135   500      500
   100   2        10013       380   400      400
   100   3        10021       14    500      500
   100   5        30326       582   600      600
   100   7        41010       8     250      250
   100   6        30433       20    450      450
   100   4        10023       36    400      400
   101   1        30421       16    15       15
   101   3        41010       8     20       20
   101   5        50169       4.29  40       40
   101   6        50417       80    27       27
   101   7        50530       45    50       50
   101   4        41100       45    35       35
   101   2        40422       50    30       30
   102   1        20108       28    100      100
   102   2        202011      23    45       45
   103   1        30433       20    15       15
   103   2        32779       7     11       11
   104   1        20510       9     7        7
   104   4        30421       16    35       35
   104   2        20512       8     12       12
   104   3        30321       1669  19       19
   105   1        50273       22.8  16       16
   105   3        50532       47    28       28
   105   2        50419       80    13       13
```

S_ITEM Data (continued)

ORD_ID	ITEM_ID	PRODUCT_ID	PRICE	QUANTITY	QUANTITY_SHIPPED
106	1	20108	28	46	46
106	4	50273	22.89	75	75
106	5	50418	75	98	98
106	6	50419	80	27	27
106	2	20201	123	21	21
106	3	50169	4.29	125	125
107	1	20106	11	50	50
107	3	20201	115	130	130
107	5	30421	16	55	55
107	4	30321	1669	75	75
107	2	20108	28	22	22
108	1	20510	9	9	9
108	6	41080	35	50	50
108	7	41100	45	42	42
108	5	32861	60	57	57
108	2	20512	8	18	18
108	4	32779	7	60	60
108	3	30321	1669	85	85
109	1	10011	140	150	150
109	5	30426	18.25	500	500
109	7	50418	75	43	43
109	6	32861	60	50	50
109	4	30326	582	1500	1500
109	2	10012	175	600	600
109	3	10022	21.95	300	300
110	1	50273	22.89	17	17
110	2	50536	50	23	23
111	1	40421	65	27	27
111	2	41080	35	29	29
97	1	20106	9	1000	1000
97	2	303211	500	50	50
98	1	404218	5	7	7
99	1	20510	9	18	18
99	2	20512	8	25	25
99	3	50417	80	53	53
99	4	50530	45	69	69
112	1	20106	11	50	50

62 rows selected.

Note: This display has been formatted.

S_ORD Description and Data

```
Column Name                    Null?          Datatype
-----------------------------  ------         ---------------------
ID                             NOT NULL       NUMBER(7)
CUSTOMER_ID                    NOT NULL       NUMBER(7)
DATE_ORDERED                                  DATE
DATE_SHIPPED                                  DATE
SALES_REP_ID                                  NUMBER(7)
TOTAL                                         NUMBER(11,2)
PAYMENT_TYPE                                  VARCHAR2(6)
ORDER_FILLED                                  VARCHAR2(1)


SQL> SELECT * FROM s_ord;

ID  CUSTOMER_ID   DATE_ORDE    DATE_SHIPSALES_REP_IDTOTAL PAYMEN      ORDER_F
---  -----------  ---------    ---------------------------------      ------
100         204   31-AUG92     10-SEP-9211     601100     CREDIT      Y
101         205   31-AUG-92    15-SEP-9214     8056.6     CREDIT      Y
102         206   01-SEP-92    08-SEP-9215     8335       CREDIT      Y
103         208   02-SEP-92    22-SEP-9215     377        CASH        Y
104         208   03-SEP-92    23-SEP-9215     32430      CREDIT      Y
105         209   04-SEP-92    18-SEP-9211     2722.24    CREDIT      Y
106         210   07-SEP-92    15-SEP-9212     15634      CREDIT      Y
107         211   07-SEP-92    21-SEP-9215     142171     CREDIT      Y
108         212   07-SEP-92    10-SEP-9213     149570     CREDIT      Y
109         213   08-SEP-92    28-SEP-9211     1020935    CREDIT      Y
110         214   09-SEP-92    21-SEP-9211     1539.13    CASH        Y
111         204   09-SEP-92    21-SEP-9211     2770       CASH        Y
97          201   28-AUG-92    17-SEP-9212     84000      CREDIT      Y
98          202   31-AUG-92    10-SEP-9214     595        CASH        Y
99          203   31-AUG-92    18-SEP-9214     7707       CREDIT      Y
112         210   31-AUG-92    10-SEP-9212     550        CREDIT      Y

16 rows selected.
```

S_PRODUCT Description

Column Name	Null?	Datatype
ID	NOT NULL	NUMBER(7)
NAME	NOT NULL	VARCHAR2(50)
SHORT_DESC		VARCHAR2(255)
LONGTEXT_ID		NUMBER(7)
IMAGE_ID		NUMBER(7)
SUGGESTED_WHLSL_PRICE‘		NUMBER(11,2)
WHLSL_UNITS		VARCHAR2(25)

S_PRODUCT Data

```
SQL> SELECT * FROM s_product;

   ID    NAME        SHORT_DESC        LONGTEXT_ID
 ----    ------------------------    -----------
IMAGE_ID SUGGESTED_WHLSL_PRICEWHLSL_UNITS
-------- --------------------------------
10011    Boot        Beginner's ski boot  518
 1001                150

10012    Ace Ski BootIntermediate ski boot519
 1002                200

10013    Pro Ski BootAdvanced ski boot    520
 1003                410

10021    Bunny Ski PoleBeginner's ski pole528
 1011                16.25

10022    Ace Ski PoleIntermediate ski pole529
 1012                21.95

10023    Pro Ski PoleAdvanced ski pole    530
 1013                40.95

20106    Junior Soccer BallJunior soccer ball613
                     11

 20108   World Cup Soccer BallWorld cup soccer ball615
                     28

 20201   World Cup NetWorld cup net       708
                     123
```

S_PRODUCT Data (continued)

```
ID    NAME        SHORT_DESC        LONGTEXT_ID
----  ----------------------------------------
IMAGE_IDSUGGESTED_WHLSL_PRICEWHLSL_UNITS
----------------------------------------
20510 Black Hawk  Knee pads, pair   1017
      Knee Pads   9

 20512Black Hawk  Elbow pads, pair  1019
      Elbow Pads  8

30321 Grand Prix Bicycle Road bicycle828
 1291             1669

30326 Himalaya BicycleMountain bicycle833
1296              582

30421 Grand Prix  Road bicycle tires927
      Bicycle Tires16

30426 Himalaya TiresMountain bicycle tires933
                  18.25

 30433New Air PumpTire pump         940
                  20

32779 Slaker Water Water bottle     1286
      Bottle         7

32861 Safe-T HelmetBicycle helmet   1368
1829                60

40421 Alexeyer ProStraight bar      928
      Lifting Bar
1381              65

40422 Pro Curling BarCurling bar    929
1382              50
```

S_PRODUCT Data (continued)

```
 ID      NAME        SHORT_DESC     LONGTEXT_ID
----     ------------------------------------
IMAGE_ID SUGGESTED_WHLSL_PRICEWHLSL_UNITS
-------- ------------------------------
40421    Alexeyer ProStraight bar      928
         Lifting Bar
1381                 65

40422    Pro Curling BarCurling bar    929
1382                 50

41010    Prostar 10  Ten pound weight  517
         Pound Weight
                      8

41020    Prostar 20  Twenty pound weight527
         Pound Weight
                     12

41050    Prostar 50  Fifty pound weight557
         Pound Weight
                     25

41080    Prostar 80  Eighty pound weight587
         Pound Weight
                     35

41100    Prostar 100   One hundred pound 607
         Pound Weightweight
                     45

 50169   Major League Baseball         676
         Baseball
1119                 4.29
```

S_PRODUCT Data (continued)

```
ID    NAME        SHORT_DESC        LONGTEXT_ID
----  ----------------------------------------
IMAGE_IDSUGGESTED_WHLSL_PRICEWHLSL_UNITS
---------------------------------------
 50273Chapman HelmetBatting helmet     780
1223              22.89

50417 Griffey GloveOutfielder’s glove  924
1367                80

50418 Alomar GloveInfielder’s glove    925
1368                75

50419 Steinbach GloveCatcher’s glove   926
1369                80

 50530Cabrera Bat Thirty inch bat      1037
1480                45

50532 Puckett Bat Thirty-two inch bat  1039
1482                47

50536 Winfield BatThirty-six inch bat  1043
1486                50
```

Note: This display has been formatted.

S_REGION Description and Data

```
Column Name       Null?          Datatype
-------------------------------  ----------------
ID                NOT NULL       NUMBER(7)
NAME              NOT NULL       VARCHAR2(50)


SQL> SELECT * FROM s_region;

 ID   NAME
---   -------------------------
  1   North America
  2   South America
  3   Africa / Middle East
  4   Asia
  5   Europe
```

S_TITLE Description and Data

```
Column Name                    Null?          Datatype
-------------------------------- ---------------
TITLE                          NOT NULL       VARCHAR2(25)


SQL> SELECT * FROM s_title;

TITLE
------------------------
President
Sales Representative
Stock Clerk
VP, Administration
VP, Finance
VP, Operations
VP, Sales
Warehouse Manager

8 rows selected.
```

Oracle8 Objects: Types, Tables

emp_type_ObjCol

```
Name                 Null?             Type
-------------------- ------------      ----------------
ID                                     NUMBER(7)
LAST_NAME                              VARCHAR2(25)
FIRST_NAME                             VARCHAR2(25)
USERID                                 VARCHAR2(8)
START_DATE                             DATE
MANAGER_ID                             NUMBER(7)
TITLE                                  VARCHAR2(25)
DEPT_ID                                DEPT_TYPE
SALARY                                 NUMBER(11,2)
COMMISSION_PCT                         NUMBER(4,2)
```

emp_type_RefCol

```
Name                 Null?             Type
-------------------- ------------      ----------------
ID                                     NUMBER(7)
LAST_NAME                              VARCHAR2(25)
FIRST_NAME                             VARCHAR2(25)
USERID                                 VARCHAR2(8)
START_DATE                             DATE
MANAGER_ID                             NUMBER(7)
TITLE                                  VARCHAR2(25)
DEPT_ID                                REF OF DEPT_TYPE
SALARY                                 NUMBER(11,2)
COMMISSION_PCT                         NUMBER(4,2)
```

dept_type

```
--------------------------------       ----------------
ID                                     NUMBER
NAME                                   VARCHAR2(25)
REGION_ID                              NUMBER(7)
```

Tables

oo_emp_Table_ObjCol

```
Name                 Null?            Type
-------------------- -------------    ----------------
ID                                    NUMBER(7)
LAST_NAME                             VARCHAR2(25)
FIRST_NAME                            VARCHAR2(25)
USERID                                VARCHAR2(8)
START_DATE                            DATE
MANAGER_ID                            NUMBER(7)
TITLE                                 VARCHAR2(25)
DEPT_ID                               DEPT_TYPE
SALARY                                NUMBER(11,2)
COMMISSION_PCT                        NUMBER(4,2)
   ID     LAST_NAME                   FIRST_NAME     USERID
--------- --------------------       --------------- --------
START_DAT MANAGER_ID TITLE
--------- ---------- ------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
    SALARY COMMISSION_PCT
---------- --------------
        1Velasquez               Carmencvelasqu
03-MAR-90                            President
DEPT_TYPE(50, 'Administration', 1)
     2500

        2Ngao                    LaDorislngao
08-MAR-90                            1 VP Operations
DEPT_TYPE(41, 'Operations', 1)
     1450

        3Nagayama                Midorimnagayam
17-JUN-91                            1 VP Sales
DEPT_TYPE(31, 'Sales', 1)
     1400
```

```
ID             LAST_NAME      FIRST_NAME     USERID
----------     ------------------------------ --------
START_DAT MANAGER_ID    TITLE
--------- ----------    ------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
    SALARY COMMISSION_PCT
---------- --------------
   4           Quick-To-See   Mark           mquickto
07-APR-90              1 VP Finance
DEPT_TYPE(10, 'Finance', 1)
     1450

        5      Ropeburn       Audry          aropebur
04-MAR-90              1 VP Administration
DEPT_TYPE(50, 'Administration', 1)
     1550

   6           Urguhart       Molly          murguhar
18-JAN-91              2 Warehouse Manager
DEPT_TYPE(41, 'Operations', 1)
     1200

   7           Menchu         Roberta        rmenchu
14-MAY-90              2 Warehouse Manager
DEPT_TYPE(42, 'Operations', 2)
     1250

        8      Biri           Ben            bbiri
07-APR-90              2 Warehouse Manager
DEPT_TYPE(43, 'Operations', 3)
     1100

        9      Catchpole      Antoinette     acatchpo
09-FEB-92              2 Warehouse Manager
DEPT_TYPE(44, 'Operations', 4)
     1300
```

```
ID          LAST_NAME      FIRST_NAME        USERID
----------  -----------------------------  --------
START_DAT MANAGER_ID TITLE
--------- ---------- ------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
   SALARY COMMISSION_PCT
---------- --------------
10          Havel          Marta             mhavel
27-FEB-91           2 Warehouse Manager
DEPT_TYPE(45, 'Operations', 5)
     1307

       11   Magee          Colin             cmagee
14-MAY-90           3 Sales Representative
DEPT_TYPE(31, 'Sales', 1)
     1400   10

12          Giljum         Henry             hgiljum
18-JAN-92           3 Sales Representative
DEPT_TYPE(32, 'Sales', 2)
     1490   13

       13   Sedeghi        Yasmin            ysedeghi
18-FEB-91           3 Sales Representative
DEPT_TYPE(33, 'Sales', 3)
     1515   10

14          Nguyen         Mai               mnguyen
22-JAN-92           3 Sales Representative
DEPT_TYPE(34, 'Sales', 4)
     1525   15

       15   Dumas          Andre             adumas
09-OCT-91           3 Sales Representative
DEPT_TYPE(35, 'Sales', 5)
     145018
```

```
ID             LAST_NAME      FIRST_NAME        USERID
----------     -------------- ---------------   --------
START_DAT MANAGER_ID TITLE
--------- ---------- -------------------------
DEPT_ID(ID, NAME, REGION_ID)
-------------------------------------------------------------
    SALARY COMMISSION_PCT
---------- --------------
   16          Maduro         Elena             emaduro
07-FEB-92            6 Stock Clerk
DEPT_TYPE(41, 'Operations', 1)
     1400

       17      Smith          George            gsmith
08-MAR-90            6 Stock Clerk
DEPT_TYPE(41, 'Operations', 1)
      940

   18          Nozaki         Akira             anozaki
09-FEB-91            7 Stock Clerk
DEPT_TYPE(42, 'Operations', 2)
     1200

       19      Patel          Vikram            vpatel
06-AUG-91            7 Stock Clerk
DEPT_TYPE(42, 'Operations', 2)
      795

   20          Newman         Chad              cnewman
21-JUL-91            8 Stock Clerk
DEPT_TYPE(43, 'Operations', 3)
      750

       21      Markarian      Alexander         amarkari
26-MAY-91            8 Stock Clerk
DEPT_TYPE(43, 'Operations', 3)
      850
```

```
ID          LAST_NAME      FIRST_NAME        USERID
----------  -----------------------------  --------
START_DAT MANAGER_ID    TITLE
--------- ----------    ------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
   SALARY COMMISSION_PCT
---------- --------------
 22         Chang          Eddie             echang
30-NOV-90               9 Stock Clerk
DEPT_TYPE(44, 'Operations', 4)
      800

       23   Patel          Radha             rpatel
17-OCT-90               9 Stock Clerk
DEPT_TYPE(34, 'Sales', 4)
      795

 24         Dancs          Bela              bdancs
17-MAR-91              10 Stock Clerk
DEPT_TYPE(45, 'Operations', 5)
      860

       25   Schwartz       Sylvie            sschwart
09-MAY-91              10 Stock Clerk
DEPT_TYPE(45, 'Operations', 5)
     1100
```

oo_emp_Table_RefCol

```
Name              Null?              Type
-------------------------------      ----------------
ID                                   NUMBER(7)
LAST_NAME                            VARCHAR2(25)
FIRST_NAME                           VARCHAR2(25)
USERID                               VARCHAR2(8)
START_DATE                           DATE
MANAGER_ID                           NUMBER(7)
TITLE                                VARCHAR2(25)
DEPT_ID                              REF OF DEPT_TYPE
SALARY                               NUMBER(11,2)
COMMISSION_PCTNUMBER(4,2)
```

```
ID          LAST_NAME         FIRST_NAME        USERID
----------  --------------------------------  --------
START_DAT MANAGER_ID TITLE
--------- ---------- -------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
    SALARY COMMISSION_PCT
---------- --------------
   1        Velasquez         Carmen            cvelasqu
03-MAR-90   President
0000220208447F54A9ED64676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      2500

        2   Ngao              LaDoris           lngao
08-MAR-90   1 VP Operations
0000220208447F54A9ED5F676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      1450

        3   Nagayama          Midori            mnagayam
17-JUN-91   1 VP Sales
0000220208447F54A9ED5A676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      1400

        4   Quick-To-See      Mark              mquickto
07-APR-90   1 VP Finance
0000220208447F54A9ED59676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      1450

        5   Ropeburn          Audry             aropebur
04-MAR-90   1 VP Administration
0000220208447F54A9ED64676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      1550
```

```
ID         LAST_NAME           FIRST_NAME      USERID
---------- ------------------- --------------- --------
START_DAT MANAGER_ID TITLE
--------- ---------- -------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
    SALARY COMMISSION_PCT
---------- --------------
   6       Urguhart            Molly           murguhar
18-JAN-91  2 Warehouse Manager
0000220208447F54A9ED5F676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1200

        9  Catchpole           Antoinette      acatchpo
09-FEB-92  2 Warehouse Manager
0000220208447F54A9ED62676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1300

       10  Havel               Marta           mhavel
27-FEB-91  2 Warehouse Manager
0000220208447F54A9ED63676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1307

       11  Magee               Colin           cmagee
14-MAY-90  3 Sales Representative
0000220208447F54A9ED5A676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1400            10

       13  Sedeghi             Yasmin          ysedeghi
18-FEB-91  3 Sales Representative
0000220208447F54A9ED5C676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1515            10
```

```
ID          LAST_NAME          FIRST_NAME      USERID
----------  ------------------ --------------- --------
START_DAT MANAGER_ID TITLE
--------- ---------- -------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
   SALARY COMMISSION_PCT
---------- --------------
       14   Nguyen             Mai             mnguyen
22-JAN-92   3 Sales Representative
0000220208447F54A9ED5D676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1525            15

       15   Dumas              Andre           adumas
09-OCT-91   3 Sales Representative
0000220208447F54A9ED5E676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1450            18

       16   Maduro             Elena           emaduro
07-FEB-92   6 Stock Clerk
0000220208447F54A9ED5F676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1400

       17   Smith              George          gsmith
08-MAR-90   6 Stock Clerk
0000220208447F54A9ED5F676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      940

       18   Nozaki             Akira           anozaki
09-FEB-91   7 Stock Clerk
0000220208447F54A9ED60676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1200
```

```
ID          LAST_NAME         FIRST_NAME      USERID
----------  --------------------------------- --------
START_DAT MANAGER_ID TITLE
--------- ---------- -------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
   SALARY COMMISSION_PCT
---------- --------------
       19  Patel             Vikram          vpatel
06-AUG-91   7 Stock Clerk
0000220208447F54A9ED60676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      795

       20  Newman            Chad            cnewman
21-JUL-91   8 Stock Clerk
0000220208447F54A9ED61676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      750

       21  Markarian         Alexander       amarkari
26-MAY-91        8 Stock Clerk
0000220208447F54A9ED61676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      850

       22  Chang             Eddie           echang
30-NOV-90         9 Stock Clerk
0000220208447F54A9ED62676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      800

       23  Patel             Radha           rpatel
17-OCT-90         9 Stock Clerk
0000220208447F54A9ED5D676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      795
```

```
ID          LAST_NAME         FIRST_NAME      USERID
----------  --------------------------------- --------
START_DAT MANAGER_ID TITLE
--------- ---------- -------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
    SALARY COMMISSION_PCT
---------- --------------
       24   Dancs             Bela            bdancs
17-MAR-91   10 Stock Clerk
0000220208447F54A9ED63676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      860

       25   Schwartz          Sylvie          sschwart
09-MAY-91   10 Stock Clerk
0000220208447F54A9ED63676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1100
```

oo_dept_table

```
Name                    Null?            Type
--------------------    -------------    ----------------
ID                                       NUMBER
NAME                                     VARCHAR2(25)
REGION_ID                                NUMBER(7)


ID                      NAME             REGION_ID
--------------------    -------------    ----------------
10                      Finance          1
31                      Sales            1
32                      Sales            2
33                      Sales            3
34                      Sales            4
35                      Sales            5
41                      Operations       1
42                      Operations       2
43                      Operations       3
44                      Operations       4
45                      Operations       5
50                      Administration   1
```

rel_emp_Table_Objcol

```
Name                      Null?            Type
--------------------      -------------    ----------------
ID                                         NUMBER(7)
LAST_NAME                                  VARCHAR2(25)
FIRST_NAME                                 VARCHAR2(25)
USERID                                     VARCHAR2(8)
START_DATE                                 DATE
MANAGER_ID                                 NUMBER(7)
TITLE                                      VARCHAR2(25)
DEPT_ID                                    DEPT_TYPE
SALARY                                     NUMBER(11,2)
COMMISSION_PCT                             NUMBER(4,2)


ID          LAST_NAME         FIRST_NAME      USERID
----------  --------------------------------- --------
START_DAT MANAGER_ID TITLE
--------- ---------- -------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
   SALARY COMMISSION_PCT
---------- --------------
 1         Velasquez         Carmen          cvelasqu
03-MAR-90   President
DEPT_TYPE(50, 'Administration', 1)
     2500

        2   Ngao              LaDoris         lngao
08-MAR-90   1 VP Operations
DEPT_TYPE(41, 'Operations', 1)
     1450

        3   Nagayama          Midori          mnagayam
17-JUN-91         1 VP Sales
DEPT_TYPE(31, 'Sales', 1)
     1400
```

```
ID          LAST_NAME         FIRST_NAME      USERID
----------  --------------------------------- --------
START_DAT MANAGER_ID TITLE
--------- ---------- ------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
   SALARY COMMISSION_PCT
---------- --------------
   4        Quick-To-See      Mark            mquickto
07-APR-90   1 VP Finance
DEPT_TYPE(10, 'Finance', 1)
     1450

        5   Ropeburn          Audry           aropebur
04-MAR-90                  1 VP Administration
DEPT_TYPE(50, 'Administration', 1)
     1550

        6   Urguhart          Molly           murguhar
18-JAN-91                  2 Warehouse Manager
DEPT_TYPE(41, 'Operations', 1)
     1200

        7   Menchu            Roberta         rmenchu
14-MAY-90                  2 Warehouse Manager
DEPT_TYPE(42, 'Operations', 2)
     1250

        8   Biri              Ben             bbiri
07-APR-90                  2 Warehouse Manager
DEPT_TYPE(43, 'Operations', 3)
     1100

        9   Catchpole         Antoinette      acatchpo
09-FEB-92                  2 Warehouse Manager
DEPT_TYPE(44, 'Operations', 4)
     1300
```

```
ID          LAST_NAME                      FIRST_NAME      USERID
----------  ------------------------------ --------
START_DAT MANAGER_ID TITLE
--------- ---------- ------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
    SALARY COMMISSION_PCT
---------- --------------
   10       Havel             Marta           mhavel
27-FEB-91                 2 Warehouse Manager
DEPT_TYPE(45, 'Operations', 5)
     1307

       11  Magee             Colin           cmagee
14-MAY-90                 3 Sales Representative
DEPT_TYPE(31, 'Sales', 1)
     1400            10

       12  Giljum            Henry           hgiljum
18-JAN-92                 3 Sales Representative
DEPT_TYPE(32, 'Sales', 2)
     1490            13

       13  Sedeghi           Yasmin          ysedeghi
18-FEB-91                 3 Sales Representative
DEPT_TYPE(33, 'Sales', 3)
     1515            10

       14  Nguyen            Mai             mnguyen
22-JAN-92                 3 Sales Representative
DEPT_TYPE(34, 'Sales', 4)
     1525            15

       15 Dumas                    Andre      adumas
09-OCT-91                 3 Sales Representative
DEPT_TYPE(35, 'Sales', 5)
     1450            18
```

```
ID          LAST_NAME         FIRST_NAME      USERID
----------  --------------------------------- --------
START_DAT MANAGER_ID TITLE
--------- ---------- ------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
   SALARY COMMISSION_PCT
---------- --------------
       16  Maduro            Elena           emaduro
07-FEB-92                 6 Stock Clerk
DEPT_TYPE(41, 'Operations', 1)
     1400

       17  Smith             George          gsmith
08-MAR-90                 6 Stock Clerk
DEPT_TYPE(41, 'Operations', 1)
      940

       18  Nozaki            Akira           anozaki
09-FEB-91                 7 Stock Clerk
DEPT_TYPE(42, 'Operations', 2)
     1200

       19  Patel             Vikram          vpatel
06-AUG-91                 7 Stock Clerk
DEPT_TYPE(42, 'Operations', 2)
      795

       20  Newman            Chad            cnewman
21-JUL-91                 8 Stock Clerk
DEPT_TYPE(43, 'Operations', 3)
      750

       21  Markarian         Alexander       amarkari
26-MAY-91                 8 Stock Clerk
DEPT_TYPE(43, 'Operations', 3)
      850
```

```
ID          LAST_NAME         FIRST_NAME      USERID
----------  --------------------------------- --------
START_DAT MANAGER_ID TITLE
--------- ---------- ------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
   SALARY COMMISSION_PCT
---------- --------------
       22  Chang             Eddie           echang
30-NOV-90                 9 Stock Clerk
DEPT_TYPE(44, 'Operations', 4)
      800

       23  Patel             Radha           rpatel
17-OCT-90                 9 Stock Clerk
DEPT_TYPE(34, 'Sales', 4)
      795

       24  Dancs             Bela            bdancs
17-MAR-91                10 Stock Clerk
DEPT_TYPE(45, 'Operations', 5)
      860

       25  Schwartz          Sylvie          sschwart
09-MAY-91                10 Stock Clerk
DEPT_TYPE(45, 'Operations', 5)
     1100
```

rel_emp_Table_RefCol

```
Name                    Null?            Type
--------------------    -------------    ----------------
ID                                       NUMBER(7)
LAST_NAME                                VARCHAR2(25)
FIRST_NAME                               VARCHAR2(25)
USERID                                   VARCHAR2(8)
START_DATE                               DATE
MANAGER_ID                               NUMBER(7)
TITLE                                    VARCHAR2(25)
DEPT_ID                                  REF OF DEPT_TYPE
SALARY                                   NUMBER(11,2)
COMMISSION_PCT                           NUMBER(4,2)
```

```
ID          LAST_NAME         FIRST_NAME      USERID
----------  --------------------------------- --------
START_DAT MANAGER_ID TITLE
--------- ---------- -------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
    SALARY COMMISSION_PCT
---------- --------------
   1        Velasquez         Carmen          cvelasqu
03-MAR-90                 President
0000220208447F54A9ED64676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     2500

        2   Ngao              LaDoris         lngao
08-MAR-90                 1 VP Operations
0000220208447F54A9ED5F676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1450

        3   Nagayama          Midori          mnagayam
17-JUN-91                 1 VP Sales
0000220208447F54A9ED5A676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1400

        4   Quick-To-See      Mark            mquickto
07-APR-90                 1 VP Finance
0000220208447F54A9ED59676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1450

        5   Ropeburn          Audry           aropebur
04-MAR-90                 1 VP Administration
0000220208447F54A9ED64676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1550
```

```
ID          LAST_NAME         FIRST_NAME      USERID
----------  --------------------------------- --------
START_DAT MANAGER_ID TITLE
--------- ---------- ------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
   SALARY COMMISSION_PCT
---------- --------------
        6  Urguhart          Molly           murguhar
18-JAN-91                 2 Warehouse Manager
0000220208447F54A9ED5F676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1200

        9  Catchpole         Antoinette      acatchpo
09-FEB-92                 2 Warehouse Manager
0000220208447F54A9ED62676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1300

       10  Havel             Marta           mhavel
27-FEB-91                 2 Warehouse Manager
0000220208447F54A9ED63676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1307

       11  Magee             Colin           cmagee
14-MAY-90                 3 Sales Representative
0000220208447F54A9ED5A676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1400           10

       13  Sedeghi           Yasmin          ysedeghi
18-FEB-91                 3 Sales Representative
0000220208447F54A9ED5C676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1515           10
```

```
ID          LAST_NAME         FIRST_NAME      USERID
----------  --------------------------------- --------
START_DAT MANAGER_ID TITLE
--------- ---------- -------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
    SALARY COMMISSION_PCT
---------- --------------
       14  Nguyen            Mai             mnguyen
22-JAN-92                 3 Sales Representative
0000220208447F54A9ED5D676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1525            15

       15  Dumas             Andre           adumas
09-OCT-91                 3 Sales Representative
0000220208447F54A9ED5E676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1450            18

       16  Maduro            Elena           emaduro
07-FEB-92                 6 Stock Clerk
0000220208447F54A9ED5F676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1400

       17  Smith             George          gsmith
08-MAR-90                 6 Stock Clerk
0000220208447F54A9ED5F676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      940

       18  Nozaki            Akira           anozaki
09-FEB-91                 7 Stock Clerk
0000220208447F54A9ED60676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1200
```

```
ID          LAST_NAME         FIRST_NAME      USERID
----------  --------------------------------- --------
START_DAT MANAGER_ID TITLE
--------- ---------- ------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
   SALARY COMMISSION_PCT
---------- --------------
       19  Patel             Vikram          vpatel
06-AUG-91                 7 Stock Clerk
0000220208447F54A9ED60676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      795

       20  Newman            Chad            cnewman
21-JUL-91                 8 Stock Clerk
0000220208447F54A9ED61676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      750

       21  Markarian         Alexander       amarkari
26-MAY-91                 8 Stock Clerk
0000220208447F54A9ED61676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      850

       22  Chang             Eddie           echang
30-NOV-90                 9 Stock Clerk
0000220208447F54A9ED62676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      800

       23  Patel             Radha           rpatel
17-OCT-90                 9 Stock Clerk
0000220208447F54A9ED5D676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      795
```

```
ID          LAST_NAME         FIRST_NAME      USERID
----------  --------------------------------- --------
START_DAT MANAGER_ID TITLE
--------- ---------- -------------------------
DEPT_ID(ID, NAME, REGION_ID)
------------------------------------------------------------
    SALARY COMMISSION_PCT
---------- --------------
       24  Dancs             Bela            bdancs
17-MAR-91                 10 Stock Clerk
0000220208447F54A9ED63676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
      860

       25  Schwartz         Sylvie          sschwart
09-MAY-91                 10 Stock Clerk
0000220208447F54A9ED63676AE03408002072C15B447F54A9ED55676AE03408
002072C15B
     1100
```

C

Frequently Asked Questions

Frequently Asked Questions

These are Frequently Asked Questions that customers have asked when using Oracle Developer Release 2.*x*. As questions and topics related to Oracle Developer Release 6 become available, they will be added to the Metalink website.

Contents

19 SQL*NET Tracing: How do I debug Tools problems?

20 How do I avoid “Demo tables need to be created” error in Forms after the DEMOBLD.SQL script is run?

21 What should I do when the ‘RR’ format changes year to current year using the To_Date or To_Char functions?

22 How are VBX and OCX controls supported using Developer?

23 How do I read or write environment variables in Oracle.ini or Registry?

24 Win NT/95: How do I display “File Open”, “File Save”, and “Select Printer” dialog boxes?

25 How can I view the procedure body that is stored in the database?

26 How do I create PDF Adobe Acrobat output from Oracle Reports?

27 How do I modify Developer resource files so that they parse?

28 How can I display Oracle Graphics or Reports output for use on a Web page?

29 How do I set up Developer Forms on the Web server?

30 How do I change the default date format on the client-side when using Developer?

31 How can I determine session activity with my Developer applications?

32 How do I get a list of environment variables for Oracle Forms and Reports?

33 How do I access Windows Help, Mouse Position, or Internal Messaging System from Developer?

34 How do I pass a Chart Query from Oracle Reports as a lexical parameter to Oracle Graphics?

Frequently Asked Questions and Answers

Accessing Reference Material

The reference numbers quoted in these answers are the index numbers that enable you to search for a document within Metalink if you are a member of the *Oracle Metals Program*, that is support levels Bronze, Silver or Gold. If you do not currently have one of these metal support services, you can call your sales representative to upgrade your support service level.

You can use these numbers for an exact match, or you can search on full words pertaining to the problem you are having, such as BOILERPLATE.

From `http://support.oracle.com/metalink`, select the TOOLS option, followed by the Developer 2000 and Oracle Forms options. From there, select Top Product Articles and enter the text for your search. Click the Advanced button, then select ConText Syntax before starting the search.

Oracle enables access to published bulletins only from Metalink. If you do not find the topic you need, call the support center for help. Material is constantly being added to this site in an effort to give customers fast access to the information they need. All information about Oracle Mercury. Oracle Mercury is an electronic support service, no phone support. Buyers of this service also have assess to Support's information repositories.

Note: These documents can also be obtained by calling the customer support hotline.

Frequently Asked Questions and Answers

1. How do I make the Quit button exit the form without firing the When-Validate-Item trigger on the input field?

   You have to set the Mouse Navigate property of the Quit button to FALSE, and in the associated WHEN-BUTTON-PRESSED trigger, include the following code:

   ```
   EXIT_FORM(no_validate,FULL_ROLLBACK);
   ```

2. Can I port the *FMX* from one operating system to another and when do I need to regenerate the *FMB* to create a new *FMX*?

   FMX files are OS dependent, while *FMB*'s are not. This means *FMB* files can be ported to different platforms while *FMX* files cannot. *FMB* files must be regenerated when they are ported onto different operating systems or different versions of Developer/2000. For example, if the *FMB* is from a 16-bit environment and you are porting to a 32-bit environment (either OS or Developer/2000 release), then you must regenerate the *FMB*.

3. Can I run the Forms Server Listener as an NT Service?

   Yes. Running the listener process as a service has the following benefits:

   a. **It enables the listener to survive the logoff/logon sequences, hence saving the overhead of restarting them for each new user.**

   b. **It also enables the listener to come up and service requests to be processed even when there are no users logged on.**

   To configure a listener process as an NT service, you must obtain the following files from the Windows NT 4.0 Resource Kit utilities:

   SRVANY.EXE and INSTSRV.EXE.

   A new feature in Developer Server Release 6.0 is the Forms Server as an NT Service from the new features overview. On Windows NT, the Forms Server can now be registered as an NT Service. This improves manageability of the Forms Server to start automatically when the machine is restarted, and to continue running when no user is logged on to the console.

   References

   BUL:13000069.6

4 How do I do a non-cartridge (STATIC) implementation of Web-enabled Forms?

In a non-cartridge implementation, you must create a static HTML file for each application. Each static HTML file you create contains hardcoded information specific to the individual application.

References

PR:1032851.6

5 How can I suppress or customize error messages that are displayed in my Oracle Forms?

You can either set the message level using the system variable SYSTEM.MESSAGE_LEVEL or trap the errors using the ON-ERROR or ON-MESSAGE triggers.

6 How do I pass parameters from Froms to Reports?

Reports can be printed from Forms using the RUN_PRODUCT built-in. In RUN_PRODUCT, you can specify parameters to pass from Forms to Reports.

References

PR: 1010797.6

7 How do I insert and display the Year 2000 in Forms?

Forms 4.5 provides the date format 'DD-MON-RR'. The RR format recognizes Year 2000 dates. You can use this date format to insert and display the dates. For additional information on Y2K and Developer date handling, see `http://oracle.com/year2000/`

References

PR:1014870.6

8 How do I create an iconic button in Forms?

Iconic buttons can be displayed ONLY on GUI platforms. Since iconic buttons are graphical objects, they cannot be displayed when running Oracle Forms in character mode.

After creating a button, set the button properties:

a Iconic = TRUE

b Icon Name = {.ico filename}

(This is the name of the icon file that you wish to use. You do not have to specify the file extension .ico on the property sheet).
Note: Iconic buttons for Forms running over the web use GIF or JPEG files.

References

PR:101560.4

9 How do I suppress the message “Do you want to save changes” in a Master-Detail form?

When a Master-Detail block is created with integrity constraints, Forms creates several triggers and procedures. To suppress the above message, you can edit the program unit, CLEAR_ALL_MASTER_DETAILS, to read CLEAR_BLOCK(DO_COMMIT) instead of CLEAR_BLOCK(ASK_COMMIT).

References

PR:1025112.6

10 How do I execute server O/S commands from a Client Forms application?

You can use the HOST command in Oracle Forms to shell out to the operating system where you can run an executable file, UNIX script or DOS batch file, or perform an OS command. You can also use the DBMS_PIPE package to call a database (stored) procedure/function (to initiate server jobs) from Oracle Forms on a client machine.

References

PR:1017477.6

11 How do I make changes to my Forms without using the Form Builder (also known as Forms Designer) tool?

In Developer/2000 Release 1.n, there is no supported way of editing Forms’ definitions outside of the provided GUI Form Builder. Developer/2000 Release 2.0 provides a documented and supported C API which enables you to programatically create and manipulate FMB files.

12 How can I generate more than one Form module in one execution?

Developer/2000 Release 2.0 provides a new tool called “Project Builder” which enables you to “Compile” an entire Developer/2000 project or subproject in a single action. With Developer/2000 Release 1.n, you can create a batch file or script which can be invoked from the HOST command.

References

PR:1024881.6

13 How can I call Windows API functions such as “Windows Help” from my Forms Application?

You can use the ORA_FFI package (documented in the Procedure Builder Programmer’s Reference) to call external C programs. A large number of 32-bit Windows API calls are available to you through the d2kwutil library that ships with Developer/2000 Release 1.3.2 and above. Add information on the demo d2khelp. It can be found under `{oracle_home}/tools/devdem?0/demo/forms/directory`.

For 5.0 (devdem?0) ?=5 and Release 6 is it ?=6.

Reference = Note 47918.1

Also for UNIX platforms Reference = 1027084.6

References

PR:1006964.6

14 How can I change the default font used by Oracle Forms 4.n?

The environment variable FORMS45_DEFAULTFONT can be set to specify the default font used by Oracle Forms 4.0 and 4.5.

References

PR:1008605.6

15 How can I use COUNT_QUERY to save the number of records retrieved by Forms?

In Forms 4.n, you can use COUNT_QUERY or GET_APPLICATION_PROPERTY(QUERY_HITS) in an ON-COUNT trigger.

References

PR:1006300.6

16 How can I use a database sequence with Forms V4.5?

You must create the sequence database object, then access the integer value the sequence generates within Forms.

References

PR:1015345.6

17 How is Lock Processing handled in Forms 4.5?

Lock Processing in Forms 4.5 is handled in one of the following ways:

- **a Forms locks the record as soon as the operator presses a key to enter or edit the value of an item.**
- **b A "SELECT . . . FOR UPDATE" statement will lock all of the rows returned by a query.**
- **c You can set the LOCKING MODE on the Block Property to IMMEDIATE (which is the default) or DELAYED to handle locks.**
- **d You can use the ON-LOCK trigger to explicitly lock the record in a table, define an exception if the record is locked, and handle the exception accordingly.**

References

PR:1004126.6

18 How can I edit Oracle environment variables defined for the 32-bit Oracle Tools in the Windows Registry?

The Oracle environment variables for 32-bit Oracle products, like Developer Release 1.3 and above, on Windows NT and Windows 95 can be edited by invoking the Registry .

For example, common locations for this application are:

- **Windows 95: `C:\WIN95\REGEDIT.EXE`**
- **Windows NT: `C:\WINNT\SYSTEM32\REGEDT32.EXE`**

For more information, please refer to the link below.

References

PR:1017313.6

19 SQL*NET Tracing: How do I debug Tools problems?

Use SQL*Net tracing to turn on a trace, capturing every interaction between any tool and any version of the database.

References

PR:1012290.6

20 How do I avoid "Demo tables need to be created" error in Forms after the DEMOBLD.SQL script is run?

You should modify the DEMOBLD.SQL to create a log file to check for any errors.

References

PR:1014637.6

21 What should I do when the ‘RR’ format changes year to current year using the To_Date or To_Char functions?

The incorrect behavior of TO_DATE and TO_CHAR with the ’RR’ date format element is a result of PL/SQL version 1.X bug 273975. This bug has been fixed in PL/SQL version 2.X incorporated with Developer Release 2.0 and above.

For workarounds please look at the references.

References

PR:1021244.6

Bug:273975

Bug:322579

22 How are VBX and OCX controls supported using Developer?

Visual Basic Controls (VBX) are a 16 bit mechanism and, as such, are only supported on 16 bit ports. OCX controls are supported on both 16 and 32 bit platforms.

References

PR:1011273.6

23 How do I read or write environment variables in Oracle.ini or Registry?

Use the Developer Bonus Pack D2KWUTIL. This is a toolset of Windows API Access Functions available with Developer Release 1.3.2 (32 bit) and above.

References

PR:1015381.4

24 Win NT/95: How do I display ‘File Open’, ‘File Save’, and ‘Select Printer’ dialog boxes?

Use the Developer Bonus Pack D2KWUTIL. This is a toolset of Windows API Access Functions available with Developer Release 1.3.2 (32 bit) and above.

References

PR:1015583.4

25 How can I view the procedure body that is stored in the Database?

You can view the procedure body by selecting from the ALL_SOURCE table.

```
SELECT text FROM ALL_SOURCE
WHERE Owner = 'SCOTT'
And Name = 'FOO';
```

Note: Name refers to the name of the procedure. Owner and name MUST be in uppercase.

26 How do I create PDF Adobe Acrobat output from Oracle Reports?

You need to install Oracle Reports 2.5.5.1.1 or higher and then set

DESFORMAT=PDF,

DESTYPE=FILE

References

PR:1020512.6

27 How do I modify Developer resource files so that they parse?

You can modify the resource file so it can be parsed by changing the location to the character resource file directory. For more information, please refer to the link below.

References

PR:1025002.6

28 How can I display Oracle Graphics or Oracle Reports output for use on a Web page?

Developer Release 2.0 and above will support interactive Web Graphics.

References

PR:1014974.4

29 How do I set up Developer Forms on the Webserver?

You need to install and configure the webserver to run forms on the Web. To do this on the Webserver side, you need to create the forms listener, configure the listener, and start the Forms listener. Please refer to the link for a step-by-step explanation.

References

PR:1032192.6

30 How do I change the default date format on the client side when using Developer?

You should set the NLS_DATE_FORMAT variable in Oracle.ini (for 16 bit) and the windows Registry (for 32 bit).

References

PR:1028054.6

31 How can I determine session activity with my Developer applications?

For example:

I want to check to see if an application is already running and switch to it if required.

I want to detect inactivity in a Developer application that is running (time out).

I want to determine if another application is running—For example: Microsoft WORD, Oracle Interoffice, Netscape Explorer, and so on.

Use the Developer Bonus Pack D2KWUTIL. This is a toolset of Windows API Access Functions available with Developer/ Release 1.3.2 (32 bit) and above.

References

PR:1015584.4

32 How do I get a list of environment variables for Oracle Forms and Reports?

You can get a list of the environment variables for Oracle Forms and Reports for UNIX and Windows from the bulletin referenced below.

References

Bul:107330.728

33 How do I access Windows Help, Mouse Position, or Internal Messaging System from Developer?

Use the WIN_API_SHELL package in the D2KWUTIL.PLL library that comes with Developer. This is a toolset of Windows API Access Functions available with Developer Release 1.3.2 (32 bit) and above.

References

PR:1015595.4

34 How do I pass a Chart Query from Oracle Reports as a lexical parameter to Oracle Graphics?

You need to create a procedure in an Oracle Graphics open display trigger which references the lexical parameter and creates the query.

References

PR:1011247.6

D

Oracle Rdb Overview

What Is Oracle Rdb?

Oracle Rdb is a multiuser relational database management system designed and developed by Oracle Corporation for high availability, high performance, and ease of use in large-scale, mission-critical applications. It includes a full set of utilities for database maintenance and an industry-standard version of SQL that allows you to easily create applications and maintain your Oracle Rdb databases.

Oracle Rdb is currently running on over 45 thousand servers at more than 15 thousand sites worldwide. It is used in many large online transaction processing (OLTP) systems and its customer list comprises some of the worlds' leading corporations. Oracle continues to enhance the functionality of Rdb by adding new features and strengthening integration with other Oracle technology and tools.

Supported Platforms

Oracle Rdb is available on the following computing platforms:

- OpenVMS VAX
- OpenVMS Alpha
- Windows NT Intel and Alpha

Oracle Rdb Features

These are some of the features that Oracle Rdb offers:

- Disaster tolerance

  The Hot Standby option is ideal where very high performance and continuous (7 by 365) availability is essential. It provides disaster tolerance across geographically dispersed systems using the existing network with no change in applications or material increment in system overhead. The Hot Standby system can be accessed simultaneously for read-only transactions.

- High performance

  Rdb is optimized to meet enterprise OLTP requirements for the highest levels of performance on OpenVMS Alpha and VAX. Features include full support for 64-bit Very Large Memory on OpenVMS Alpha, mixed-clusters, patented Record Caching; pinning tables in memory for rapid access, a patented Dynamic Query Optimizer, and Query Outlines for repeatable query performance.

- Ease of management

  Rdb provides comprehensive, simple management of Rdb databases. Default database settings provide great performance for Rdb out of the box. The Enterprise Manager DBAPack for Rdb provides a complete set of management tools for Rdb that can be run from the Oracle Enterprise Manager console. The TRACE/Expert option provides performance monitoring and expert database design for Rdb. Rdb8 integration with Oracle Enterprise Manager provides common graphical interfaces to manage Rdb and Oracle RDBMS from the same Windows console.

Oracle Rdb Features (continued)

- Support for Windows NT Intel and Alpha

  Compaq customers can extend their OpenVMS environment to include NT and run Rdb applications on commodity Intel platforms. Rdb8 on NT is a native, multi-threaded, complete implementation of Rdb.

- Integration with other Oracle technologies

  Rdb customers can leverage their investment in Rdb by enhancing existing applications and developing new ones with the latest Oracle tools for Web and client-server application development, and powerful business intelligence analysis. Customers can make a single investment in products and training that is completely portable across Rdb and Oracle RDBMS. Rdb supports Oracle Application Server, Oracle Developer, Oracle Designer, Oracle Discoverer, and Oracle Express Server.

- Support for SQL*Net for Rdb

  This enables integration with the Oracle technologies listed above. Customers and application partners can use standard Oracle APIs to build applications that run on Rdb and Oracle RDBMS. Data sharing between Rdb and Oracle RDBMS is enabled without the use of gateways. Rdb includes support for the Oracle Call Level Interface (OCI), Oracle network protocols, and DB Links.

Other Information

Oracle Rdb Education Oracle Education offers a comprehensive series of Oracle Rdb courses. The course titles include:

- *Introduction to Rdb: Technical Overview and SQL Language*
- *Rdb for the Database Administrator*
- *Rdb for the Database Designer*
- *Rdb 3GL Programming*
- *Rdb Performance and Tuning*
- *Rdb Internals*

To receive the Oracle Education U.S. schedule and catalog, which contains complete course descriptions, pricing, and enrollment information, call 1.800.633.0575 within the continental U.S. or 650.633.5019 from outside the U.S. For international course information, contact the Oracle site nearest to you. You can also get Oracle Education information and order catalogs from the following website: `http://www.oracle.com`.

Other Information (continued)

Oracle Rdb Product Information The Rdb website is a resource for finding out the latest news on the product and its features. Look here for information such as:

- New product releases
- Product interoperability information
- How customers are using Oracle Rdb
- Calendar of product-related events

To access the website, go to the following URL: `http://www.oracle.com/rdb`

Oracle Rdb Demonstration CD-ROM To receive a free Oracle Rdb product demonstration CD-ROM, send an e-mail including your name, address, and company to: `infordb@us.oracle.com`.

Specify that you'd like to receive the free Oracle Rdb CD-ROM.

E

Locking in Form Builder

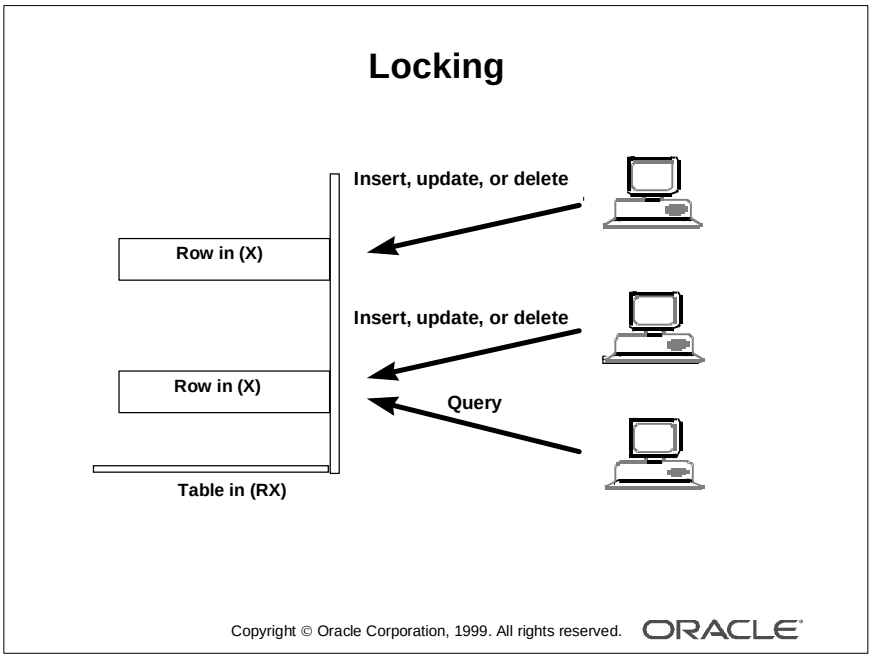
Objectives

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

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

Overview

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


Locking
Insert, update, or delete
Row in (X)
Insert, update, or delete
Row in (X)
Query
Table in (RX)
Copyright © Oracle Corporation, 1999. All rights reserved.
ORACLE

Locking

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

Oracle8 Locking

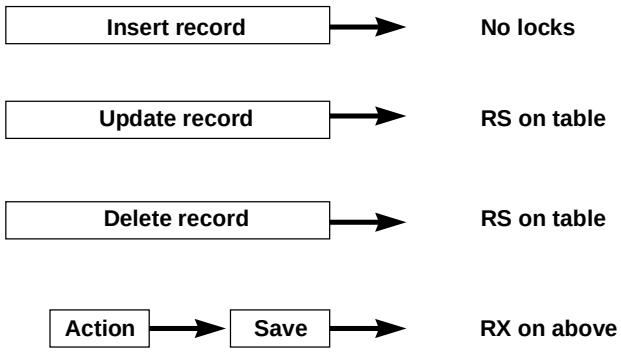
Form Builder applications that connect to an Oracle8 database are subject to the standard locking mechanisms employed by the server. Here is a reminder of the main points:

- Oracle8 uses row-level locking to protect data that is being inserted, updated, or deleted.
- Queries do not prevent other database sessions from performing data manipulation. This also applies to the reverse situation.
- Locks are released at the end of a database transaction (following a rollback or commit).

A session issues locks to prevent other sessions from performing certain actions on a row or table. The main Oracle8 locks that affect Form Builder applications are:

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

Default Locking in Form Builder



Concurrent Updates and Deletes

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

Default Locking in Forms

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

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

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

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

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

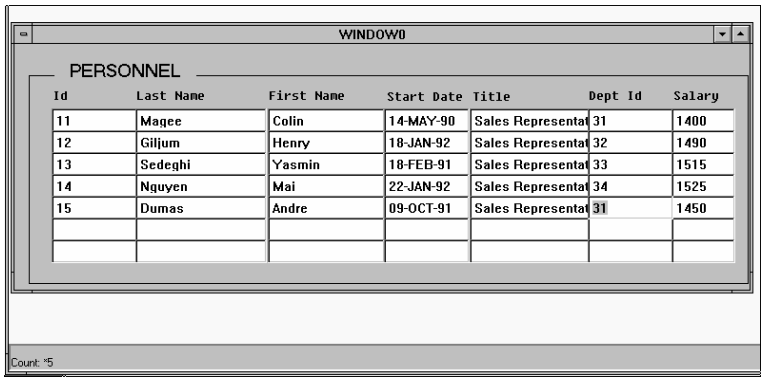
What Happens When Users Compete for the Same Row?

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

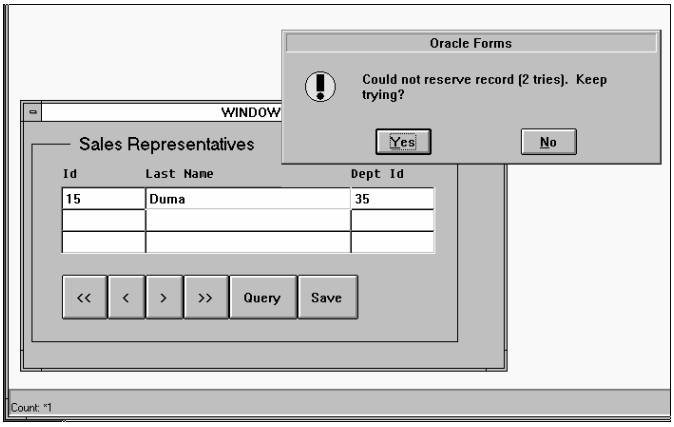
Form Builder also keeps each user informed of these events through messages and alerts when:

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

UserA-Step 1



UserB-Step 2



Example: Two Users Accessing the Same Record

1. UserA is running the Personnel application, and queries the sales representatives. The record for employee 15, Dumas, is updated so that his department is changed to 31.

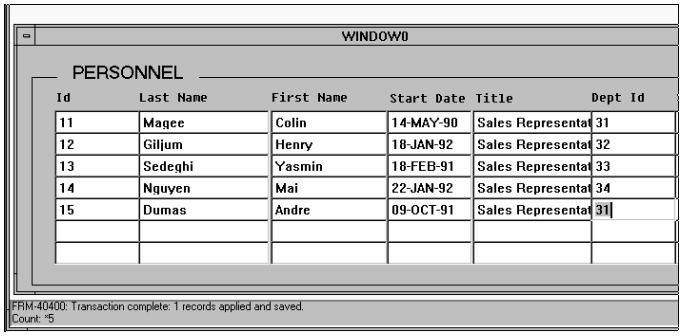
   UserA does *not* save the change at this point in time. The row that corresponds to the changed record is now locked (exclusively). The current situation is shown in the top slide, opposite.

2. UserB is running the Summit application, and has started a form that accesses the sales representatives. Employee Dumas still appears in department 15 in this form, because UserA has not yet saved the change to the database.

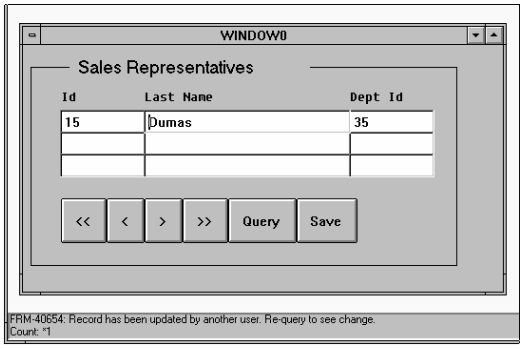
   UserB attempts to update the record by changing the sales representative's name to Agasi. Since this action requests a lock on the row, and this row is already locked by UserA, Forms issues an alert saying the attempt to reserve the record failed. This user can request additional attempts, or reply No, and try later.

   The situation is shown in the lower slide, opposite. UserB replies No. (This results in the fatal error message 40501, which confirms that the original update action has failed).

UserA-Step 3



UserB-Step 4



Example: Two Users Accessing the Same Record (continued)

3. Back in the Personnel form, UserA now saves the change to Dumas' department, which applies the change to the database row, and then releases the lock at the end of the transaction.
   This is shown in the upper slide, opposite.
4. In the Sales Representatives form, UserB may now alter the record. However, since the database row itself has now changed since it was queried in this form, Form Builder tells UserB that it must be requeried before a change can be made.
   This situation is shown in the lower slide, opposite. Once UserB requeries, the record can then be changed.

Concurrent Updates and Deletes

Achieved by:

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

Locking in Triggers

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

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

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

Locking by DML Statements

If you include data manipulation language statements in transactional triggers, their execution causes:

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

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

Locking with Built-Ins

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

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

```
UPDATE s_ord
SET    date_stamp = SYSDATE
WHERE  ROWID = :order.rowid;
```

Locking with Built-in Subprograms

The following built-ins allow locking:

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

On-Lock Trigger

Example

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

On-Lock Trigger

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

Use this trigger to:

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

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

Example

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

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

Summary

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

Summary

Default Locking

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

Locking in Triggers

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

F

Oracle8 Object Features in Oracle Developer

Objectives

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

- **Describe the new Oracle8 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 Object Navigator**

Overview

Introduction

In this lesson, you will review certain object features of Oracle8. This lesson also explains how objects are displayed in the Object Navigator.

Objectives

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

- Describe the new Oracle8 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 Object Navigator

New Oracle8 Scalar Datatypes

- **NCHAR**
- **NVARCHAR2**
- **FLOAT**
- **NLS types**

New Oracle8 Datatypes

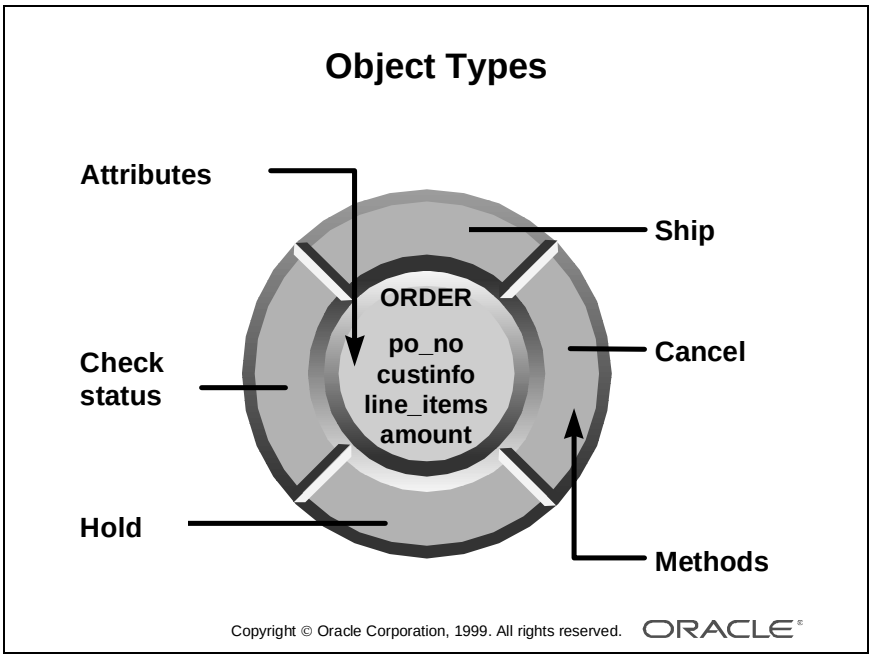
Scalar Datatypes

- NCHAR stores fixed-length (blank-padded if necessary) NLS character data. How the data is represented internally depends on the national character set, which might use a fixed-width encoding such as US7ASCII or a variable-width encoding such as JA16SJIS.
- NVARCHAR2 stores variable-length NLS character data. How the data is represented internally depends on the national character set, which might use a fixed-width encoding such as WE8EBCDIC37C or a variable-width encoding such as JA16DBCS.
- FLOAT is a subtype of NUMBER. However, you cannot specify a scale for FLOAT variables. You can only specify a binary precision, which is the total number of binary digits.

There is no change to the way scalar datatypes are displayed in Oracle Developer. The new datatypes are automatically converted to existing Oracle Developer item datatypes.

NLS Types

Oracle8 offers extended NLS (National Language Support) including national character sets and the datatypes NCHAR and NVARCHAR2, which store NLS data. With NLS, number and date formats adapt automatically to the language conventions specified for a user session. Users around the world can interact with Oracle in their native languages. NLS is discussed in *Oracle8 Server Reference Manual.*


Object Types
Attributes
Ship
ORDER
po_no
custinfo
line_items
amount
Check status
Cancel
Hold
Methods
Copyright © Oracle Corporation, 1999. All rights reserved.
ORACLE

Object Types

An object type is a user-defined composite datatype. Orcale8 requires enough knowledge of a user-defined datatype to interact with it. So, in some sense, an object type is similar to a record type, and in some sense similar to a package.

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

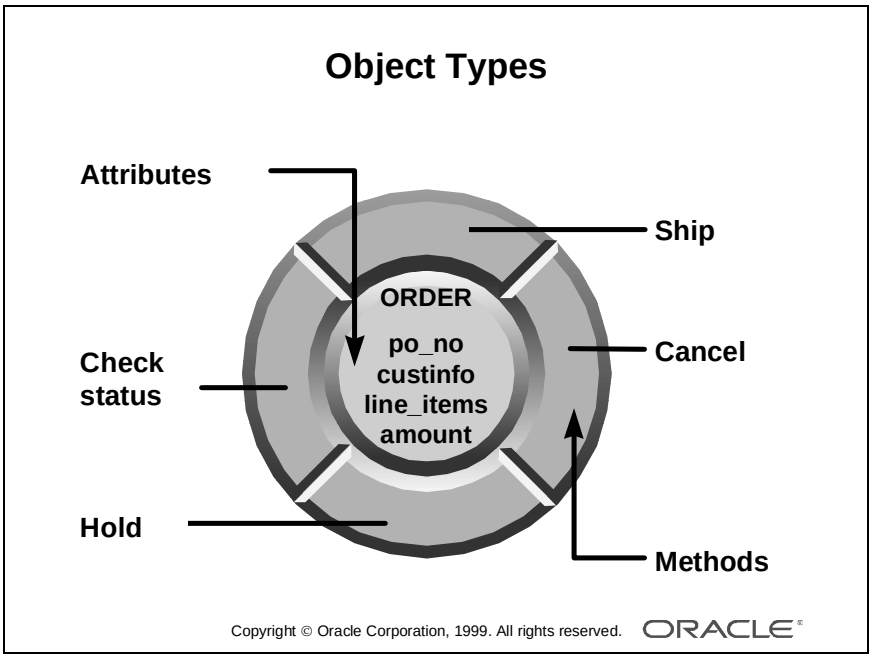
Attributes An object type is similar to a record type in that it is declared to be composed of one or more subparts that are of predefined datatypes. Record types call these subparts *fields*, but object types call these subparts *attributes*.
Attributes define the object structure.

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

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

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

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


Object Types
Attributes
Ship
ORDER
po_no
custinfo
line_items
amount
Check
status
Cancel
Hold
Methods
Copyright © Oracle Corporation, 1999. All rights reserved.
ORACLE

Object Types (continued)

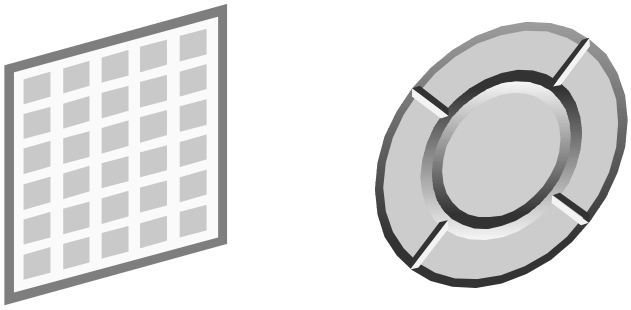
Methods An object type is also similar to a package. Once an object is declared, its attributes are similar to package variables. And like packages, objects types can contain procedures and functions. In object types, these subprograms are known as *methods*. A method describes the behavior of an object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

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

Here is an example of an object type:

```
CREATE TYPE dept_type AS OBJECT
   (dept_idNUMBER(2),
    dnameVARCHAR2(14),
    loc     VARCHAR2(3),
    MEMBER PROCEDURE set_dept_id (d_id NUMBER),
       PRAGMA RESTRICT_REFERENCES (set_dept_id,
         RNDS,WNDS,RNPS,WNPS),
    MEMBER FUNCTION get_dept_id RETURN NUMBER,
       PRAGMA RESTRICT_REFERENCES (get_dept_id,
         RNDS,WNDS,RNPS,WNPS));
CREATE TYPE BODY dept_type AS
   MEMBER PROCEDURE set_dept_id (d_id NUMBER)
   IS
   BEGIN
      dept_id := d_id;
   END;
   MEMBER FUNCTION get_dept_id
      RETURN NUMBER
   IS
   BEGIN
      RETURN (dept_id);
   END;
END;
```

Object Tables



Object table based on object type

Creating Oracle8 Objects

Introduction

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

Object Tables

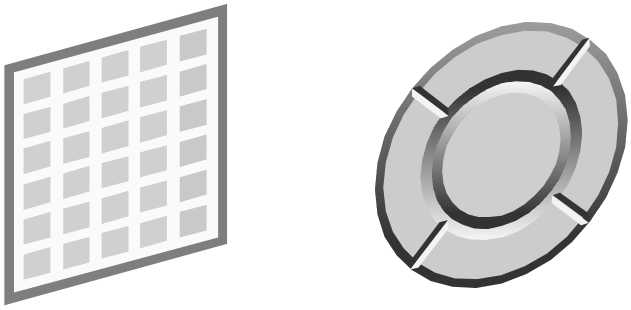
One way to create an object is to create a table whose rows are objects of that object type. Here is an example of an object table declaration:

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

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

Note: REF types are reviewed later.

Object Columns



Object column based on object type

Object Columns

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

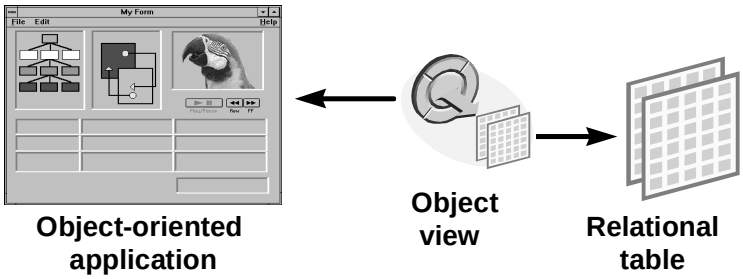
```
CREATE TABLE o_customer (
   custid   NUMBER (6) NOT NULL,
   name     VARCHAR2 (45),
   repid    NUMBER (4) NOT NULL,
   creditlimit NUMBER (9,2),
   address      address_type,
   phone    phone_type);
```

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

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

Note: Object REFs are reviewed later in this section.

Object Views



Object-oriented application

Object view

Relational table

Object views based on object types

Object Views

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

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

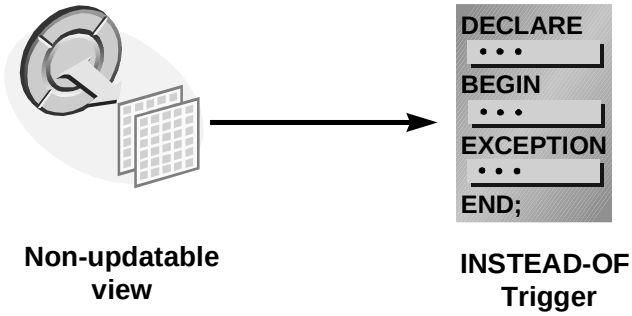
This dilemma is addressed by object views. Like all views, an object view transforms the way a tables appear to a user, without changing the actual structure of the table. Object views make relational tables look like object tables. This allows the developers to postpone converting the data from relational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

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

Note: Object REFs are reviewed later in this section.
Here is an example of an object view creation statement:

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

INSTEAD-OF Triggers



Non-updatable view

INSTEAD-OF Trigger

Object Views (continued)

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

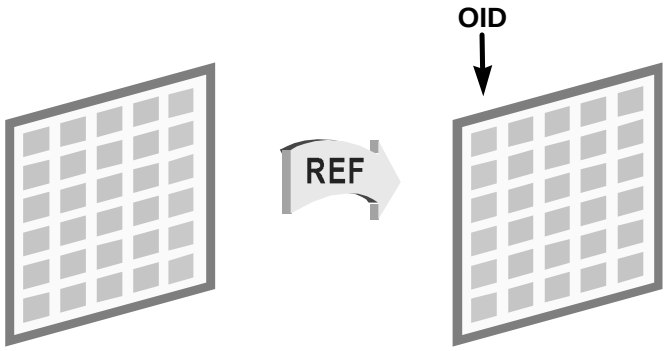
These triggers are called INSTEAD-OF triggers because, unlike other types of triggers, Oracle fires the trigger instead of executing the triggering statement. The trigger performs update, insert, or delete operations directly on the underlying tables.

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

INSTEAD-OF triggers are activated for each row.

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

References to Objects



Referencing Objects

Introduction

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

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

Object REFs

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

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

Or, you can associate two objects by storing the OID of one object in an attribute of another.

The stored copy of the OID then becomes a pointer, or reference (REF), to the original object.

The attribute or column that holds the OID is of datatype REF.

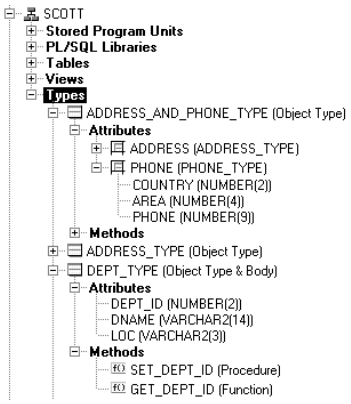
Note: Object columns are not assigned OIDs and cannot be pointed to by a REF.

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

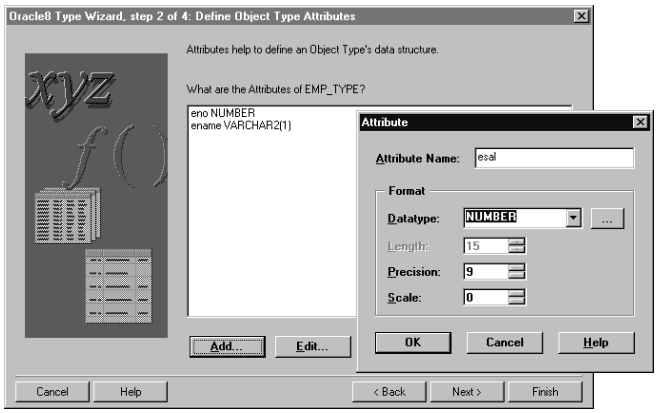
```
CREATE TABLE o_emp
   ( empno  NUMBER(4) NOT NULL,
    ename   VARCHAR2(10),
    job     VARCHAR2(10),
    mgr     NUMBER(4),
    hiredate DATE,
    sal     NUMBER(7,2),
    comm    NUMBER(7,2),
    dept    REF dept_type SCOPE IS o_dept) ;
```

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

Object Types in Object Navigator



Object Type Wizard



Displaying Oracle8 Objects in the Object Navigator

The object navigator lists declared types in the *Database Objects* section, along with tables, views, and other Oracle objects.

Object Types

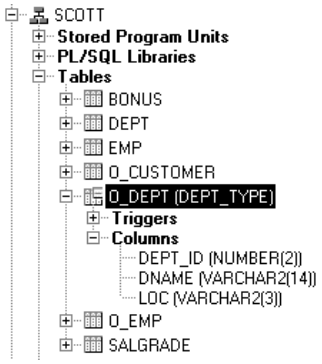
Both the attributes and the methods are listed under each type. Also, the nested types within address_and_phone_type are displayed in an indented sublevel.

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

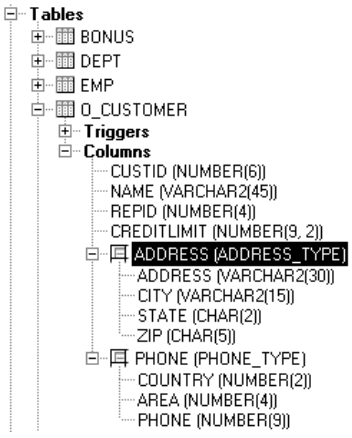
Oracle8 Type Wizard

Object types can be created using the Oracle8 Type Wizard. The wizard allows you to define the attributes and methods.

Object Tables in Object Navigator



Object Columns in Object Navigator



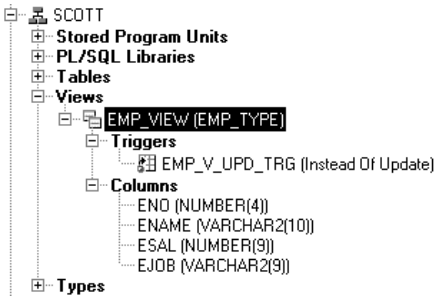
Object Tables

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

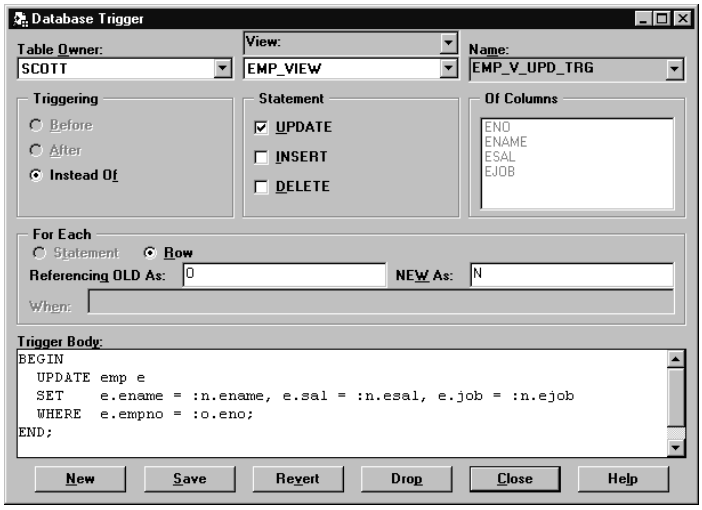
Object Columns

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

Object Views in Object Navigator



INSTEAD-OF Trigger Dialog Box



Object Views

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

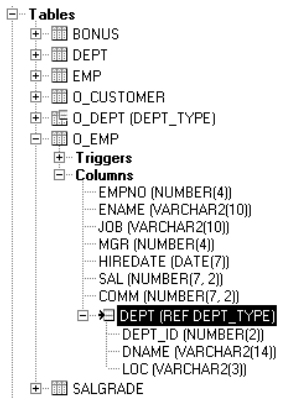
INSTEAD-OF Triggers INSTEAD-OF database triggers can now be created through the trigger creation dialog box, just like any other database trigger.

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

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

Reference For more information about INSTEAD-OF triggers, see *Oracle8 Server SQL Reference Manual* and *Oracle8 Concepts Manual.*

Object REFs in Object Navigator



Object REFs

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

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

Summary

- **Oracle8 introduced three new scalar datatypes.**
- **Objects and object types allow representation of complex data.**
- **Three kinds of objects: 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.**

Summary

New Oracle8 Datatypes

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

Objects

Three kinds of objects are object tables, object columns, and object views. INSTEAD-OF triggers allow DML on object views.

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

Oracle8 Objects in the Object Navigator

The Object Navigator can display certain types of objects.

G

Using the Layout Editor in Oracle Developer

Lesson Objectives

- **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**

Overview

Introduction

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

Objectives

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

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

Using the Layout Editor

Common features:

- **Moving and resizing objects and text**
- **Defining colors and fonts**
- **Importing and manipulating images and drawings**
- **Creating geometric lines and shapes**

Using the Layout Editor

Layout types:

- **Canvases in Forms**
- **Display layers for Graphics**
- **Report layouts**

Why Use the Layout Editor?

The Layout Editor is a graphical tool for defining the arrangement and appearance of visual objects. The Layout Editor opens windows in the Oracle Developer Tool Builders to present the surfaces on which you can arrange objects. Some objects only occur in a subset of the Oracle Developer tools, and so the associated Layout Editor appears only there.

The following are common to each tool:

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

Layout Types

You can use the Layout Editor to control the visual layout in each of the main Oracle Developer Tools. The surfaces include:

- Canvas-views in Forms

  A Canvas is the surface on which you arrange a form's objects. Its View is the portion of that Canvas that is initially visible within a window at run time. You can also see Stacked or Tabbed canvas-views in the Layout Editor; their views might overlay others within the same window. You can also display stacked views in the Layout Editor.

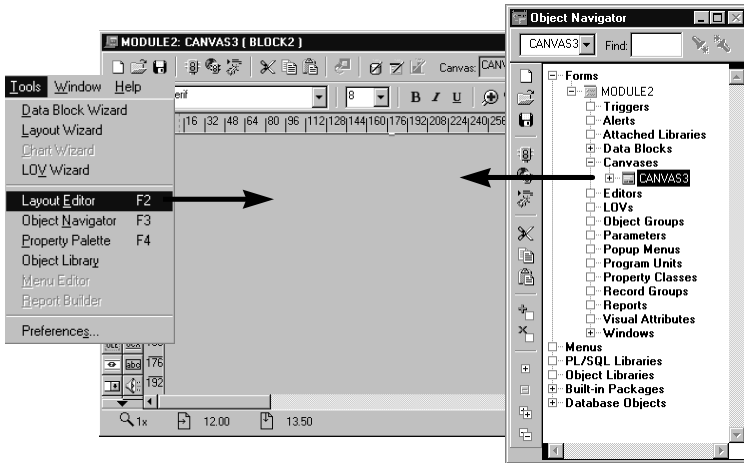
- Display layers in Graphics

  The layers of a Display can contain charts, text, and graphical objects. These layers can be overlaid or hidden when required. Geometric shapes that you create in Graphics can be given their own functionality.

- Layouts in Reports

  Here, the Layout Editor lets you plan the format of a report. This includes the frame structure of the report, as well as buttons, graphics, and background text (boilerplate).

Invoking the Layout Editor



How to Access the Layout Editor

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

Note: In Graphics, the Layout Editor is open while you have a Display open. Reports has three editors, the Layout Editor, the Data Model Editor and the Parameter Form Editor, that all displayed in the same window, one at a time.

Opening from the Object Navigator

Double-click:

- The Layout icon within a report hierarchy, in Reports

  or

- A Canvas-view icon within a form hierarchy in Forms

Opening from the Builder Menus

1. Make sure that you have a context for either a report in Reports or a form in Forms. You can do this by selecting the appropriate objects in the Navigator.
2. In Forms, select Tools—>Layout Editor from the builder menu.
   In Reports, select View—>Layout Model from the builder menu to set the context for the editor window.

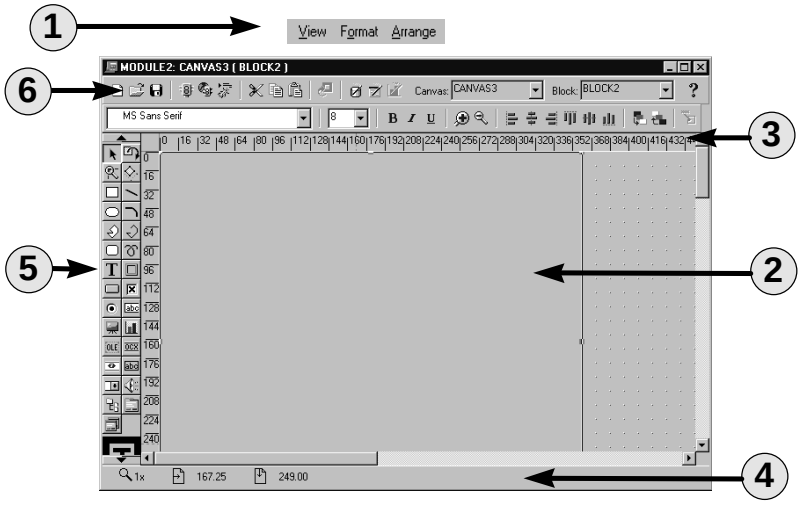
In Forms, you can open several Layout Editor windows—one for each canvas-view. Make sure that you have the canvas you want.

Closing the Layout Editor

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

Closing a layout in Graphics implies that you want to close its entire display (the Graphics module).

Layout Editor: Components



Components of the Layout Editor

Common components of the Layout Editor are:

- Menu facilities

  While the Layout Editor window is active, the Main builder menu changes to include three new items: View, Arrange, and Format. These are submenus for controlling the Layout Editor.

- Horizontal toolbar

  This appears across the top of the window, and is a subset of the tools from the Object Navigator.

- Stylebar

  This appears under the horizontal toolbar, and is a subset of the Format menu. It may also contain some other tools.

- Vertical toolbar

  This contains the tools for creating and modifying objects on the layout. Note that some tools in the palette may be hidden if you have reduced the size of the Layout Editor window. If so, scroll buttons appear on the vertical toolbar.

  There are three types of tools:

  - Graphics tools—for creating and modifying lines and shapes
  - Product-specific tools
  - Manipulation tools—for controlling color and patterns

- Rulers and ruler guides

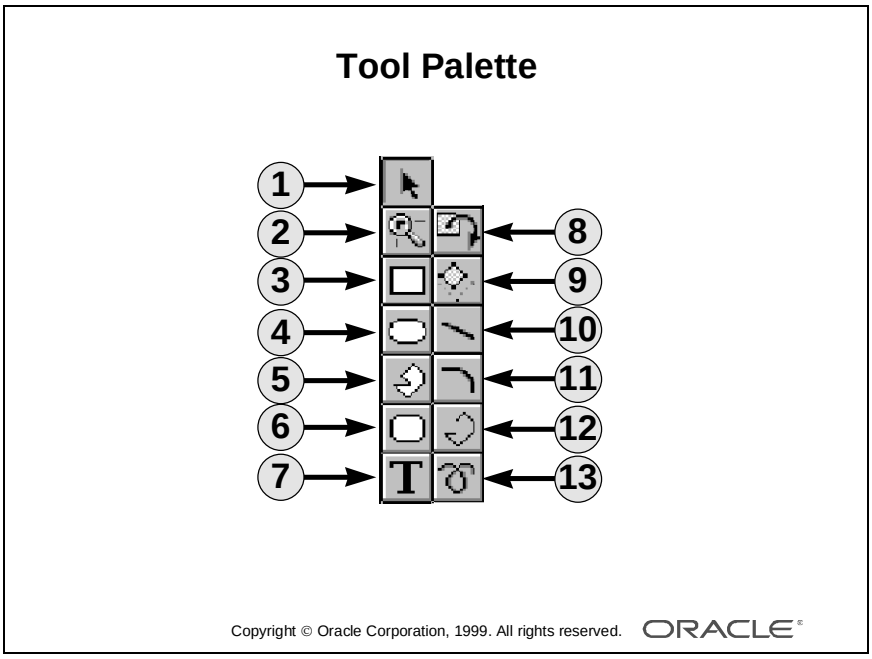
  Rulers are horizontal and vertical markers to aid alignment, and appear at the top and side of the layout region. You can switch these off or have their units altered, as required. Drag ruler guides from the rulers across the layout region to mark positions in the layout.

- Layout/Painting region

  This is the main central/right area where you can place and manipulate objects. A grid pattern is displayed in this area to aid alignment of objects. You can switch off or rescale this grid if required (in Forms, the grid is hidden if the View—>Show Canvas option is switched on).

- Status line

  This appears at the bottom of the window. It shows you the mouse position and drag distance (when moving objects), and the current magnification level.



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

Creating and Modifying Objects in the Layout

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

Creating Lines and Shapes

Create geometric lines and shapes by selecting from the graphics tools in the vertical toolbar. These include:

- Rectangles/squares
- Ellipses/circles
- Polygons and polylines
- Lines and arcs
- Freehand tool

The default tool in the vertical toolbar is Select, which lets you select and move objects.

Creating a New Line or Shape

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

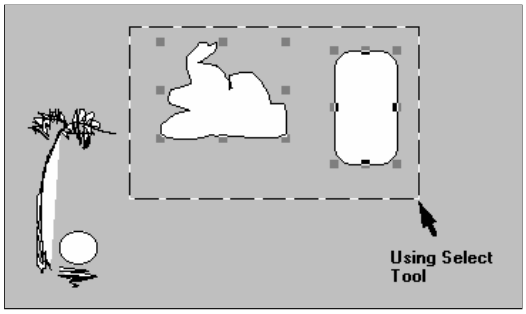
Notice that the object remains selected after this procedure (selection handles appear on its boundaries) until you deselect it by clicking elsewhere.

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

Creating Text

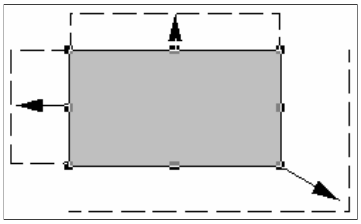
The Text tool (T) lets you open a Boilerplate Text object on the layout. You can type one or more lines of text into this object while it is selected with the Text tool. Uses of Text objects vary according to the Oracle Developer tool in which you create them.

Selecting Objects



Manipulating Objects

Expand/contract in one direction



Expand/contract diagonally

Selecting Objects for Modification

With the Select tool active, you can select one or more objects on the layout to move or modify.

To select *one* object:

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

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

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

To select *several* objects together:

- Hold down the [Shift] key and then click each object to be selected

  or
- Draw a bounding box around the objects (providing the objects are adjacent to each other)

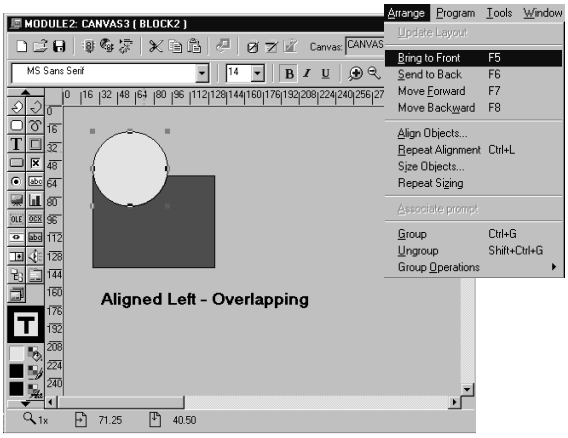
Changing the Size or Ratio

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

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

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

Moving, Aligning, and Overlapping



Moving and Aligning Objects

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

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

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

Note: If you position an object using one grid-snap spacing, and then try to position other objects under different settings, it may prove difficult to align them with each other.

Try to stick to the same snap units, if you use Grid Snap at all.

Overlapping Objects

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

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

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

Groups in the Layout

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

Manipulating Objects As a Group

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

Placing Objects into a Group

1. Select the objects on the layout that are to be grouped together.
2. Select Arrange—>Group from the menu.

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

Groups have different implications depending upon the Oracle Developer tool that is using them. Graphics, for example, can reference a group in the layout programmatically.

Manipulating Individual Group Members

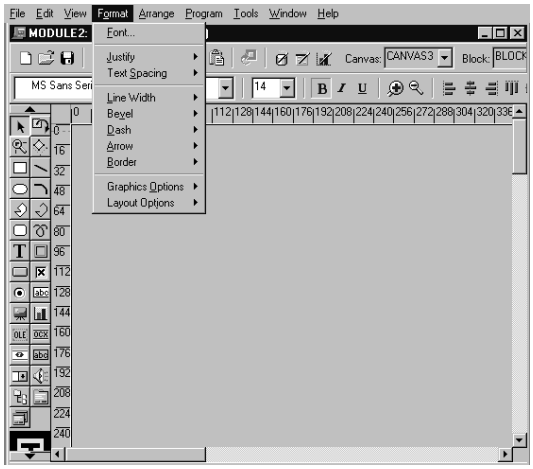
To manipulate group members individually, select the group, then click the individual member. You can use options in the Group Operations menu to remove objects from the group, or to reselect the parent group.

Other Tools for Manipulating Objects

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

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

Format Menu



Formatting Objects in the Layout

The builder's Format menu provides a variety of facilities for changing the style and appearance of objects in the layout. These include:

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

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

Some of the format options are available from the stylebar.

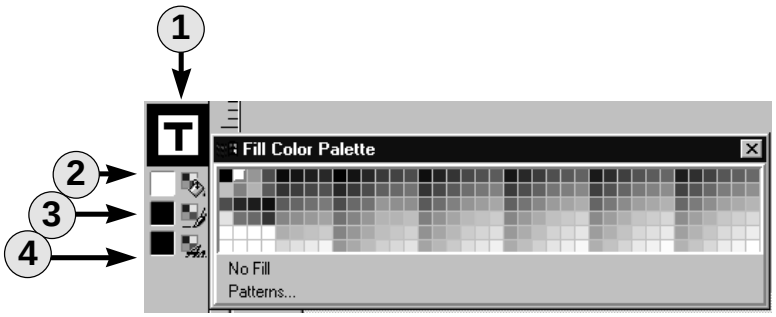
Changing Fonts on Textual Objects

There are a number of ways to change the font characteristics of textual objects; they depend on which Oracle Developer tool you are using. The common methods provided by the Layout Editor are:

1. Select the objects in the layout whose content text you want to change (these may be boilerplate text objects and other textual object types supported by the Oracle Developer tool you are using).
2. Choose the font style and size that you require from the stylebar or select Format from the Main Builder menu, and choose the font style and size that you require.

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

Color and Pattern Tools



1	Sample window
2	Fill color/pattern
3	Line color
4	Text color

Coloring Objects and Text

There are three tools in the vertical toolbar for coloring objects. These are:

- Fill Color: Use this tool to define the colors and pattern for an object's body.
- Line Color: Use this tool to define the color of a line, or the boundary line around an object.
- Text Color: This tool lets you choose the color for the text.

Coloring Objects

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

Coloring an Area

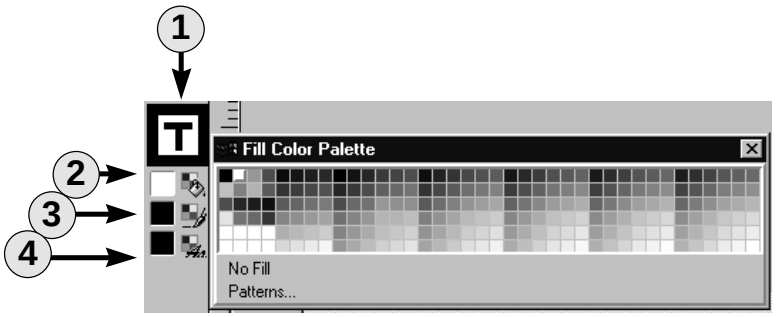
1. Select the objects whose color you want to change.
2. Select the Fill Color tool. The color palette appears.
   If you want the objects to become transparent, choose No Fill at the bottom of the color palette window.
3. Select a color from those visible. If you want the fill area to be patterned instead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

Coloring a Line

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

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

Color and Pattern Tools



1	Sample window
2	Fill color/pattern
3	Line color
4	Text color

Coloring Text

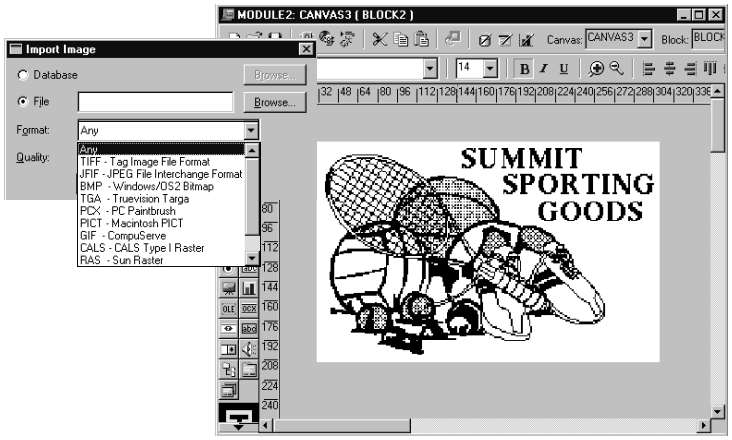
1. Select the textual objects whose color you want to change in the layout.
2. Select the Text Color tool. The color palette appears, showing the available colors.
3. Select your color choice from those shown (click the appropriate square).
4. Notice that the selected objects on the layout have adopted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

Altering the Color Palette

By selecting Format—>Layout Options—>Color Palette from the menu, you can edit the Color palette that is presented when choosing colors. This option is only available when the builder option Color Palette Mode is set to Editable. Changes to the Color palette are saved with the current module.

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

Importing Images



Importing Images and Drawings

Oracle Developer tools allow both static and dynamic bit-mapped images to be integrated into the application. Here, we discuss how you can import static images and drawings; that is, ones that are fixed on the layout. These might include:

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

Importing Drawings

You can import line art onto the layout from either the filesystem or the database. Select File—>Import—>Drawing from the menus, and specify the source in the Import dialog box. The format can be Oracle or a CGM file.

Importing Images

Select File—>Import—>Image from the menus, and set the Import Image options:

Option	**Description**
(Radio buttons) File/Database	Where to import from
Filename	File to be imported
Format	Image format (BMP, TIFF, PCX, PICT, GIF, CALS, RAS, Oracle Format, and PCD [Kodak Photo CD])
Quality	Range from Excellent to Poor. This is a trade off between image resolution and memory required.

Manipulating the Imported Image or Drawing

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

Summary

- **Create objects by:**
  - **Choosing a palette tool**
  - **Clicking and dragging on layout region**
- **Color palette tools for fill area, lines, and text**
- **View, arrange, and format menus available for layout**
- **Objects may be grouped for operations**
- **Import images and drawings by File—>Import**

Summary

- Create objects by:
  - Choosing the correct toolbar tool.
  - Clicking and Dragging the Layout region.
- There are color palette tools for fill area, lines, and text.
- View, Arrange, and Format menus are available while you are in the Layout Editor.
- You can group objects for operations.
- Import images and drawings by selecting Edit—>Import.