CHAPTER 6

TRIGGERS & BUILT-INS

Chapter Objectives

In this Chapter, you will learn about:

Oracle Forms applications come equipped with a significant amount of default processing. That is, when events occur, there is always default code that responds. Quite often this default processing is not enough to give the application the functionality your users require. The problem is that you cannot directly access and edit the code for the default processing to make it do what you want. This is why you write triggers: to complement, augment, or replace this default processing.

In this Chapter, you will delve a bit deeper into trigger and event concepts. You will learn how and when triggers fire, and how they fire in relation to each other. You will also write triggers of your own.

PL/SQL language, syntax and structure will not be discussed here, since it is assumed that you have already had ample experience writing database stored procedures, functions, and triggers.

LAB 6.1

TRIGGER BASICS

LAB OBJECTIVES

After this Lab, you will be able to:

- Use PL/SQL and SQL in Triggers
- Understand Trigger Scope
- Categorize Triggers

In the simplest terms, a trigger contains PL/SQL code that responds to Forms events. You have already been exposed to many triggers and the events that fire them in the previous Labs and Exercises, so you have a general idea of how they work.

One of the advantages of using Oracle Forms and an Oracle database together is that the PL/SQL programming language is used in both of them. So, if you have already written packages and procedures for the database, then you already know how to write triggers in Forms. But, before you begin writing triggers, it is necessary to understand when they fire and how they are organized.

TRIGGER SCOPE

Triggers are always attached to other objects. The level of the object in the Forms hierarchy helps determine the scope of the attached trigger(s). The `ON-POPULATE-DETAILS` trigger in Figure 6.1 is defined at the block level. It is attached to the `COURSE` block in the Object Navigator and will only fire in response to events within the scope of the `COURSE` block.

Triggers can be attached to items and forms as well. Triggers at the item level fire in response to events within the scope of their respective items. Form-level triggers fire in response to events within the scope of the form.

go to contents

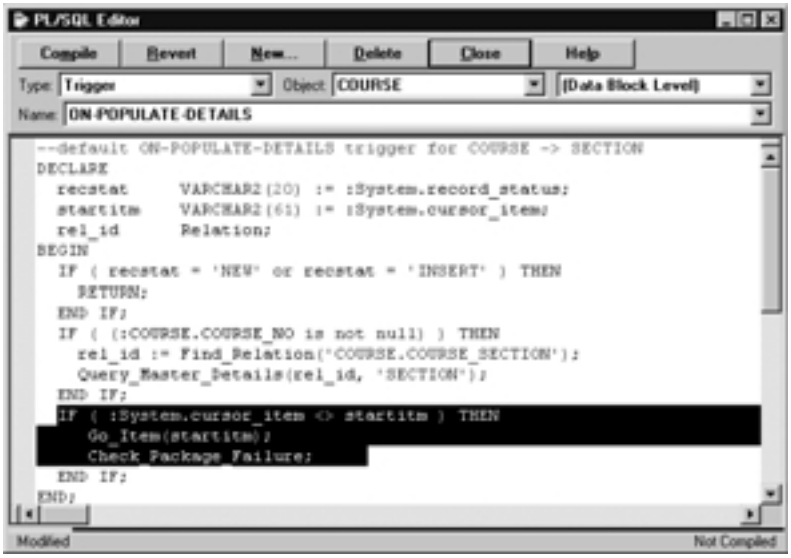

Figure 6.1 ■ The `ON-POPULATE-DETAILS` trigger displayed in the PL/SQL Editor.

Certain trigger types can be attached at the item level, the block level, or the form level. You can attach a trigger at higher levels of the Forms hierarchy to increase its scope.

■ *FOR EXAMPLE:*

In previous Exercises, you attached `WHEN-BUTTON-PRESSED` triggers to individual items. You created a button item called `EXIT` and attached a trigger directly to that item which fired code to exit the form. The trigger's scope was limited to the `EXIT` button. That is, the trigger only fired in response to the Button Pressed event of the `EXIT` item (button). It is possible to attach a `WHEN-BUTTON-PRESSED` trigger at the block or form level. What does this mean? Assume you have a `CONTROL` block with five buttons and you assign a `WHEN-BUTTON-PRESSED` trigger at the block level. When will the trigger fire? It will fire in response to a Button Pressed event for *any* of the buttons in the block. What does this mean? It means the scope of the trigger is now at the block level rather than only at the item level.

CATEGORIES OF TRIGGERS

The Forms help system categorizes triggers in two ways: by name and by functional category. Understanding the two methods of categorization will help you understand when and why certain triggers fire, which will

in turn help you decide which triggers to choose when you want to respond to Forms events.

CATEGORIZING TRIGGERS BY NAME

There are five named trigger categories. The first word in a trigger's name will tell how it will affect Forms default processing and when it will fire relative to Forms default processing.

The five named categories are as follows:

1) **When** event triggers, which augment Forms default processing.
2) **On** event triggers, which replace Forms default processing.
3) **Pre** event triggers, which fire just before a When event or an On event.
4) **Post** event triggers, which fire just after a When event or On event.
5) **Key** triggers, which fire when a user presses a certain key.

You would choose the appropriate trigger from one of these categories depending on what you want your own trigger code to do and how you want Forms to handle its own default processing.

■ *FOR EXAMPLE:*

Assume you wanted to write some code to respond to the Commit Transactions event which fires each time a form tries to insert a record. There are a number of insert-related triggers to choose from, including `ON-INSERT`, `PRE-INSERT`, and `POST-INSERT`. Do you want to replace Forms default insert processing and write all of the insert logic yourself? In that case, use an `ON-INSERT` trigger. Do you want to fire some of your own logic just before Forms executes its default insert processing? In that case, you would use a `PRE-INSERT` trigger. Or, you might want to use a `POST-INSERT` trigger to fire just after Forms has completed its default processing.

CATEGORIZING TRIGGERS BY FUNCTION

Triggers can also be categorized by the functions to which they are related. A `WHEN-BUTTON-PRESSED` trigger is an Interface Event trigger because it responds to the Button Pressed event, which, as its category name implies, is an interface event. `ON-INSERT` and `PRE-INSERT` triggers belong to the Transactional functional category because they are related to transactions and respond when there are transaction-related events. The Forms help system lists a number of functional trigger categories. In the Exercises in this Lab, and in the rest of the Labs in this Chapter, you will focus on the following functional categories:

1) **Query triggers**, which respond to events regarding queries.
2) **Validation triggers**, which respond to events regarding the validation of items and records.
3) **Transactional triggers**, which respond to events regarding inserting, updating, and committing of records.
4) **Key triggers**, which respond to Key Press events.

Each trigger falls into both a named and a functional trigger category.

■ *FOR EXAMPLE:*

The `ON-UPDATE` trigger falls into both the On event trigger named category and the Transactional functional trigger category.

LAB 6.1 EXERCISES

6.1.1 USE PL/SQL AND SQL IN TRIGGERS

Open the form `EX06_01.fmb` in the Form Builder. Open the PL/SQL Editor for the `STUDENT.ZIP` item's `WHEN-VALIDATE_ITEM` trigger.

a) What typical PL/SQL sections and constructs can you see here?

b) Look at the SQL statement that defines the `c_val_zip` cursor. How are the block and item expressed?

c) How would you write an SQL statement to select `DESCRIPTION` from the `COURSE` table into a `DESCRIPTION` display item in a `SECTION` block? The `DESCRIPTION` value you select should correspond to the `COURSE_NO` value that is currently in the form.

Look at the STUDENT.EXIT button's WHEN-BUTTON-PRESSED trigger.

d) Is this still PL/SQL? Why or why not?

6.1.2 UNDERSTAND TRIGGER SCOPE

Open form EX06_01.fmb in the Form Builder.

STUDENT.SALUTATION and STUDENT.ZIP both have WHEN-VALIDATE-ITEM triggers.

a) When an item needs to be validated, which of these triggers will fire? Will Forms simply fire both?

b) For EX06_01.fmb, at which levels within the Forms hierarchy are there WHEN-VALIDATE-ITEM triggers?

Run the form. Type Mr. into the SALUTATION item and then press the TAB key. Look at the status line for the Forms Runtime.

c) Which of the WHEN-VALIDATE-ITEM triggers fired? What does this tell you about the firing order of triggers?

Keep the form running, but go to the Form Builder.

d) Are there any WHEN-VALIDATE-ITEM triggers at the item level for FIRST_NAME and LAST_NAME?

Go back to the Forms Runtime. Take the following two actions and watch the status line after each. Type Joe into the FIRST_NAME item and press the TAB key. Type Smith into the LAST_NAME item and press the TAB key.

e) Which trigger fired? Why do you think this happened?

Exit and close the Forms Runtime and return to the Form Builder. View the properties for the SALUTATION item's WHEN-VALIDATE-ITEM trigger. Change the Execution Hierarchy property to Before. Run the form, type Mr. in the SALUTATION item and press the TAB key. Read the alert message, then click the OK button. Take note of the status line.

f) How has changing the Execution Hierarchy property affected the form?

Select the Triggers node under the STUDENT.SALUTATION item and click the Create button in the Object Navigator.

g) Can you create a PRE-FORM trigger here? Why not?

6.1.3 CATEGORIZE TRIGGERS

You will not need to open a specific form to complete this Exercise. However, you may want to have the Form Builder open in case you need to access the help system.

a) What trigger would you create to replace the default delete processing? What named category does this fall under? What functional category does it fall under?

b) Is it mandatory that you write triggers to respond to each event? What happens if you don't?

In the Exercises for Lab 6.2, you will create display items called `CITY` and `STATE` for a block based on the `STUDENT` table. You will also write a trigger to populate the `CITY` and `STATE` items with values that correspond to the value that has been fetched into the `ZIP` item.

c) When should the trigger you write populate these items? Before or after the query is issued?

d) Based on your answer to Question c, what trigger should you create? At what level should you attach it to the form? Search the help system if you are having trouble with this question.

Also in the Exercises for Lab 6.2, you will write another trigger to check that the value a user has entered into the ZIP item is valid in that it exists in the ZIPCODE table.

e) Which trigger should you create? What named and functional categories does this trigger belong to?

In the Exercises for Lab 6.3, you will write two triggers to set the values for the audit columns. These triggers will assign values to CREATED_BY, CREATED_DATE, MODIFIED_BY, and MODIFIED_DATE so that they can be inserted or updated to the database.

f) Should these triggers fire before or after the inserts and updates are issued?

g) Based on your answer to Question f, which triggers should you choose and what are their functional and named categories?

LAB 6.1 EXERCISES ANSWERS

6.1.1 ANSWERS

Open the form EX06_01.fmb in the Form Builder. Open the PL/SQL Editor for the STUDENT.ZIP item's WHEN-VALIDATE-ITEM trigger.

a) What typical PL/SQL sections and constructs can you see here?

Answer: There are Declare, Begin, *and* End *statements, a cursor, and conditional logic.*

The PL/SQL blocks you write in Forms triggers are identical in structure to the code you have written for Oracle database stored procedures. The WHEN-VALIDATE-ITEM trigger includes a DECLARE section for variables, cursors, and so on and a BEGIN statement that is followed by executable commands. Although there are none here, you can also include an EXCEPTION section in your Forms triggers for error handling.

FOR EXAMPLE:

The WHEN-VALIDATE-ITEM could have been written a little differently, in which case, it would have had to include an exception handler. Instead of using a cursor, a simple SQL statement could have been used to fetch rows from the database. Therefore, the trigger could no longer use the cursor attribute %NOTFOUND to detect invalid records. It would have to include an exception instead. The code would look like this:

```
BEGIN
    SELECT city, state
    INTO :STUDENT.CITY, :STUDENT.STATE
    FROM zipcode
    WHERE zip = :STUDENT.ZIP;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
    MESSAGE('Zipcode does not exist in Zipcode
table.');
    RAISE FORM_TRIGGER_FAILURE;
END;
```

This trigger is using the pre-defined NO_DATA_FOUND exception to handle instances in which no rows are returned for the SELECT…INTO statement. You are not restricted to pre-defined exceptions, however. You can create your own user-defined exceptions in Forms triggers just as you have in standard PL/SQL stored procedures.

b) Look at the SQL statement that defines the c_val_zip cursor. How are the block and item expressed?

Answer: The block and item are expressed as : STUDENT.ZIP.

Whenever you want to refer to an item and its block in an SQL statement in a trigger, you must express the reference using the following syntax:

```
:block.item
```

Note that this applies to the SELECT...INTO section of the SQL statement as well as the WHERE clause.

c) How would you write an SQL statement to select DESCRIPTION from the COURSE table into a DESCRIPTION display item in a SECTION block? The DESCRIPTION value you select should correspond to the COURSE_NO value that is currently in the form.

Answer: See description below.

```
SELECT description
INTO :SECTION.DESCRIPTION
FROM course
WHERE course.course_no = :SECTION.COURSE_NO.
```

Look at the STUDENT.EXIT button's WHEN-BUTTON-PRESSED trigger.

d) Is this still PL/SQL? Why or why not?

Answer: Yes it is.

The EXIT_FORM statement is a Forms built-in. Even though there are no BEGIN or END statements listed here, this is still PL/SQL. If there is nothing to declare in the DECLARE statement, then it is not mandatory that you include a BEGIN and an END statement. You can simply issue a series of PL/SQL executable commands. The STUDENT.EXIT button's WHEN-BUTTON-PRESSED trigger has only one line, which is a simple call to a Forms built-in. It is possible to have more involved PL/SQL triggers that still do not include a BEGIN or an END statement.

6.1.2 Answers

Have form EX06_01.fmb open in the Form Builder.

STUDENT.SALUTATION and STUDENT.ZIP both have WHEN-VALIDATE-ITEM triggers.

a) When an item needs to be validated, which of these triggers will fire? Will Forms simply fire both?

Answer: The trigger that is attached to the item that is being validated will fire.

The trigger that fires is determined by the scope of the event. The Validate Item event will occur at a specific item. Therefore, only that item's

WHEN-VALIDATE item trigger will fire. This will still be true even if other WHEN-VALIDATE-ITEM triggers are attached to other items in the form. This same rule applies to triggers attached at the block level.

b) For EX06_01.fmb, at which levels within the Forms hierarchy are there WHEN-VALIDATE-ITEM triggers?

Answer: There are WHEN-VALIDATE-ITEM *triggers at the item, block, and form levels.*

In Question a you learned that the same trigger can exist for multiple objects at the same level (i.e., WHEN-VALIDATE-ITEM triggers for each item in a block), but the only trigger that will fire is the one attached to the object. It is also possible to have the same trigger at different levels in the form. In form EX06_01.fmb, there are WHEN-VALIDATE-ITEM triggers at the item, block, and form levels. It is possible to have some or all of these triggers fire. You will explore how to control the firing order of triggers in the following questions.

Run the form. Type Mr. into the SALUTATION item and then press the TAB key. Look at the status line for the Forms Runtime.

c) Which of the WHEN-VALIDATE-ITEM triggers fired? What does this tell you about the firing order of triggers?

Answer: The item-level WHEN-VALIDATE-ITEM *has fired.*

This tells you that the default firing order of triggers is determined by the level of the object in the Forms hierarchy. The trigger that is attached to the object at the lowest level in the hierarchy will take precedence over triggers of the same name that are higher in the hierarchy. In this example, the SALUTATION item's WHEN-VALIDATE-ITEM trigger will take precedence and fire instead of the block-and-form level WHEN-VALIDATE-ITEM triggers.

Keep the form running, but go to the Form Builder.

d) Are there any WHEN-VALIDATE-ITEM triggers at the item level for FIRST_NAME and LAST_NAME?

Answer: No, there are no WHEN-VALIDATE-ITEM *triggers attached to these items.*

Go back to the Forms Runtime. Take the following two actions and watch the status line after each. Type Joe into the FIRST_NAME item and press the TAB key. Type Smith into the LAST_NAME item and press the TAB key.

e) Which trigger fired? Why do you think this happened?

Answer: The block-level `WHEN-VALIDATE-ITEM` *trigger fired.*

The block-level `WHEN-VALIDATE-ITEM` trigger fired in both of these cases because there was no `WHEN-VALIDATE-ITEM` trigger at the item level. The Validate Item event occurred, so Forms went searching for `WHEN-VALIDATE-ITEM` triggers. It found none at the item level, so it continued to the block level. It found one there and fired it. This can be very useful if there is logic that you'd like to execute for every item in the block.

Exit and close the Forms Runtime and return to the Form Builder. View the properties for the `SALUTATION` item's `WHEN-VALIDATE-ITEM` trigger. Change the `Execution Hierarchy` property to `Before`. Run the form. Type `Mr.` in the `SALUTATION` item and press the TAB key. Read the alert message, then click the `OK` button. Take note of the status line.

f) How has changing the `Execution Hierarchy` property affected the form?

Answer: Now both the item-level and block-level `WHEN-VALIDATE-ITEM` *triggers have fired.*

The default firing order has been changed so that the block-level trigger is now fired as well, instead of being ignored. The item-level trigger is fired first, then the block-level trigger. As you can tell, the `Execution Hierarchy` property lets you determine the order in which like triggers at different levels in the Forms hierarchy should be fired. By selecting `Before`, you indicated that you wanted the lowest level `WHEN-VALIDATE-ITEM` trigger to fire before any `WHEN-VALIDATE-ITEM` triggers at higher levels. You can imagine what would have happened if you had set `Execution Hierarchy` to `After`.

What would happen if you were to set the block-level `WHEN-VALIDATE-ITEM`'s `Execution Hierarchy` property to `Before`? When the Validate Item event occurred for `SALUTATION`, all three of this forms' `WHEN-VALIDATE-ITEM` triggers would fire in the following order: item, block, form.

Select the `Triggers` node under the `STUDENT.SALUTATION` item and click the `Create` button in the Object Navigator.

g) Can you create a `PRE-FORM` trigger here? Why not?

Answer: No you cannot.

A `PRE-FORM` trigger is form-specific and therefore cannot be defined at the item level. That is, not all trigger types can be defined at multiple levels in the Forms hierarchy.

go to contents

6.1.3 ANSWERS

a) What trigger would you create to replace the default delete processing? What named category does this fall under? What functional category does it fall under?

Answer: You would create the `ON-DELETE` *trigger. This is an On event trigger that falls into the Transactional category.*

Since the `ON-DELETE` trigger replaces the way Forms would normally delete a record, it is considered a Transactional trigger.

b) Is it mandatory that you write triggers to respond to each event? What happens if you don't?

Answer: No it is not. Forms default processing handles the event.

Form `EX06_01.fmb` has a number of triggers that respond to events like Validate Item, Button Pressed, and so on. But, think back to the forms you created in earlier Chapters. You did not write any triggers to respond to the Validate Item events, yet the events still occurred when you changed an item's value and then navigated to another item. Forms looked for triggers to respond to the events, but when it found none, it simply executed the default processing.

c) When should the trigger you write populate these items? Before or after the query is issued?

Answer: The trigger should populate these items after the query is issued and the results have been returned to the form.

d) Based on your answer to Question c, what trigger should you create? At what level should you attach it to the form? Search the help system if you are having trouble with this question.

Answer: You should create a `POST-QUERY` *trigger and attach it at the block level.*

`POST-QUERY` is a Post event trigger that belongs to the Query group of triggers.

As its name implies, the `POST-QUERY` trigger will fire each time a record is returned to the block. `POST-QUERY` triggers can be attached at the block or form level, but not at the item level. If you attach a `POST-QUERY` trigger to a `STUDENT` block, for example, it will only fire when records are fetched to the `STUDENT` block. If you attach a `POST-QUERY` trigger at the form level of a form that has a `STUDENT` and an `ENROLLMENT` block, it will fire whenever a record is fetched into either block.

go to contents

e) Which trigger should you create? What named and functional categories does this trigger belong to?

Answer: You should create a `WHEN-VALIDATE-ITEM` *trigger. This is a When event trigger that belongs to the Validation functional category.*

f) Should these triggers fire before or after the inserts and updates are issued?

Answer: These triggers should fire before the inserts and updates are issued.

g) Based on your answer to Question f, which triggers should you choose and what are their functional and named categories?

Answer: You should choose `PRE-INSERT` *and* `PRE-UPDATE` *triggers. These are Pre event triggers that belong to the Transactional functional category.*

LAB 6.1 SELF-REVIEW QUESTIONS

In order to test your progress, you should be able to answer the following questions:

1) What is the scope within which triggers fire?
- **a)** ____ The object they are attached to
- **b)** ____ The PL/SQL block
- **c)** ____ The user's session
- **d)** ____ The PL/SQL Editor

2) When will a block-level `WHEN-BUTTON-PRESSED` trigger fire?
- **a)** ____ In response to Button Pressed events for buttons belonging to the block
- **b)** ____ For every button in the form without an item-level `WHEN-BUTTON-PRESSED` trigger
- **c)** ____ For every button in the block that has a `WHEN-BUTTON-PRESSED` trigger with `Execution Hierarchy` set to `Override`
- **d)** ____ All of the above

3) Where can a `WHEN-NEW-ITEM-INSTANCE` trigger be defined?
- **a)** ____ At the form level
- **b)** ____ At the block level
- **c)** ____ At the item level
- **d)** ____ All of the above
- **e)** ____ Only b & c

4) Which of the following are true about a `POST-TEXT-ITEM` trigger?
- **a)** ____ It is a When event trigger
- **b)** ____ It is a navigational trigger
- **c)** ____ It will replace Forms default processing
- **d)** ____ None of the above

go to contents

5) Which of the following is true about When event triggers?
- **a)** ____ They augment default Forms processing
- **b)** ____ They replace default Forms processing
- **c)** ____ They only respond to interface events
- **d)** ____ None of the above

6) Which of the following is true about the `ON-ERROR` trigger?
- **a)** ____ It fires when you compile code that has errors
- **b)** ____ It replaces default Forms processing
- **c)** ____ It is created when the relation object is created
- **d)** ____ It rolls back the form when errors occur

7) What will happen if an item-level `WHEN-NEW-ITEM-INSTANCE` trigger's `Execution Hierarchy` property is set to `Override`?
- **a)** ____ All other `WHEN-NEW-ITEM-INSTANCE` triggers attached to other items in the block will be overriden
- **b)** ____ All other `WHEN-NEW-ITEM-INSTANCE` triggers at higher levels in the Forms hierarchy will be overridden
- **c)** ____ New items will be created in the block to override the old ones
- **d)** ____ All of the above

8) Which of the following cannot be done with a `WHEN-NEW-FORMS-INSTANCE` trigger?
- **a)** ____ It cannot be created in the same form as a `PRE-FORM` trigger
- **b)** ____ It cannot be created at the item level
- **c)** ____ You cannot use the `SET_ITEM_PROPERTY` in it
- **d)** ____ You cannot use it to set block properties

Quiz answers appear in Appendix A, Section 6.1.

LAB 6.2

CREATING TRIGGERS OF VARIOUS TYPES

LAB OBJECTIVES

After this Lab, you will be able to:

- Create Query Triggers
- Create Validation Triggers
- Create Transactional Triggers
- Create Key Triggers

There are hundreds of triggers in Forms, and multiple ways to use each trigger. In this Lab, you will learn to write some commonly used triggers. The code you write will be specific to the objects in the `STUDENT` application, but can serve as templates for triggers you write in your own applications.

QUERY TRIGGERS

The `POST-QUERY` is often used to populate non-base table display items in a block. These non-base table display items are sometimes referred to as "lookup" items, and are used to make one or more of the base-table items more meaningful.

■ *FOR EXAMPLE:*

Assume you have created a block based on the `SECTION` table. You are displaying all of its columns, including `COURSE_NO`. To make each record more meaningful, you'd like to display the course's description as well. However, the `DESCRIPTION` column resides in the `COURSE` table, so you can't include it as a base-table item in the `SECTION` block. You must,

therefore, include it as a display item and populate it with a POST-QUERY trigger. To do this, you would create a display item, name it DESCRIPTION (or whatever you'd like), and use a POST-QUERY trigger to fetch records into the DESCRIPTION item.

The form would fetch a section record into the block and then fire the POST-QUERY trigger. The trigger would then fetch the corresponding course.description from the database and place it into the DESCRIPTION display item.

VALIDATION TRIGGERS

There are two Validation triggers that are commonly used in Forms: WHEN-VALIDATE-ITEM and WHEN-VALIDATE-RECORD. Each serves to validate data entered by a user. In the Exercises, you will write some simple Validation triggers to confirm that:

1) Values entered by a user adhere to the business rules.
2) Values entered into foreign-key items exist in the parent table.

■ *FOR EXAMPLE:*

Assume there is a business rule in the STUDENT application that states that no class can cost more than $5,000. Whenever a user enters a value into a COST item, you want the form to confirm that the value they've entered adheres to the rule. You could do so by writing a WHEN-VALIDATE-ITEM trigger that contains the following code and attach it to the COST item:

```
IF :SECTION.COST > 5000 THEN
    MESSAGE('Course costs must be less than
$5,000.');
    RAISE FORM_TRIGGER_FAILURE;
END IF;
```

The WHEN-VALIDATE-ITEM trigger will fire when both of the following two conditions have been met:

1) The user has changed the value in the item.
2) The user has navigated out of the item.

If the user enters a value greater than 5000 and navigates out of the item, the Validate Item event will occur and the WHEN-VALIDATE-ITEM trigger

will fire. Since validation has failed, the user will receive a message and processing will stop.

Validation triggers can also be used to check that values entered into foreign-key items exist in the parent table.

■ *FOR EXAMPLE:*

Assume you have a form based on the `ENROLLMENT` table. You want the application to confirm that the value entered for `SECTION_ID` exists in the `SECTION` table. If it doesn't, the `INSERT` or `UPDATE` statement will be rejected by the database. If the database is going to reject it anyway, which is essentially validation, then why repeat the code in the application? For one thing, it makes the application a bit more user-friendly. The user will be alerted to his mistake immediately rather than later at the time of the insert. It also makes it easier to process the error. You respond to and handle validation item-by-item rather than by trying to process the error message returned by the database, which might not always be meaningful.

TRANSACTIONAL TRIGGERS

There are a number of Transactional triggers used to augment or replace Forms default transaction processing. The `ON-POPULATE-DETAILS` and `ON-CHECK-DELETE-MASTER` master-detail triggers are considered Transactional triggers. `PRE-CHANGE`, `POST-FORMS-COMMIT`, `POST-DATABASE-COMMIT`, and many other transaction-related triggers allow you to write your own processing logic in and around Forms-level and database-level transactions. In this Lab, you will experiment with two: the `PRE-INSERT` and `PRE-UPDATE` triggers. You will use these to set values for the audit columns.

KEY TRIGGERS

Key triggers fire whenever a user presses a corresponding key on the keyboard. If a user presses the down arrow, or the down key, then the `KEY-DOWN` trigger will fire. Key triggers can be used if you want to change or replace default key processing.

LAB 6.2 EXERCISES

6.2.1 CREATE QUERY TRIGGERS

Use the wizards to quickly create a form based on the `STUDENT` table. `Enforce data integrity` on the wizard's table page should be **unchecked.** Include the audit columns in the block, but do not display them

on the canvas. Lay the items out in Form style. Create two display items in the STUDENT block and name them CITY and STATE. Position them after STUDENT.ZIP in the block and just to the right of STUDENT.ZIP on the canvas. Size and align them so that they are arranged neatly, but do not be overly concerned with the look of the form.

Use the code below to answer Questions a–d.

```
DECLARE
    CURSOR c_city_state IS SELECT city, state
           FROM zipcode
           WHERE zip = :STUDENT.ZIP;
BEGIN
    OPEN c_city_state;
    FETCH c_city_state INTO :STUDENT.CITY,
:STUDENT.STATE;
    CLOSE c_city_state;
END;
```

a) What two database columns is this trigger querying?

b) How will the POST-QUERY trigger know which record to fetch from the database?

c) Which items are being populated? Which line of code populates these items?

d) Which object should you attach the POST-QUERY trigger to?

Create a POST-QUERY trigger and attach it to the object that was your answer for Question d. Type the code above into the PL/SQL Editor and click the Compile button.

e) Were there any errors?

When the POST-QUERY trigger compiles correctly, run the form and issue a query.

f) Were the CITY and STATE items populated?

If not, look at the Forms Runtime's status line for error messages. Select Help | Display Errors from the Forms Runtime's Main Menu to see more details.

g) What was the error? Why did this happen? What should you do to the display items to correct this?

Exit the form and fix the mistake. Run the form again and issue a query to test the POST-QUERY trigger.

h) Did it populate CITY and STATE this time? What happens when you scroll from record to record?

i) If you were to create a new form based on the ENROLLMENT table, what are some display items you could create and populate with a POST-QUERY trigger?

j) What would the code for the trigger be?

Save the form as `R_POSTQ_VAL.fmb`.

6.2.2 CREATE VALIDATION TRIGGERS

In the following Exercise questions, you will write a Validation trigger for `STUDENT.ZIP` in the `R_POST_VAL.fmb` form. The trigger will validate that the `ZIP` value a user wishes to insert or update exists in the `Zipcode` table. Use the code below to answer Questions a–e.

```
DECLARE
    v_invalid BOOLEAN;
    CURSOR c_val_zip IS SELECT city, state
    FROM zipcode
    WHERE zip = :STUDENT.ZIP;
BEGIN
    OPEN c_val_zip;
    FETCH c_val_zip INTO :STUDENT.CITY, :STUDENT.STATE;
    v_invalid := c_val_zip%NOTFOUND;
    IF v_invalid THEN
          MESSAGE('This zipcode is invalid. Re-enter
another.');
          RAISE FORM_TRIGGER_FAILURE;
    END IF;
END;
```

a) What variable are you declaring to help check the validity of the `ZIPCODE` value? What is its data type?

b) Which line of code assigns a value to this variable? What cursor attribute are you using to assign the value?

c) What will be the value of the `v_invalid` variable if the cursor fails to fetch a row? What will this mean about the value the user has entered?

d) What two commands will the trigger issue if the value is invalid?

e) Why is the trigger fetching values into the `CITY` and `STATE` columns if the purpose is to validate the `ZIP` item? Won't the columns be populated by the `POST-QUERY` trigger?

Create a `WHEN-VALIDATE-ITEM` trigger for the `STUDENT.ZIP` item and enter the code above. Compile the trigger. Run the form and issue a query. Change the value in the `ZIP` item to `123` and press the TAB key.

f) Has the `WHEN-VALIDATE-ITEM` trigger fired? What two things about the form's behavior indicate that it has?

g) What Forms object could you attach to this item to help the user choose a valid Zip Code?

Save the changes to form R_POSTQ_VAL.fmb.

LAB 6.2

Use the wizards to quickly create a form based on the SECTION table. Enforce data integrity on the wizard's table page should be **unchecked.** Include the audit columns in the block, but do not display them on the canvas. Lay the items out in Form style.

Assume that there is a building called L5 on the STUDENT campus. The rooms in this building can only seat 15 students or less. In the SECTION table's LOCATION column, all of the rooms in the L5 building are named L501, L502, and so on. When users are inserting or updating section records, you want to prevent them from making the CAPACITY greater than 15 for any room in the L5 building.

h) Could you write a Validation trigger to enforce this rule? What would the code be? **Write your answer on paper first.** The trigger code should not be overcomplicated. You should be able to do it in three simple statements.

i) If you use a WHEN-VALIDATE-ITEM trigger, which item could the trigger code be attached to? Do not create the trigger, simply write down your answer.

j) If your answer to Question i was CAPACITY, when will the WHEN-VALIDATE-ITEM trigger fire?

k) If the user inserted a new row, what would happen if the user set CAPACITY to 25 first and then set LOCATION to L501? Would the validation take place? Why not?

l) Which Validation trigger could you create to make sure that the trigger fires for each record? Which object should you attach it to? Create the trigger, enter the code, and test the form.

You do not need to save this form as you will not need it in future Exercises.

6.2.3 CREATE TRANSACTIONAL TRIGGERS

Use the wizards to quickly create a form based on the `COURSE` table. Leave `Enforce data integrity` **unchecked.** Include the audit columns in the block **and** on the canvas. Normally you would not include the audit columns on the canvas. You are doing it here so you can see the outcome of the trigger code. Give the canvas a `Form`-style layout and set the audit columns to be display items.

Set the `COURSE_NO` initial value property to:

```
:SEQUENCE.COURSE_NO_SEQ.NEXTVAL
```

In this Exercise, you will write Transactional triggers to set the values for the audit columns.

a) Why do you have to write a trigger to set these values? Why not make the user input these values?

b) Should this trigger be assigned to the form or block level?

c) What two pieces of information will you need to get from the system to assign values for the audit columns?

The code for the PRE-INSERT trigger will be as follows:

```
DECLARE
    v_block         VARCHAR2(30);
    v_username      VARCHAR2(30);
    v_date          DATE;
BEGIN
  v_username := GET_APPLICATION_PROPERTY(USERNAME);
  v_date := SYSDATE;
  v_block := :SYSTEM.CURSOR_BLOCK;

  COPY(v_date, v_block||'.CREATED_DATE');
  COPY(v_username,  v_block||'.CREATED_BY');
  COPY(v_date, v_block||'.MODIFIED_DATE');
  COPY(v_username,  v_block||'.MODIFIED_BY');
END;
```

d) Which built-in is being used to get the user's name?

e) How is the value of v_block assigned?

f) What parameters are being passed to the COPY built-in?

Create the PRE-INSERT trigger at the form level and enter the code above. Run the form and try to insert a new record.

g) Did the trigger work? How do you know?

h) What trigger should you create to set `MODIFIED_BY` and `MODIFIED_DATE` every time a record is changed?

i) What will the code be for this trigger?

j) Could you reuse these triggers exactly as they are for forms with `SECTION` blocks? `STUDENT` blocks? Any block?

Save this form as `R_TRANS.fmb`.

6.2.4 Create Key Triggers

In this Exercise, you will create a simple Key trigger for response when the user clicks the `Execute Query` button on the keyboard.

You will also explore a Key trigger that is written by the Form Builder when you select `Enforce data integrity` in the Data Block Wizard. This `KEY-DELREC` trigger is written whenever primary-foreign key constraints exist in the database that correspond to one of the items in the block.

Open `EX06_02.fmb` in the Form Builder. Use the Object Navigator to create a form-level `KEY-EXEQRY` trigger. Add the following statement to the trigger:

```
MESSAGE('You have pressed the F8 key to execute a query');
```

Run the form and press the F8 key on the keyboard to test the trigger. Exit the form after you have tested the trigger to return to the Form Builder.

a) Did the Key trigger respond when the key was pressed? Why didn't the form execute a query?

Add the EXECUTE_QUERY statement to the end of the KEY-EXEQRY trigger. The trigger should now be as follows:

```
MESSAGE('You have pressed the F8 key to execute a query');
EXECUTE_QUERY;
```

Create a WHEN-BUTTON-PRESSED trigger for the ZIPCODE.EXECUTE_QUERY button. Add the following statement:

```
EXECUTE_QUERY;
```

Run the form and press the F8 key on the keyboard to confirm that the query has been executed. Now click the Execute Query button.

b) Was the message text issued along with the query? Why not?

Change the code in the WHEN-BUTTON-PRESSED trigger to the following:

```
DO_KEY('EXECUTE_QUERY');
```

Run the form and test the Execute Query button.

c) What function did the DO_KEY built-in provide?

Study the code for the ZIPCODE block's KEY-DELREC trigger.

go to contents

d) What function will this trigger perform?

Run the form. Click the `Enter Query` button on the toolbar to put the form into `Enter_Query` mode. Issue a query for the Zip Code 06605. Click the `Remove Record` button on the toolbar.

e) Did the `KEY-DELREC` trigger fire? What built-in must the trigger associated with the `Remove Record` button use to make this happen?

LAB 6.2 ANSWERS

6.2.1 ANSWERS

a) What two database columns is this trigger querying?

Answer: It is querying the `CITY` *and* `STATE` *columns in the* `ZIPCODE` *table.*

The cursor `c_city_state` defines the query, which will fetch the values to populate the display items `CITY` and `STATE`. In this case, the SQL statement is rather simple. It is merely selecting two columns from the same table. Statements can be much more complicated in that they can include more columns, joins, complicated `WHERE` clauses, and so on.

b) How will the `POST-QUERY` trigger know which record to fetch from the database?

Answer: The `WHERE` *clause indicates that the* `ZIP` *in the* `ZIPCODE` *table should correspond with the* `ZIP` *item in the* `STUDENT` *block.*

Note that the item is expressed as: `block.item`, which is the same syntax you learned in Lab 6.1.

c) Which items are being populated? Which line of code populates these items?

Answer: The `CITY` *and* `STATE` *items in the* `STUDENT` *block are being populated. They are populated with the following line of code:*

```
FETCH c_city_state INTO :STUDENT.CITY, :STUDENT.STATE;
```

go to contents

The cursor fetches the records directly into the items CITY and STATE. Instead of using a cursor, the POST-QUERY trigger can also be written using a SELECT. . . INTO statement like the following:

```
SELECT city, state
INTO :STUDENT.CITY, :STUDENT.STATE
FROM zipcode
WHERE zipcode = :STUDENT.ZIP;
```

d) Which object should you attach the POST-QUERY trigger to?

Answer: You should attach the POST-QUERY *trigger to the* STUDENT *block.*

In this case, the POST-QUERY trigger applies only to the STUDENT block, so it should be attached to the STUDENT block. It is possible to attach POST-QUERY triggers at the form level, but that only makes sense if you want the trigger to apply to all of the blocks in the form.

e) Were there any errors?

If there were some errors, perhaps you made a small typo. Compare your code with that in Figure 6.2. Also, take a moment to study the buttons in the PL/SQL Editor.

Figure 6.2 ■ The PL/SQL Editor showing a successfully compiled POST-QUERY trigger.

You have already worked with the PL/SQL Editor in previous Chapters, and have probably found that it is a rather simple, yet handy, tool for writing code. Now that your triggers are becoming more complicated, it is worth a bit more exploration.

THE PL/SQL EDITOR

Refer to Figure 6.2. The buttons across the top of the PL/SQL Editor's window are self-explanatory and do not require discussion. Just below the buttons are three list items: `Type`, `Object`, and one with no label. As you can see, these indicate the type of PL/SQL object that you are creating, along with the object to which you are attaching it. In this case, you are working with a trigger that is attached to the `STUDENT` block at the block level. The `Name` list item below indicates which trigger you are working with.

The PL/SQL Editor can also help you debug your code.

■ *FOR EXAMPLE:*

If there had been a mistake in your code, the PL/SQL Editor may have looked like Figure 6.3.

Note that the gray area below the trigger code lists error messages. In this case, the `ZIPCODE` table was misspelled, so the Form Builder could not find it in the database.

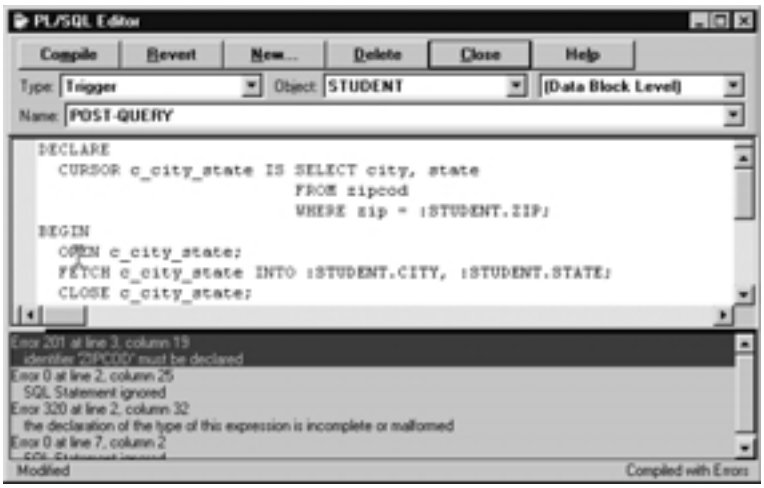

Figure 6.3 ■ The PL/SQL Editor showing errors in a trigger.

You will use the PL/SQL Editor in this Chapter to write triggers and also in later Chapters to write PL/SQL program units.

When the POST-QUERY trigger compiles correctly, run the form and issue a query.

f) Were the CITY and STATE items populated?

Answer: No they were not.

g) What was the error? Why did this happen? What should you do to the display items to correct this?

Answer: The error was FRM-40505 Unable to Perform Query.

If you looked at the Show Errors window, you would have been able to see the error in the SELECT statement. CITY and STATE are not base-table items in this block and should not be included in the query. They are non-base-table items, so their Database Item properties should have been set to No.

It is important to remember that, although CITY and STATE are being populated by values from the database, they are not base-table items in this block.

Run the form and issue a query to test the POST-QUERY trigger.

h) Did it populate CITY and STATE this time? What happens when you scroll from record to record?

Answer: Yes, the CITY *and* STATE *values change to correspond with the value in the* ZIP *item.*

i) If you were to create a new form based on the ENROLLMENT table, what are some display items you could create and populate with a POST-QUERY trigger?

Answer: You could create display items to show the student's LAST_NAME *and* FIRST_NAME *and perhaps the* COURSE_NO.

In these examples, you are using the POST-QUERY trigger to provide lookup values to make the form data more meaningful to the user. In the STUDENT form created above, the ZIP item was much more meaningful to the user when the CITY and STATE values were supplied along with it.

In the case of a form based on the ENROLLMENT table, what values from other tables might make the enrollment information more meaningful? The LAST_NAME and FIRST_NAME would make STUDENT_ID more meaningful. And perhaps the COURSE_NO would make the SECTION_ID more

meaningful. You could include even more, like the DESCRIPTION of the course, the INSTRUCTOR_ID, and so on. In the following question, write a trigger that would populate LAST_NAME, FIRST_NAME, and COURSE_NO.

j) What would the code for the trigger be?

Answer: See below.


```
DECLARE
    CURSOR c_student_name is
          SELECT first_name, last_name
          FROM student
          WHERE student_id =
:ENROLLMENT.STUDENT_ID;
    CURSOR c_course_no is
          SELECT course_no
          FROM section
          WHERE section_id =
:ENROLLMENT.SECTION_ID;
BEGIN
    OPEN c_student_name;
    FETCH c_student_name INTO :ENROLLMENT.LAST_NAME,
:ENROLLMENT.FIRST_NAME;
    CLOSE c_student_name;
    OPEN c_course_no;
    FETCH c_course_no INTO :ENROLLMENT.COURSE_NO;
    CLOSE c_course_no;
END;
```

Note that, in this case, the trigger had more than one cursor.

Save the form as R_POSTQ_VAL.fmb.

6.2.2 Answers

a) What variable are you declaring to help check the validity of the ZIPCODE value? What is its data type?

Answer: The variable is v_invalid *and its data type is* Boolean.

b) Which line of code assigns a value to this variable? What cursor attribute are you using to assign the value?

Answer: See below.

go to contents

The line of code is:

```
V_invalid := c_val_zip%NOTFOUND;
```


The cursor attribute is %NOTFOUND. As you know from your experience with PL/SQL, %NOTFOUND evaluates to TRUE if a cursor does not fetch a record from the database; it evaluates to FALSE if the cursor successfully fetches a record.

Here is where the actual validation in the WHEN-VALIDATE-ITEM trigger occurs. The cursor opens and begins trying to fetch rows. If it is unable to fetch a row that matches the criteria in the WHERE clause, it sets the %NOTFOUND attribute to TRUE.

c) What will be the value of the v_invalid variable if the cursor fails to fetch a row? What will this mean about the value the user has entered?

Answer: The value of v_invalid *will be* TRUE.

d) What two commands will the trigger issue if the value is invalid?

Answer: The trigger will issue a message to the user and RAISE FORM_TRIGGER_FAILURE.

FORM_TRIGGER FAILURE is a pre-defined, built-in Forms exception. It is used to halt Forms processing when an error has occurred. In this case, the user has entered an invalid value in the ZIP item. After receiving the message, the FORM_TRIGGER_FAILURE built-in exception will not allow the user to continue until they enter a valid Zip Code.

FORM_TRIGGER_FAILURE is not limited to WHEN-VALIDATE-ITEM triggers. It can be used in any Forms PL/SQL object. However, it cannot be used in PL/SQL objects that are stored in the database.

e) Why is the trigger fetching values into the CITY and STATE columns if the purpose is to validate the ZIP item? Won't the columns be populated by the POST-QUERY trigger?

Answer: See discussion below.

By fetching CITY and STATE into the form as you validate ZIP, you are killing two birds with one stone. As the user creates new records or edits existing ones, you will want to validate the ZIP value. You also want the CITY and STATE display items to change so that they correspond to the new ZIP value. The POST-QUERY trigger will not fire to populate these

items while the user is editing items and tabbing around the form because there has not been a query.

f) Has the WHEN-VALIDATE-ITEM trigger fired? What two things about the form's behavior indicate that it has?

Answer: The message appeared on the hint line, and it is not possible to navigate from the ZIP *item.*

g) What Forms object could you attach to this item to help the user choose a valid Zip Code?

Answer: An LOV.

If the user is having trouble entering valid values, it might be helpful to provide an LOV for them to select from.

h) Could you write a Validation trigger to enforce this rule? What would the code be? The trigger code should not be overcomplicated. You should be able to do it in three simple statements.

Answer: See the code below.

```
IF :SECTION.LOCATION LIKE 'L5%'
    AND :SECTION.CAPACITY > 15
    THEN MESSAGE('Capacity must be less than 15 for
sections in the L5 building.');
    RAISE FORM_TRIGGER_FAILURE;
END IF;
```

Note that the trigger will check two of the items in the form and if both conditions are met, it will issue a message to the user and RAISE FORM_TRIGGER_FAILURE to halt processing.

i) If you use a WHEN-VALIDATE-ITEM trigger, which item should the trigger code be attached to?

Answer: It could be attached to the CAPACITY *or* LOCATION *item.*

j) If your answer to Question i was CAPACITY, when will the WHEN-VALIDATE-ITEM trigger fire?

Answer: It will fire after you change the CAPACITY *value and then navigate out of the item.*

k) If the user inserted a new row, what would happen if the user set CAPACITY to 25 first and then set LOCATION to L501? Would the validation take place? Why not?

Answer: No.

Because the trigger is attached to the CAPACITY column, it will fire only after the user has changed CAPACITY and navigated out of it. It will not fire when the user changes and navigates out of LOCATION. Therefore, if the user changes the CAPACITY item first, then changes the LOCATION item second, the trigger will not fire. This goes back to trigger scope; a trigger will fire only within its scope, which in this case is within the validation of CAPACITY.

So, what could you do? You could copy the trigger and attach it to both items, but that would not be very elegant. You could put the WHEN-VALIDATE-ITEM trigger at the block level, but then it would fire for every item, which would be inelegant and inefficient. Or, you could put a different trigger at the block level. Read the next question for more details on what you could do.

l) Which Validation trigger could you create to make sure that the trigger fires for each record? Which object should you attach it to? Create the trigger, enter the code, and test the form.

Answer: You could create a WHEN-VALIDATE-RECORD *trigger and attach it to the block.*

The WHEN-VALIDATE-RECORD trigger will fire once for the entire record when the Validate Record event occurs. So, no matter what order a user enters or changes the values in the block, the WHEN-VALIDATE-RECORD trigger will fire and catch any invalid values.

6.2.3 ANSWERS

a) Why do you have to write a trigger to set these values? Why not make the user input these values?

Answer: These are audit columns and should be maintained by the system.

The purpose of these columns is to keep a strict record of when, and by whom, each record was updated or changed. If the audit trail is being kept for security reasons, as well as for record-keeping reasons, then it does not make sense to allow the user to edit the values.

The most secure method would be to populate these columns with database triggers. But, for the purpose of these Exercises, you will have the form populate them.

b) Should this trigger be assigned to the form or block level?

Answer: `PRE-INSERT` *triggers can be set at either the form or block level.*

The code you will write in this Exercise will be block-independent. That is, the block names will not be hard-coded into the trigger so that they can apply to any base-table block in the form. Therefore, if you put the triggers at the form level, their scope will be for all base-table blocks.

In this application, all of the base-table blocks will contain the audit items.

c) What two pieces of information will you need to get from the system to assign values for the audit columns?

Answer: You will need to get the user's name and the date.

d) Which built-in is being used to get the user's name?

Answer: The `GET_APPLICATION_PROPERTY` *built-in.*

e) How is the value of `v_block` assigned?

Answer: `v_block` *is assigned using the* `:SYSTEM.CURSOR_BLOCK` *system variable.*

System variables hold internal information about the form. `SYSTEM.CURSOR_BLOCK` holds the value of the current navigation block. There are many other system variables that you can reference to get all sorts of internal information like the name of the current item, if the form is in Enter Query mode or Normal mode, the current position of the mouse, and so on.

In this case, you want to get the name of the current block so that you can set the values for the audit items appropriately.

f) What parameters are being passed to the `COPY` built-in?

Answer: A value and a `block.item` *name.*

The `COPY` built-in takes a value and copies it somewhere else. Here, you are copying the value in the variable into one of the audit items. You could also use the `COPY` built-in to copy the value in one variable into another variable.

go to contents

The value for each of the audit items could have been set without the COPY built-in using the following syntax:

```
:COURSE.CREATED_BY := v_username;
```


While this method would work, it is not block-independent since you had to hard-code the block name into the statement.

g) Did the trigger work? How do you know?

Answer: Yes, a `Transaction complete one record applied and saved` *message appeared in the hint line.*

Also note that the values appeared in the items in the form. This, however, is not an indication that the insert succeeded in the database. This merely indicates that the values were successfully populated in the items. The PRE-INSERT trigger fires before an insert. The code you have written in the PRE-INSERT only assigns values to items in the form; it does nothing to make sure that those values are inserted to the database. Once the PRE-INSERT trigger has completed, Forms continues with its default insert processing. Forms writes an INSERT statement that includes every data item in the block and sends it off to the database.

h) What trigger should you create to set MODIFIED_BY and MODIFIED_DATE every time a record is changed?

Answer: You should use a PRE-UPDATE *trigger.*

i) What will the code be for this trigger?

Answer: See the code below.

```
DECLARE
    v_block         VARCHAR2(30);
    v_username      VARCHAR2(30);
    v_date          DATE;
BEGIN
  v_username := GET_APPLICATION_PROPERTY(USERNAME);
  v_date := SYSDATE;
  v_block := :SYSTEM.CURSOR_BLOCK;
  COPY(v_date, v_block||'.MODIFIED_DATE');
  COPY(v_username,  v_block||'.MODIFIED_BY');
END;
```

Note that the only difference is that the CREATED_BY and CREATED_DATE items are not being populated here.

j) Could you reuse these triggers exactly as they are for forms with SECTION blocks? STUDENT blocks? Any block?

Answer: Yes you could.

Since you didn't hard-code the block names into the triggers, you have made them portable across forms.

Save this form as R_TRANS.fmb.

6.2.4 ANSWERS

a) Did the Key trigger respond when the key was pressed? Why didn't the form execute a query?

Answer: Yes, the trigger responded.

The MESSAGE statement that was written to the trigger was successfully executed. However, the form didn't execute a query. Key triggers, like On triggers, replace Forms default processing, so the original default processing will not occur. To augment the default processing of a keystroke, you must remember to manually enter the necessary code.

■ *FOR EXAMPLE:*

For the KEY-EXEQRY trigger to reproduce the default processing, the code must be as follows:

```
MESSAGE('You have pressed the F8 key to execute a query');
EXECUTE_QUERY;
```

b) Was the message text issued along with the query? Why not?

Answer: No it was not.

The WHEN-BUTTON-PRESSED trigger is not aware of the code that is in the KEY-EXEQRY trigger, so of course it will not fire the MESSAGE statement. The problem here is that the application will behave differently if the user executes a query by pressing the F8 key on the keyboard, or if they click the Execute Query button on the screen. In almost all cases, you

will want the behavior to be the same no matter how the user chooses to issue a query. This is not only true for executing queries, but for all instances when you decide to use Key triggers. Question c will help you find a solution to this problem.

Change the code in the WHEN-BUTTON-PRESSED trigger to the following:

```
DO_KEY('EXECUTE_QUERY');
```

Run the form and test the Execute Query button.

c) What function did the DO_KEY built-in provide?

Answer: The DO_KEY built-in fired the KEY-EXEQRY trigger.

When executed, DO_KEY fires the Key trigger associated with the built-in it has accepted as its parameter.

■ *FOR EXAMPLE:*

If you issue the statement

```
DO_KEY('COMMIT_FORM');
```

the KEY-COMMIT trigger will fire.

If you issue the statement

```
DO_KEY('ENTER_QUERY');
```

the KEY-ENTQRY trigger will fire.

Study the code for the ZIPCODE block's KEY-DELREC trigger.

d) What function will this trigger perform?

Answer: See the discussion below.

The form will prevent the current record from being marked for deletion if that record has child records in another table.

Run the form. Click the Enter Query button on the toolbar to put the form into Enter Query mode. Issue a query for the Zip Code 06605. Click the Remove Record button on the toolbar.

e) Did the KEY-DELREC trigger fire? What built-in must the trigger associated with the Remove Record button use to make this happen?

Answer: Yes, the DO_KEY *built-in was used.*

This example illustrates the usefulness of the DO_KEY built-in. The default processing for the deletion of a record has been overwritten and replaced with a KEY-DELREC trigger. By using the DO_KEY built-in behind the toolbar buttons, the application is ensuring that any Key trigger logic will be fired.

You will use the DO_KEY built-in again when you create your own toolbar in Chapter 8, "Canvases and Windows."

LAB 6.2 SELF-REVIEW QUESTIONS

In order to test your progress, you should be able to answer the following questions:

1) Which of the following is true about POST-QUERY triggers?
- **a)** ____ They are not valid at the form level
- **b)** ____ They fire after a record has been fetched
- **c)** ____ You can attach them to record groups
- **d)** ____ a & b

2) Where could you attach a POST-QUERY trigger if you want it to populate a display item named STUDENT.LAST_NAME?
- **a)** ____ To the primary key item in the block
- **b)** ____ To the STUDENT block
- **c)** ____ To the LAST_NAME item
- **d)** ____ To any item in the form that will be queried

3) Which of the following is true about the PL/SQL Editor?
- **a)** ____ You can use it to write triggers and program units
- **b)** ____ It will check the syntax of your code
- **c)** ____ It will indent your code automatically
- **d)** ____ All of the above

4) When does the WHEN-VALIDATE-ITEM trigger fire?
- **a)** ____ In response to a Validate Item event
- **b)** ____ When an item is not valid
- **c)** ____ When the user navigates out of an item and that item's value has been changed
- **d)** ____ a & c

5) Which trigger should you use to validate an entire record?

a) ____ `POST-VALIDATE-RECORD`
b) ____ `WHEN-VALIDATE-RECORD`
c) ____ `WHEN-NEW-RECORD-INSTANCE`
d) ____ a & b

6) What is `FORM_TRIGGER_FAILURE`?

a) ____ A built-in to respond to the failure of an event
b) ____ A built-in you can use to crash the operating system
c) ____ A built-in exception to help you handle errors
d) ____ An event you can respond to with the `ON-ERROR` trigger

Quiz answers appear in Appendix A, Section 6.2.

LAB 6.3

FORMS BUILT-INS

LAB OBJECTIVES

After this Lab, you will be able to:

- Use Forms Built-ins

The Forms built-ins are a set of PL/SQL functions and procedures that perform standard application functions. You have already used built-ins like `EXIT_FORM` and `COMMIT_FORM` in previous Labs.

In these cases, you simply typed the built-in's name, and in doing so, accepted its default functionality. But, like the PL/SQL functions and procedures you have written yourself, most built-ins can accept parameters. The parameters you pass a built-in will affect its behavior.

FOR EXAMPLE:

When you used the `EXIT_FORM` built-in, you didn't pass it any parameters. The code looked like this:

```
EXIT_FORM;
```

However, the `EXIT_FORM` built-in can also accept parameters that affect what the form does when it exits. It might look like this:

```
EXIT_FORM('DO_COMMIT');
```

By passing `EXIT_FORM` the `DO_COMMIT` parameter, you are telling the form to validate and commit any outstanding changes in the form as well as exit the form.

go to contents

There are hundreds of built-ins in Oracle Forms, and you will learn and use many of them throughout the course of this book. In the next few sections, you will be introduced to some of the more common types of built-ins. A comprehensive list of all the built-ins and their individual functions and uses is provided by the Forms help system.

GET_ BUILT-INS

There are a number of built-ins that are prefixed with the word "GET_". You used the GET_APPLICATION_PROPERTY in Lab 6.1 to get the user's name and assign it to the CREATED_BY and MODIFIED_BY items. The code looked like this:

```
:COURSE.CREATED_BY := GET_APPLICATION_PROPERTY(USERNAME);
```

This specific built-in is used to get information about the application. There are other GET_ built-ins that you can use to get properties about other Forms objects such as items, blocks, canvases, and so on. It is also possible, and quite common, to assign the results of a GET_ built-in to a variable.

■ *FOR EXAMPLE:*

If you wanted to assign the user name to a variable called v_username, the code would look like this:

```
DECLARE
    v_user_name VARCHAR2(50);
BEGIN
    v_user_name := GET_APPLICATION_PROPERTY(USERNAME);
END;
```

SET_ BUILT-INS

There is another group of built-ins that are prefixed with the word "SET_". As you can imagine, you use them to set certain values.

■ *FOR EXAMPLE:*

You can use the SET_BLOCK_PROPERTY built-in to set properties about a block at run-time. The following two statements set the ORDER BY clause and the WHERE clause for a block called SECTION:

```
SET_BLOCK_PROPERTY('SECTION',DEFAULT_WHERE,'INSTRUCTOR_ID=101');
SET_BLOCK_PROPERTY('SECTION', ORDER_BY, 'SECTION_ID);
```

The SET_ built-ins are accepting three parameters: the name of the object to be adjusted, the name of the property to be set, and the value to give that property. There are SET_ built-ins for other objects in Forms, like windows, items, canvases, and so on.

FIND_ BUILT-INS

In both cases above, you used the SECTION objects' name in the SET_ statements. While this is correct, it is slightly inefficient since Forms must use resources to look up the object by name. Every object in Forms is assigned a unique object ID at run-time. Since you are referring to the SECTION block more than once, it is better to refer to it by its ID in the built-ins so that Forms can look up the object more efficiently. To get an object's ID, you must use one of the FIND_ built-ins. In this case, since you are working with a block, you would use the FIND_BLOCK built-in. You would use it to find the ID of SECTION, and then use that ID in the subsequent SET_ statements. The code would look like this:

```
DECLARE
    v_block_id  BLOCK;
BEGIN
    v_block_id := FIND_BLOCK('SECTION');
    SET_BLOCK_PROPERTY(v_block_id, DEFAULT_WHERE,
'INSTRUCTOR_ID = 101');
    SET_BLOCK_PROPERTY(v_block_id, ORDER_BY,
'SECTION_ID');
END;
```

The variable v_block_id is used to hold the result of the FIND_BLOCK built-in. Then, v_block_id is used in the subsequent SET_ statements to identify the object. Now, the SECTION object is only referenced by name once, in the FIND_BLOCK statement. The SET_ statements use the object ID for the SECTION block, which makes the code much more efficient.

LAB 6.3 EXERCISES

6.3.1 USE FORMS BUILT-INS

In this Exercise, you will use built-ins to manipulate the properties of a window at run-time.

Open the form `EX06_03.fmb` in the Form Builder. Create a `WHEN-NEW-FORM-INSTANCE` trigger at the form level.

a) How can you use the `GET_APPLICATION_PROPERTY` built-in to get the name of the current form module?

b) What built-in can you use to size the window `MAINWIN` to `200`, `200`? Try to do this using only one built-in statement.

c) How can you use the same built-in from Question b to set the title for `MAINWIN`? The title should be as follows:

```
'This is form... <FORM NAME> '
```

At the end of the title, you should insert the result of the `GET_APPLICATION_PROPERTY` built-in from Question a.

Run the form and test the built-ins.

d) What can you do to refer to `MAINWIN` more efficiently than by using its object name? Change the code in the `WHEN-NEW-FORM-INSTANCE` trigger to do this.

go to contents

e) Would this trigger fire properly if MAINWIN did not exist?

Create a PRE-RECORD trigger for the ZIPCODE block and give it the following code:

```
GO_ITEM('ZIPCODE.CITY');
```

Run the form and issue a query.

f) What was the error you received in the hint line?

LAB 6.3 EXERCISE ANSWERS

6.3.1 ANSWERS

a) How can you use the GET_APPLICATION_PROPERTY built-in to get the name of the current form module?

Answer: See the discussion below.

GET_APPLICATION_PROPERTY is a built-in function that returns a value, so in this case, its result should be assigned to a variable as is shown in the code below:

```
DECLARE
  v_form_name     VARCHAR2(50);
BEGIN
  v_form_name := GET_APPLICATION_PROPERTY(CURRENT_FORM_NAME);
END;
```

There are many GET_ built-ins that allow you to get properties for objects or information from the system. You can use GET_ITEM_PROPERTY, GET_CANVAS_PROPERTY, and GET_BLOCK_PROPERTY as well to find the current value of a property and then act on it. In this Exercise, you will

get the current form name property for the application item and then display it to the user in the window's title. In the last Exercise, you used the GET_APPLICATION_PROPERTY to get the current user name and insert it into the database.

Note that the v_form_name variable is a VARCHAR2. This corresponds with the data type of the value that the GET_APPLICATION_PROPERTY returns. The built-in would have failed if you had set v_form_name to NUMBER, BOOLEAN, or another data type. This applies to all built-ins that accept and return values; you must always be conscious of the data type that the built-in is using so that you can write your code accordingly. You may have noticed that when you read about the GET_APPLICATION_PROPERTY built-in in the help system, there was information about the parameters the built-in accepts and the data types it returns. This type of information is available for all built-ins.

b) What built-in can you use to size the window MAINWIN to 200, 200? Try to do this using only one built-in statement.

Answer: See the discussion below.

You could have used the following built-in:

```
SET_WINDOW_PROPERTY('MAINWIN', WINDOW_SIZE, 200, 200);
```

The SET_ built-ins are similar to the GET_ built-ins in that you can SET_ properties for most Forms objects. You can use SET_ITEM_PROPERTY, SET_CANVAS_PROPERTY, SET_BLOCK_PROPERTY, and many others to set the value of a property.

Note that in the SET_WINDOW_PROPERTY example, the same rules regarding data types of values apply. The syntax for the built-in is as follows:

```
SET_WINDOW_PROPERTY(object name, property, value);
```

The object name accepts a VARCHAR2 parameter, so the value must be in quotes. The value in the example above was a NUMBER since you were setting the size of the window. However, the data type of the value can change depending on the type of property you are setting. In the next example, you will set the title of the window, which will require that you pass the built-in a VARCHAR2 value. What this discussion illustrates is that when you begin to use built-ins, it is important that you consult the

help files often to confirm that you are using the proper syntax and that you are passing parameters using the proper data types.

c) How can you use the same built-in from Question b to set the title for MAIN-WIN? The title should be as follows:

```
'This is form . . . <FORM NAME>'
```

At the end of the title, you should insert the result of the GET_APPLICATION_PROPERTY built-in from Question a.

Answer: See the discussion below.

Again, you would use the SET_WINDOW_PROPERTY to set the title. This statement would be as follows:

```
SET_WINDOW_PROPERTY('MAINWIN', TITLE, 'This is form '||v_form_name);
```

The syntax is the same as in the SET_WINDOW_PROPERTY statement that you used to set the window size. What is different are the values that you are passing to the built-in.

Note that you passed the v_form_name variable into the built-in. This is very common in that it keeps you from hard-coding values into the built-in.

Run the form and test the built-ins.

d) What can you do to refer to MAINWIN more efficiently than by using its object name? Change the code in the WHEN-NEW-FORM-INSTANCE trigger to do this.

Answer: You can refer to it by object ID using the FIND_WINDOW *built-in.*

The FIND_WINDOW built-in will get the window's object ID for you. Then you can use the item ID in the SET_ built-ins instead of the item name. As you learned in the Lab text, it takes fewer resources to refer to an object by its ID than to refer to it by name, so whenever possible, it is more efficient to refer to an object by its ID.

Just like GET_ and SET_, there are FIND_ built-ins for virtually every object in Forms, all of which will return an object's system ID. To use the FIND_ built-ins, you must employ variables of specific types.

■ *FOR EXAMPLE:*

When you declare a variable to hold the system ID of a block, the variable must be of type BLOCK. So, if you want to FIND_ the ID of the ZIP-

CODE block and assign it to a variable called v_block_id, the code would look like this:

```
DECLARE
    v_block_id BLOCK;
BEGIN
    v_block_id := FIND_BLOCK('ZIPCODE');
…
```

Variables to hold the IDs of items would be of type ITEM, blocks of type BLOCK, and so on.

The code in the WHEN-NEW-FORMS-INSTANCE trigger should look like this:

```
DECLARE
  v_form_name     VARCHAR2(50);
  v_window_id     WINDOW;
BEGIN
   v_form_name := GET_APPLICATION_PROPERTY(CURRENT_FORM_NAME);
   v_window_id := FIND_WINDOW('MAINWIN');
   SET_WINDOW_PROPERTY(v_window_id, WINDOW_SIZE, 200, 200);
   SET_WINDOW_PROPERTY(v_window_id, TITLE, 'This is form
'||v_form_name);
END;
```

Note that a new variable called v_window_id has been created to hold MAINWIN's object ID. v_window_id is set using the FIND_WINDOW built-in. Then, the SET_ statement's v_window_id is used to refer to the window instead of its name. Note that the v_window_id variable is not in single quotes in the built-in statements.

e) Would this trigger fire properly if MAINWIN did not exist?

Answer: No it would not.

The FIND_WINDOW built-in accepts the MAINWIN window object's name. If this window did not exist in the form, you would see the following error when the trigger tried to fire:

```
FRM-41052 Cannot find window. Invalid id.
```

This type of error is not limited to the FIND_WINDOW built-in. You would receive similar errors if any built-in tried to refer to an object that did not

exist in the form. To guard against this, it is wise to write the trigger so that it can alert you or the user to the absence of an object.

■ FOR EXAMPLE:

```
DECLARE
  v_form_name     VARCHAR2(50);
  v_window_id     WINDOW;
BEGIN
  v_form_name := GET_APPLICATION_PROPERTY(CURRENT_FORM_NAME);
  v_window_id := FIND_WINDOW('MAINWIN');
  IF ID_NULL(v_window_id) THEN
MESSAGE('MAINWIN does not exist. Error in WHEN-NEW-FORM-INSTANCE trigger');
  RAISE FORM_TRIGGER_FAILURE;
  END IF;
  SET_WINDOW_PROPERTY(v_window_id, WINDOW_SIZE, 200, 200);
SET_WINDOW_PROPERTY(v_window_id, TITLE, 'This is form '||v_form_name);
END;
```

The `ID_NULL` built-in evaluates whether or not the value `v_window_id` is null. If it is null, it means the `MAINWIN` window does not exist.

Create a `PRE-RECORD` trigger for the `ZIPCODE` block and give it the following code:

```
GO_ITEM('ZIPCODE.CITY');
```

Run the form and issue a query.

f) What was the error you received in the hint line?

Built-ins that have to do with navigation are deemed restricted. That is, they cannot be used in Navigational triggers. The `PRE-RECORD` *trigger is a Navigational trigger, as is* `PRE-TEXT-ITEM`, `POST-BLOCK`, `POST-QUERY`, *and many others. You are unable to use restricted built-ins like* `GO_ITEM`, `GO_BLOCK`, *and so on in these triggers. As you have already seen, the help system will indicate whether a built-in is restricted or not under the "Built-in Type" heading. The help for triggers will also indicate what types of built-ins they can include under the "Legal Commands" heading.*

LAB 6.3 SELF REVIEW QUESTIONS

In order to test your progress, you should be able to answer the following questions:

1) Which of the following is true about built-ins?
- **a)** ____ They are not valid in triggers
- **b)** ____ They fire at will
- **c)** ____ They are PL/SQL functions and procedures that provide standard application functionality
- **d)** ____ None of the above

2) What is true about the `EXIT_FORM` built-in?
- **a)** ____ You can pass it parameters
- **b)** ____ It gets a form out of Enter Query mode
- **c)** ____ It is not valid in `WHEN-BUTTON-PRESSED` triggers
- **d)** ____ a & b

3) Which built-in would you use to find the width of an item?
- **a)** ____ `FIND_ITEM_PROPERTY`
- **b)** ____ `GET_APPLICATION_PROPERTY`
- **c)** ____ `GET_ITEM_PROPERTY`
- **d)** ____ a & c

4) Which parameters can `SET_ITEM_PROPERTY` accept?
- **a)** ____ Item names
- **b)** ____ Object IDs
- **c)** ____ Property names
- **d)** ____ All of the above

5) Which of the following cannot be used in the `SET_ITEM_PROPERTY` built-in?
- **a)** ____ Variables as parameters
- **b)** ____ The `:BLOCK.ITEM` syntax
- **c)** ____ The `Prompt` property
- **d)** ____ Non-navigable items

6) What could you use to get an object's ID?
- **a)** ____ A `GET_` statement
- **b)** ____ A system variable
- **c)** ____ A `FIND_` built-in
- **d)** ____ a & c

Quiz answers appear in Appendix A, Section, 6.3.

go to contents

CHAPTER 6

TEST YOUR THINKING

1) Open the R_STUDENT.fmb form that you created in the "Test Your Thinking" section of Chapter 2 to complete this question. Add two display items called CITY and STATE to the form. Populate these display items with a POST-QUERY trigger. Validate STUDENT.ZIPCODE with a WHEN-VALIDATE-ITEM trigger.

2) Open the R_INSTRUCTOR.fmb form that you created in the "Test Your Thinking" section of Chapter 2 to complete this question. Add two display items called CITY and STATE to the form. Populate these display items with a POST-QUERY trigger. Validate INSTRUCTOR.ZIPCODE with a WHEN-VALIDATE-ITEM trigger.

3) Open the R_CRSESECT.fmb form you created in the "Test Your Thinking" section of Chapter 4 to complete this question. Add one display item to the SECTION block for the INSTRUCTOR_NAME. Populate this display item with a POST-QUERY trigger. The INSTRUCTOR_NAME item should contain both the FIRST_NAME and LAST_NAME of the instructor. The name should appear as follows:

```
Joe Smith
```

Validate INSTRUCTOR_ID with a WHEN-VALIDATE-ITEM trigger.

4) Open the R_STUDENRL.fmb form you created in the "Test Your Thinking" section of Chapter 4 to complete this question. Add two display items to the ENROLLMENT block: one for the COURSE_NO of the course, and one for the LOCATION of the course. Populate these display items with a POST-QUERY trigger.

Validate SECTION_ID with WHEN-VALIDATE-ITEM triggers.

go to contents