Database Programming with PL/SQL

15-3
Using Conditional Compilation
Objectives

This lesson covers the following objectives:

• Describe the benefits of conditional compilation

• Create and conditionally compile a PL/SQL program containing selection, inquiry, and error directives

• Create and conditionally compile a PL/SQL program which calls the `DBMS_DB_VERSION` server-supplied package
Purpose

• Imagine you are creating a presentation as part of a school project. You create it on your home PC using Microsoft Office 2016, which has some nice new features such as a holographic graph display.

• When it’s finished, you will present your work to your class at school, but until the day of the presentation you won’t know if the classroom PC will have Office 2016 or an older version such as Office 2010, which can’t display holographic graphs.
Purpose

• You want to include holographic graphs in your presentation, but you don’t want it to fail while you are presenting to your class.

• Conditional compilation can prevent an embarrassing moment.
What is Conditional Compilation?

• Wouldn’t it be great if you could somehow create your presentation so that if you show it using Office 2016, the holographic graphs are displayed, but if you show it using Office 2010, the standard graphs are displayed instead?

• That way, it won’t fail regardless of the Office version you use, and you automatically get the benefit of the new features if they’re available.

• You can do exactly this in PL/SQL by using Conditional Compilation.
What is Conditional Compilation?

• Conditional Compilation allows you to include some source code in your PL/SQL program that may be compiled or may be ignored (like a comment is ignored), depending on:
  – the values of an initialization parameter
  – the version of the Oracle software you are using
  – the value of a global package constant
  – any other condition that you choose to set.

• You control conditional compilation by including directives in your source code. Directives are keywords that start with a single or double dollar sign ($ or $$).
Conditional Compilation and Microsoft Office

• You can’t really use PL/SQL with Microsoft Office, so this is not a real example, but let’s pretend:

```sql
CREATE OR REPLACE PROCEDURE lets_pretend IS
BEGIN
...
  $IF MS_OFFICE_VERSION >= '2016' $THEN
    include_holographics;
  $ELSE
    include_std_graphic;
  $END
...
END lets_pretend;
```

• `$IF, $THEN, $ELSE` and `$END` are *selection directives*. 
Conditional Compilation and Oracle Versions

• You can’t test which Office version you’re using, but you can test which Oracle version you’re using.
• This is a "real" subprogram:

```
CREATE OR REPLACE FUNCTION lets_be_real
  RETURN NUMBER
  $IF DBMS_DB_VERSION.VERSION >= 11 $THEN
    DETERMINISTIC
  $END
IS BEGIN
  RETURN 17; -- real source code here !
END lets_be_real;

BEGIN
  DBMS_OUTPUT.PUT_LINE('Function returned ' || lets_be_real);
END lets_be_real;
```
Conditional Compilation and Oracle Versions

- Deterministic functions are new in Oracle at Version 11.
- This code includes the word `DETERMINISTIC` if we compile the function on Version 11 or later, and is ignored if we compile on Version 10 or earlier.

```sql
CREATE OR REPLACE FUNCTION lets_be_real
  RETURN NUMBER
  $IF DBMS_DB_VERSION.VERSION >= 11 $THEN
    DETERMINISTIC
  $END
IS BEGIN
  RETURN 17; -- real source code here!
END lets_be_real;
```
What is in the Data Dictionary Now?

- After compiling the function on the previous slide, what is stored in the Data Dictionary?

- `USER_SOURCE` contains your complete source code, including the compiler directives:

  ```sql
  SELECT text FROM USER_SOURCE
  WHERE name = 'LET'S BE REAL'
  ORDER BY line;
  ```

<table>
<thead>
<tr>
<th>TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTION lets_be_real</td>
</tr>
<tr>
<td>RETURN NUMBER</td>
</tr>
<tr>
<td>$IF DBMS_DB_VERSION.VERSION &gt;= 11 $THEN</td>
</tr>
<tr>
<td>DETERMINISTIC</td>
</tr>
<tr>
<td>$END</td>
</tr>
<tr>
<td>IS BEGIN</td>
</tr>
<tr>
<td>RETURN 17; -- real source code here!</td>
</tr>
<tr>
<td>END lets_be_real;</td>
</tr>
</tbody>
</table>
Seeing Which Code has been Compiled

• If you want to see which code has actually been included in your compiled program, you use the **DBMS_PREPROCESSOR** Oracle-supplied package:

```sql
BEGIN
    DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE
    ('FUNCTION','<YOUR_USERNAME>','lets_be_real');
END;
```

• This function was compiled using Oracle Version 11:

```sql
FUNCTION lets_be_real
    RETURN NUMBER
    DETERMINISTIC
IS BEGIN
    RETURN 17; -- real source code here!
END lets_be_real;
```
Using Selection Directives

• There are five selection directives: $IF, $THEN, $ELSE, $ELSIF and $END (not $ENDIF).

• Their logic is the same as IF, THEN, ELSE and so on, but they control which code is included at compile time, not what happens at execution time.

...  
$IF condition $THEN statement(s);  
$ELSE statement(s);  
$ELSIF statement(s);  
$END  
...

• Notice that $END does not end with a semicolon(;) unlike END;
Using Selection Directives Example

• You have created a bodiless package that declares a number of global constants:

```sql
CREATE OR REPLACE PACKAGE tax_code_pack IS
    new_tax_code CONSTANT BOOLEAN := TRUE;
    -- but could be FALSE
    ...
END tax_code_pack;
```

• Now, you want to create a subprogram that declares an explicit cursor whose `WHERE` clause will depend on the value of the Boolean package constant.
Using Selection Directives Example

• Now let’s look at the contents of the Data Dictionary.
• Remember, the package set **NEW_TAX_CODE** to **TRUE**.

```sql
CREATE OR REPLACE PROCEDURE get_emps IS
    CURSOR get_emps_curs IS
        SELECT * FROM employees
        WHERE
            $IF tax_code_pack.new_tax_code $THEN
                salary > 20000;
            $ELSE
                salary > 50000;
            $END
    BEGIN
        FOR v_emps IN get_emps_curs LOOP
            /* real code here */
        END LOOP;
    END get_emps;
```
Using Selection Directives Example

What’s in the Data Dictionary?

```
SELECT text FROM USER_SOURCE
  WHERE type = 'PROCEDURE' and name = 'GET_EMPS'
ORDER BY line;
```

---

### EXAMPLE

<table>
<thead>
<tr>
<th>TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCEDURE get_emps IS</td>
</tr>
<tr>
<td>CURSOR get_emps_curs IS</td>
</tr>
<tr>
<td>SELECT * FROM employees</td>
</tr>
<tr>
<td>WHERE</td>
</tr>
<tr>
<td>$IF tax_code_pack.new_tax_code $THEN salary &gt; 20000;</td>
</tr>
<tr>
<td>$ELSE</td>
</tr>
<tr>
<td>salary &gt; 50000;</td>
</tr>
<tr>
<td>$END</td>
</tr>
<tr>
<td>BEGIN</td>
</tr>
<tr>
<td>FOR v_emps IN get_emps_curs LOOP</td>
</tr>
<tr>
<td>NULL; /* real code here */</td>
</tr>
<tr>
<td>END LOOP;</td>
</tr>
<tr>
<td>END get_emps;</td>
</tr>
</tbody>
</table>
Using Selection Directives Example

And what code was compiled?

```
BEGIN
    DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE
    ('PROCEDURE','<YOUR_USERNAME>','GET_EMPS');
END;
```

```
PROCEDURE get_emps IS
    CURSOR get_emps_curs IS
        SELECT * FROM employees
        WHERE
            salary > 20000;

BEGIN
    FOR v_emps IN get_emps_curs LOOP
        NULL; /* real code here */
    END LOOP;
END get_emps;
```
The **PLSQL_CCFLAGS** Initialization Parameter

- You may want to use the selection directives, such as `$IF`, to test for a condition that has nothing to do with global package constants or Oracle software versions.
- For example, you may want to include extra code to help you debug a PL/SQL program, but once the errors have been corrected, you do not want to include this code in the final version because it will slow down the performance.
- You can control this using the **PLSQL_CCFLAGS** initialization parameter.
The **PLSQL_CCFLAGS** Parameter

- **PLSQL_CCFLAGS** allows you to set values for variables, and then test those variables in your PL/SQL program.
- You define the variables and assign values to them using **PLSQL_CCFLAGS**.
- Then, you test the values of the variables in your PL/SQL program using *inquiry directives*.
- An inquiry directive is the name of the variable prefixed by a double dollar sign ($$).
Using **PLSQL_CCFLAGS** and Inquiry Directives

- First, set the value of the parameter:

```sql
ALTER SESSION SET PLSQL_CCFLAGS = 'debug:true';
```

- Then compile your PL/SQL program:

```sql
CREATE OR REPLACE PROCEDURE testproc IS BEGIN
...
  $IF $$debug $THEN
    DBMS_OUTPUT.PUT_LINE('This code was executed');
  $END
...
END testproc;
```
Using \texttt{PLSQL\_CCFLAGS} and Inquiry Directives

• After you have debugged the program, remove the debugging code by:

\begin{verbatim}
ALTER SESSION SET PLSQL\_CCFLAGS = 'debug:false';
\end{verbatim}

• Compile your procedure one more time to remove the debugging code.
Using **PLSQL_CCFLAGS** and Inquiry Directives

DEBUG is not a keyword: you can use any name you like, and it can be a number or a character string, not just a Boolean.

```sql
ALTER SESSION SET PLSQL_CCFLAGS = 'testflag:1';
```

```plsql
CREATE OR REPLACE PROCEDURE testproc IS BEGIN...
  $IF $$testflag > 0 $THEN
    DBMS_OUTPUT.PUT_LINE('This code was executed');
  $END
...
END testproc;
```
Using **PLSQL_CCFLAGS** and Inquiry Directives

You can set more than one variable, and then test them either together or separately:

```sql
ALTER SESSION SET PLSQL_CCFLAGS = 'firstflag:1, secondflag:false';
```

```sql
CREATE OR REPLACE PROCEDURE testproc IS BEGIN
  ...
  $IF $$firstflag > 0 AND NOT $$secondflag $THEN
    DBMS_OUTPUT.PUT_LINE('Testing both variables');
  $ELSIF $$secondflag $THEN
    DBMS_OUTPUT.PUT_LINE('Testing one variable');
  $END
  ...
END testproc;
```
Using `PLSQL_CCFLAGS` and Inquiry Directives

You can see which settings of `PLSQL_CCFLAGS` were used to compile a program by querying `USER_PLSQL_OBJECT_SETTINGS`.

```sql
SELECT name, plsql_ccflags
FROM USER_PLSQL_OBJECT_SETTINGS
WHERE name = 'TESTPROC';
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>PLSQL_CCFLAGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TESTPROC</td>
<td>firstflag:1, secondflag: false</td>
</tr>
</tbody>
</table>
Using `PLSQL_CCFLAGS` and Inquiry Directives

And, as always, you can see what was included in the compiled program using `DBMS_PREPROCESSOR`.

```plsql
BEGIN
    DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE ('PROCEDURE', '<YOUR_USERNAME>', 'TESTPROC');
END;
```

```plsql
PROCEDURE testproc IS BEGIN
    --
    DBMS_OUTPUT.PUT_LINE('Testing both variables');
    --
END testproc;
```
Using **DBMS_DB_VERSION**

- **DBMS_DB_VERSION** is a bodiless package that defines a number of constants, including:
  - **VERSION** *(the current Oracle software version)*
  - **VER_LE_11** *(= TRUE if the current Oracle software is version 11 or earlier)*
  - **VER_LE_10** *(= TRUE if the current Oracle software is version 10 or earlier)*

- So:
  
  $\text{IF DBMS_DB_VERSION.VER_LE_11 } \text{ THEN } \ldots$

- Is exactly the same as:

  $\text{IF DBMS_DB_VERSION.VERSION } \leq 11 \text{ THEN } \ldots$
Terminology

Key terms used in this lesson included:

• Conditional compilation
• DBMS_DB_VERSION
• DBMS_PREPROCESSOR
• Inquiry and selection directives
• PLSQL_CCFLAGS
• USER_SOURCE
Summary

In this lesson, you should have learned how to:

• Describe the benefits of conditional compilation

• Create and conditionally compile a PL/SQL program containing selection, inquiry, and error directives

• Create and conditionally compile a PL/SQL program which calls the `DBMS_DB_VERSION` server-supplied package