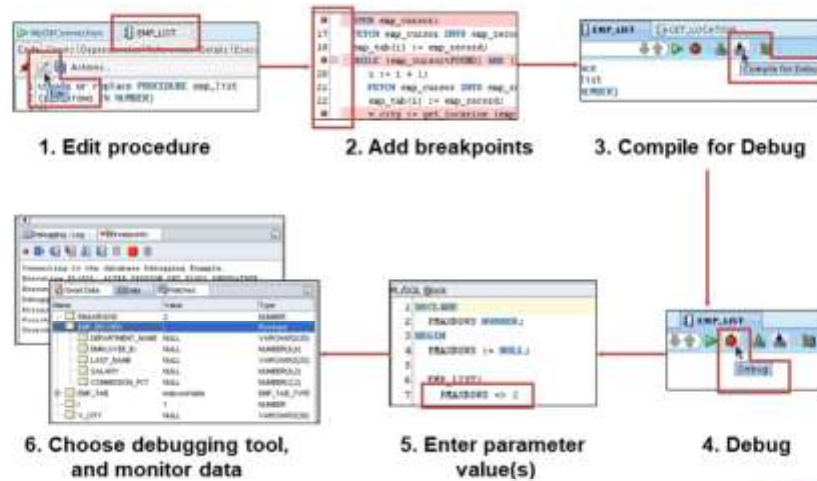


## Debugging a Subprogram: Overview



ORACLE

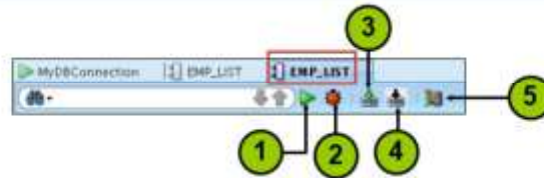
11-2

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

### Moving Through Code While Debugging

The SQL Developer debugger enables you to control the execution of your program. You can control whether your program executes a single line of code, an entire subprogram (procedure or function), or an entire program block. By manually controlling when the program should run and when it should pause, you can quickly move over the sections that you know work correctly and concentrate on the sections that are causing problems.

## The Procedure or Function Tab Toolbar



Icon	Description
1. Run	Starts normal execution of the function or procedure, and displays the results in the Running - Log tab
2. Debug	Executes the subprogram in debug mode, and displays the Debugging - Log tab, which includes the debugging toolbar for controlling execution
3. Compile	Compiles the subprogram
4. Compile for Debug	Compiles the subprogram so that it can be debugged
5. Profile	Displays the Profile window that you use to specify parameter values for running, debugging, or profiling a PL/SQL function or procedure

ORACLE


11-4

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

### The Procedure or Function Code Tab

This tab displays a toolbar and the text of the subprogram, which you can edit. You can set and unset breakpoints for debugging by clicking to the left of the thin vertical line beside each statement with which you want to associate a breakpoint. (When a breakpoint is set, a red circle is displayed.)

## The Debugging – Log Tab Toolbar



Icon	Description
1. Find Execution Point	Goes to the next execution point
2. Step Over	Bypasses the next subprogram and goes to the next statement after the subprogram
3. Step Into	Executes a single program statement at a time. If the execution point is located on a call to a subprogram, it steps into the first statement in that subprogram
4. Step Out	Leaves the current subprogram and goes to the next statement with a breakpoint

**ORACLE**

11-5 Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

### The Debugging – Log Tab Toolbar

The Debugging - Log contains the debugging toolbar and informational messages.

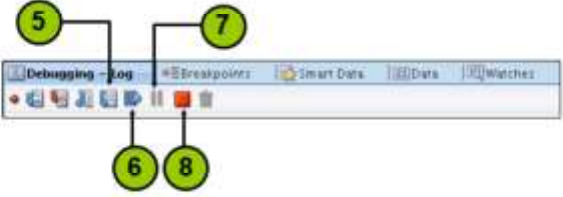
1. Find Execution Point: Goes to the execution point (the next line of source code to be executed by the debugger)
2. Step Over: Bypasses the next subprogram (unless the subprogram has a breakpoint) and goes to the next statement after the subprogram. If the execution point is located on a subprogram call, it runs that subprogram without stopping (instead of stepping into it), and then positions the execution point on the statement that follows the call. If the execution point is located on the last statement of a subprogram, Step Over returns from the subprogram, placing the execution point on the line of code that follows the call to the subprogram from which you are returning.
3. Step Into: Executes a single program statement at a

time. If the execution point is located on a call to a subprogram, Step Into steps into that subprogram and places the execution point on its first statement. If the execution point is located on the last statement of a subprogram, Step Into returns from the subprogram, placing the

execution point on the line of code that follows the call to the subprogram from which you are returning.

4. Step Out: Leaves the current subprogram and goes to the next statement

## The Debugging – Log Tab Toolbar



Icon	Description
5. Step to End of Method	Goes to the last statement of the current subprogram
6. Resume	Continues execution
7. Pause	Halts execution but does not exit
8. Terminate	Halts and exits the execution

**ORACLE**

11-6 Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

### The Debugging – Log Tab Toolbar (continued)

5. Step to End of Method: Goes to the last statement of the current subprogram
6. Resume: Continues execution
7. Pause: Halts execution but does not exit, thus allowing you to resume execution
8. Terminate: Halts and exits the execution. You cannot resume execution from this point; instead, to start running or debugging from the beginning of the subprogram, click the Run or Debug icon in the Source tab toolbar.

## Additional Tabs

Name	Value	Type
P_MAXROWS	100	NUMBER
REC_EMP	Rowtype	EMP_TAB_TYPE
EMP_TAB	indexed table	EMP_TAB_TYPE
I	1	NUMBER
V_CITY	NULL	VARCHAR2(30)

Tab	Description
Breakpoints	Displays breakpoints, both system-defined and user-defined.
Smart Data	Displays information about variables. You can specify these preferences by right-clicking in the Smart Data window and selecting Preferences.
Data	Located under the code text area; displays information about all variables
Watches	Located under the code text area; displays information about watches

ORACLE

11.7

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

### Additional Tabs

#### Setting Expression Watches

A watch enables you to monitor the changing values of variables or expressions as your program runs. After you enter a watch expression, the Watches window displays the current value of the expression. As your program runs, the value of the watch changes as your program updates the values of the variables in the watch expression.

A watch evaluates an expression according to the current context, which is controlled by the selection in the Stack window. If you move to a new context, the expression is reevaluated for the new context. If the execution point moves to a location where any of the variables in the watch expression are undefined, the entire watch expression becomes undefined. If the execution point returns to a location where the watch expression can be evaluated, the Watches window again displays the value of the watch expression.

To open the Watches window, click View, then Debugger, then Watches. To add a watch, right-click in the Watches window and select Add Watch. To edit a watch, right-click in the Watches window and select Edit Watch. Note: If you cannot see some of the debugging tabs described in this lesson, you can re-display such tabs by using the View > Debugger menu option.

## Debugging a Procedure Example: Creating a New emp\_list Procedure

```
1 CREATE OR REPLACE PROCEDURE emp_list(pnameemp IN NUMBER) IS
2 BEGIN
3   SELECT d.department_name,
4          e.employee_id,
5          e.last_name,
6          e.salary,
7          e.commission_pct
8   FROM department d,
9        employee e
10  WHERE d.department_id = e.department_id;
11 emp_record emp_record%TYPE;
12 emp_tab emp_tab%TYPE := TABLE OF emp_record%TYPE BY BINARY_INTEGER;
13 emp_tab := emp_tab%TYPE;
14 i NUMBER := 1;
15 v_city VARCHAR2(100);
16 BEGIN
17   OPEN emp_cursor;
18   FETCH emp_cursor;
19   FETCH emp_cursor;
20   FETCH emp_cursor;
21   emp_tab(i) := emp_record;
22   WHILE (emp_cursor % FOUND)
23   LOOP
24     i := i + 1;
25     FETCH emp_cursor;
26     INTO emp_record;
27     emp_tab(i) := emp_record;
28     v_city := get_location(emp_record.department_name);
29     LABEL_OUTPUT.PUT_LINE('Employee ' || emp_record.last_name || ' works in ' || v_city);
30   END LOOP;
31   CLOSE emp_cursor;
32   FOR j IN REVERSE 1..i
33   LOOP
34     LABEL_OUTPUT.PUT_LINE(emp_tab(j).last_name);
35   END LOOP;
36 END emp_list;
```

ORACLE

11-8

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Debugging a Procedure Example: Creating a New emp\_list Procedure The emp\_list procedure gathers employee information such the employee's department name, ID, name, salary, and commission percentage. The procedure creates a record to store the employee's information. The procedure also creates a table that can hold multiple records of employees.

"i" is a counter.

The code opens the cursor and fetches the employees' records. The code also checks whether or not there are more records to fetch or whether the number of records fetched so far is less than the number of records that you specify. The code eventually prints out the employees' information. The procedure also calls the get\_location function that returns the name of the city in which an employee works.

Note: Make sure that you are displaying the procedure code in edit mode. To edit the procedure code, click the Edit icon on the procedure's toolbar.



## Debugging a Procedure Example: Creating a New get\_location Function

```
1 CREATE OR REPLACE FUNCTION get_location(p_deptname IN VARCHAR2) RETURN VARCHAR2 AS
2   v_loc_id NUMBER;
3   v_city VARCHAR2(30);
4 BEGIN
5   SELECT d.location_id,
6         l.city
7   INTO v_loc_id,
8        v_city
9   FROM departments d,
10        locations l
11  WHERE UPPER(department_name) = UPPER(p_deptname)
12         AND d.location_id = l.location_id;
13   RETURN v_city;
14 END get_location;
```

ORACLE

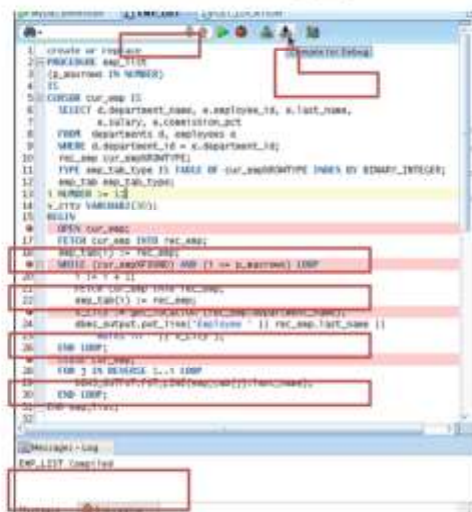
11-9

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

## Debugging a Procedure Example: Creating a New Procedure

This function returns the city in which an employee works. It is called from the emp\_list procedure.

## Setting Breakpoints and Compiling emp\_list for Debug Mode



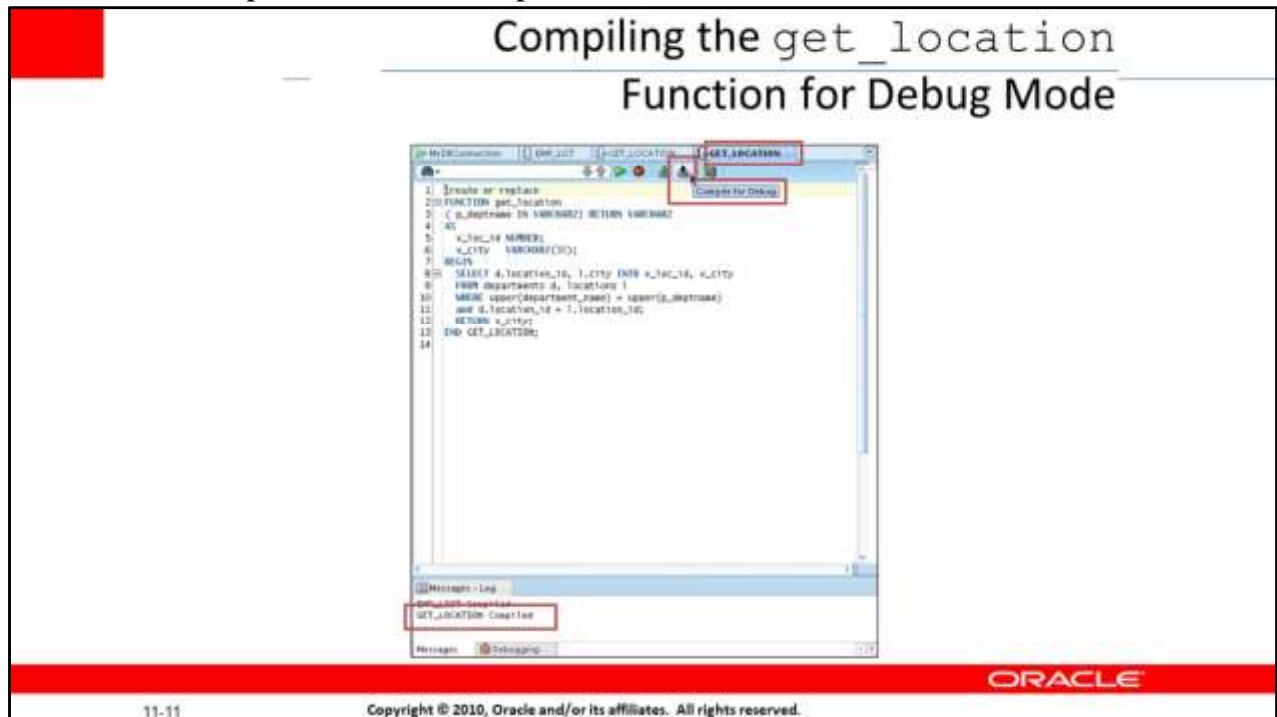
ORACLE

11-10

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

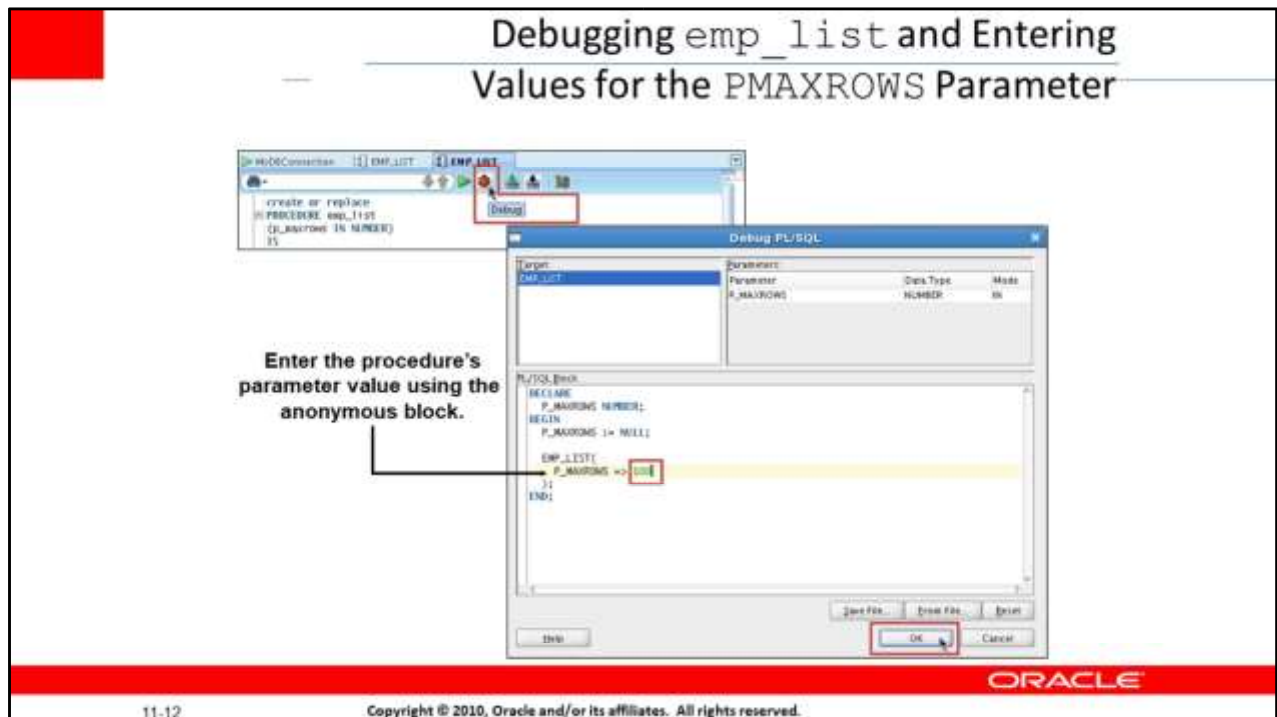
## Setting Breakpoints and Compiling emp\_list for Debug Mode

In the example in the slide, the emp\_list procedure is displayed in edit mode, and four breakpoints are added to various locations in the code. To compile the procedure for debugging, right-click the code, and then select Compile for Debug from the shortcut menu. The Messages – Log tab displays the message that the procedure was compiled.



## Compiling the get\_location Function for Debug Mode

In the example in the slide, the get\_location function is displayed in edit mode. To compile the function for debugging, right-click the code, and then select Compile for Debug from the shortcut menu. The Messages – Log tab displays the message that the function was compiled.

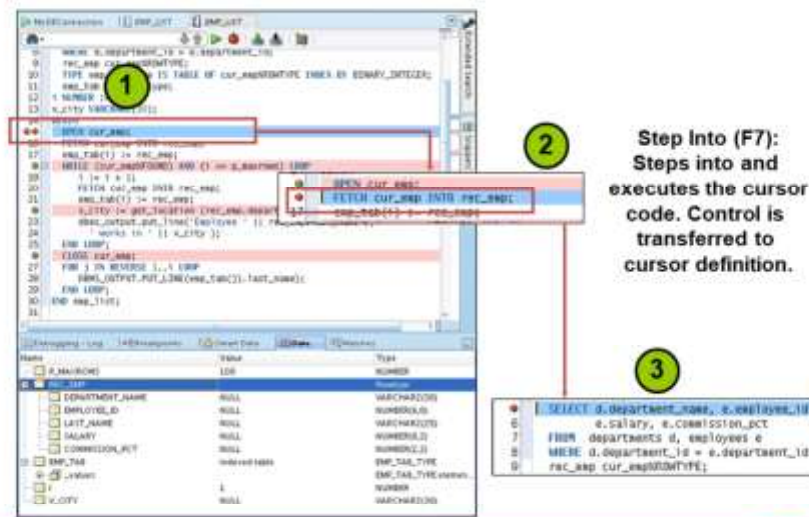


## Debugging emp\_list and Entering Values for the P\_MAXROWS Parameter

The next step in the debugging process is to debug the procedure using any of the several available methods mentioned earlier, such as clicking the Debug icon on the procedure's toolbar. An anonymous block is displayed, where you are prompted to enter the parameters for this procedure. emp\_list has one parameter, P\_MAXROWS, which specifies the number of records to return. Replace the second P\_MAXROWS with a number such as 100, and then click OK.



## Debugging emp\_list: Step Into (F7) the Code



11.14

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

### Debugging emp\_list: Step Into the Code

Selecting the Step Into command executes a single program statement at a time.

If the execution point is located on a call to a subprogram, the Step Into command steps into that subprogram and places the execution at the first statement in the subprogram.

In the example in the slide, pressing F7 executes the line of code at the first breakpoint. In this case, program control is transferred to the section where the cursor is defined.

## Viewing the Data

The screenshot shows the Oracle IDE's Data tab. The variable `EMP_ID` is selected, and its value is `100`. The Data pane shows the variable's type as `NUMBER(8,0)`. The variable is part of a collection named `EMP_TAB`, which is of type `EMP_TAB_TYPE`. The collection contains one element, `EMP_ID`, with a value of `100`. The variable `EMP_ID` is of type `NUMBER(8,0)`. The variable `EMP_ID` is of type `NUMBER(8,0)`.

11-15

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

## Viewing the Data

While you are debugging your code, you can use the Data tab to display and modify the variables. You can also set watches to monitor a subset of the variables displayed in the Data tab. To display or hide the Data, Smart Data, and Watch tabs: Select View > Debugger, and then select the tabs that you want to display or hide.

## Modifying the Variables While Debugging the Code

The screenshot shows the Oracle IDE's Data tab. The variable `EMP_ID` is selected, and its value is `100`. The Data pane shows the variable's type as `NUMBER(8,0)`. The variable is part of a collection named `EMP_TAB`, which is of type `EMP_TAB_TYPE`. The collection contains one element, `EMP_ID`, with a value of `100`. The variable `EMP_ID` is of type `NUMBER(8,0)`. The variable `EMP_ID` is of type `NUMBER(8,0)`.

11-16

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE



## Modifying the Variables While Debugging the Code

To modify the value of a variable in the Data tab, right-click the variable name, and then select Modify Value from the shortcut menu. The Modify Value window is displayed. The current value for the variable is displayed. You can enter a new value in the second text box, and then click OK.

**Debugging emp\_list: Step Over the Code**

**Step Over (F8):**  
Executes the Cursor  
(same as F7),  
but control is not transferred  
to Open Cursor code

```
14 BEGIN
15 OPEN cur_emp;
16 FETCH cur_emp INTO rec_emp;
17 emp_tab(i) := rec_emp;
18 WHILE (cur_emp%FOUND) AND (i <= 2000000) LOOP
19     i := i + 1;
20     FETCH cur_emp INTO rec_emp;
21     emp_tab(i) := rec_emp;
22     v_city := get_location(rec_emp.department_name);
23     dbms_output.put_line('Employee ' || rec_emp.last_name ||
24                           ' works in ' || v_city);
25 END;
```

11-17 Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

## The Step Over Debugging Tool

The Step Over command, like Step Into, enables you to execute program statements one at a time. However, if you issue the Step Over command when the execution point is located on a subprogram call, the debugger runs that subprogram without stopping (instead of stepping into it), and then positions the execution point on the statement that follows the subprogram call. If the execution point is located on the last statement of a subprogram, choosing Step Over causes the debugger to return from the subprogram, placing the execution point on the line of code that follows the call to the subprogram you are returning from. You can step over a subprogram in any of the following ways: Select Debug > Step Over, press F8, or click the Step Over icon in the Debugging – Log toolbar.

In the example in the slide, stepping over will execute the open cursor line without transferring program control to the cursor definition as was the case with the Step Into option example.





## Running to the Cursor Location

When stepping through your application code in the debugger, you may want to run to a particular location without having to single step or set a breakpoint. To run to a specific program location: In a subprogram editor, position your text cursor on the line of code where you want the debugger to stop. You can run to the cursor location using any of the following procedures: In the procedure editor, right-click and choose Run to Cursor, choose the Debug > Run to Cursor option from the main menu, or press F4. Any of the following conditions may result:

When you run to the cursor, your program executes without stopping, until the execution reaches the location marked by the text cursor in the source editor.

If your program never actually executes the line of code where the text cursor is, the Run to Cursor command will cause your program to run until it encounters a breakpoint or until it finishes.

Debugging emp\_list: Step to End of Method

```
17 BEGIN
18   FETCH emp_cursor INTO emp_record;
19   emp_tab(i) := emp_record;
20   WHILE (emp_cursor%FOUND) AND (i <= pmaxrows) LOOP
21     i := i + 1;
22     FETCH emp_cursor INTO emp_record;
23     emp_tab(i) := emp_record;
24     v_city := get_location(emp_record.department_name,
25                           emp_record.last_name);
26     dbms_output.put_line('Employee ' || emp_record.last_name ||
27                           ' works in ' || v_city);
28   END LOOP;
29   CLOSE emp_cursor;
30   FOR j IN REVERSE 1..i LOOP
31     DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
32   END LOOP;
33 END emp_list;
```

```
17 BEGIN
18   FETCH emp_cursor INTO emp_record;
19   emp_tab(i) := emp_record;
20   WHILE (emp_cursor%FOUND) AND (i <= pmaxrows) LOOP
21     i := i + 1;
22     FETCH emp_cursor INTO emp_record;
23     emp_tab(i) := emp_record;
24     v_city := get_location(emp_record.department_name,
25                           emp_record.last_name);
26     dbms_output.put_line('Employee ' || emp_record.last_name ||
27                           ' works in ' || v_city);
28   END LOOP;
29   CLOSE emp_cursor;
30   FOR j IN REVERSE 1..i LOOP
31     DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
32   END LOOP;
33 END emp_list2;
```

Loops until i <= PMAXROWS

```
1 DECLARE
2   PMAXROWS NUMBER;
3 BEGIN
4   PMAXROWS := 100;
5 END;
```

ORACLE

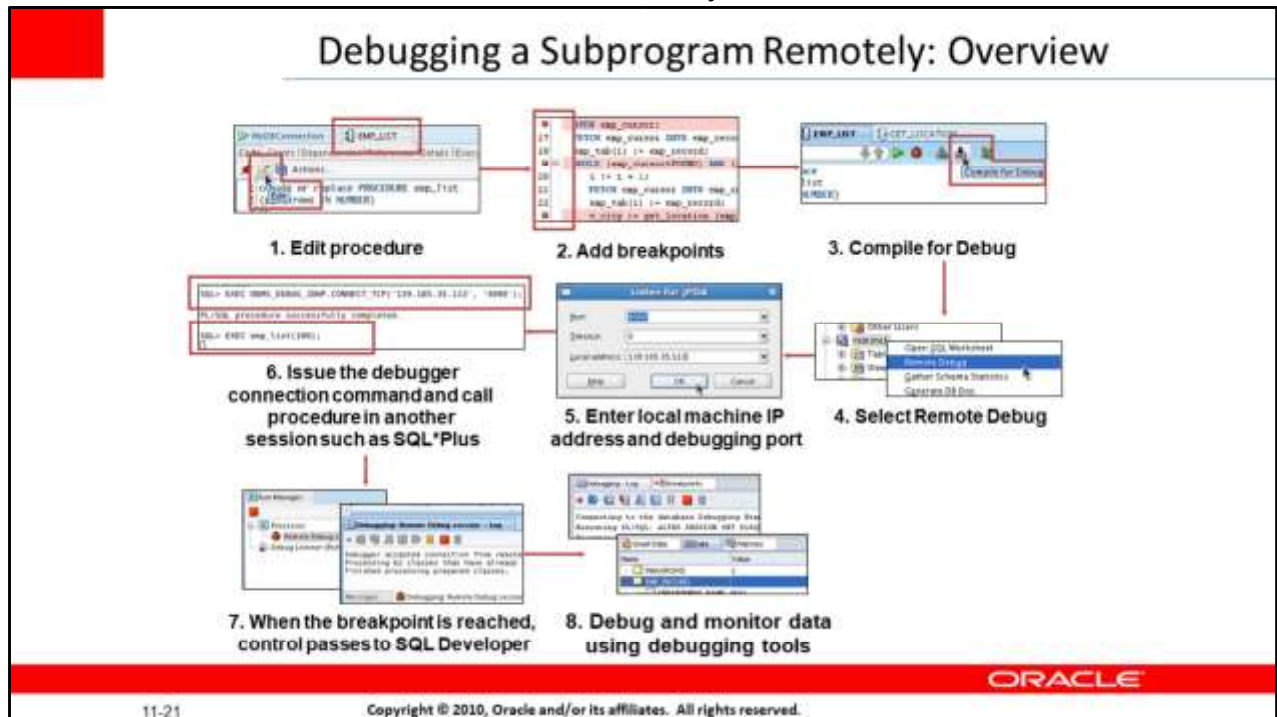
11-20 Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

## Debugging emp\_list: Step to End of Method

Step to End of Method goes to the last statement in the current subprogram or to the next breakpoint if there are any in the current subprogram. In the example in the slide, the Step to End of Method debugging tool is used.

Because there is a second breakpoint, selecting Step to End of Method transfers control to that breakpoint. Selecting Step to End of Method again goes through

the iterations of the while loop first, and then transfers the program control to the next executable statement in the anonymous block.



## Remote Debugging

You can use remote debugging to connect to any PL/SQL code in a database and debug it, regardless of where the database is located as long as you have access to the database and have created a connection to it. Remote debugging is when something else, other than you as a developer, kicks off a procedure that you can then debug. Remote debugging and local debugging have many steps in common. To debug remotely, perform the following steps:

1. Edit the subprogram that you would like to debug.
2. Add the breakpoints in the subprogram.
3. Click the Compile for Debug icon in the toolbar.
4. Right-click the connection for the remote database, and select Remote Debug.
5. In the Debugger - Attach to JPDA dialog box, enter the local machine IP address and port number, such as 4000, and then click OK.
6. Issue the debugger connection command using a different session such as SQL\*Plus, and then call the procedure in that session.
7. When a breakpoint is reached, control is passed back to

the original SQL Developer session and the debugger toolbar is displayed.

8. Debug the subprogram and monitor the data using the debugging tools discussed earlier.

