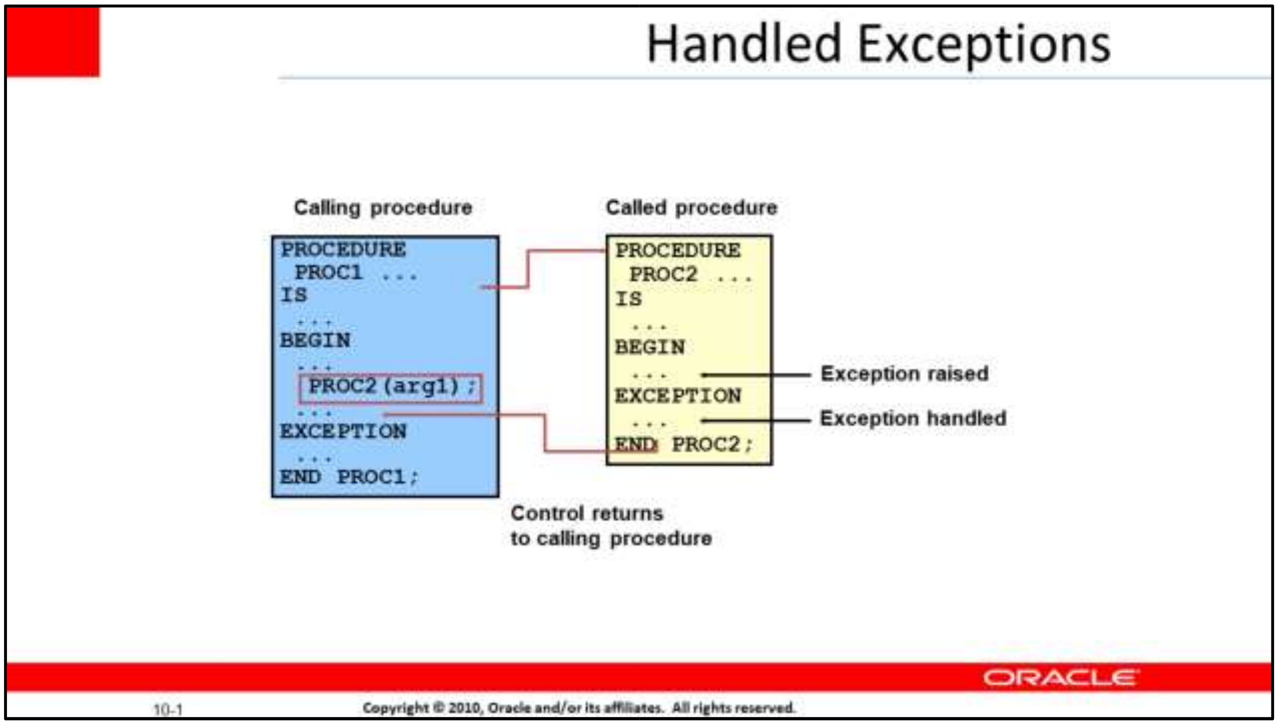


Handled Exceptions

```
PROCEDURE
 PROC1 ...
IS
 ...
BEGIN
 ...
  PROC2(arg1);
 ...
EXCEPTION
 ...
END PROC1;
```

```
PROCEDURE
 PROC2 ...
IS
 ...
BEGIN
 ...
EXCEPTION
 ...
END PROC2;
```

Handled Exceptions

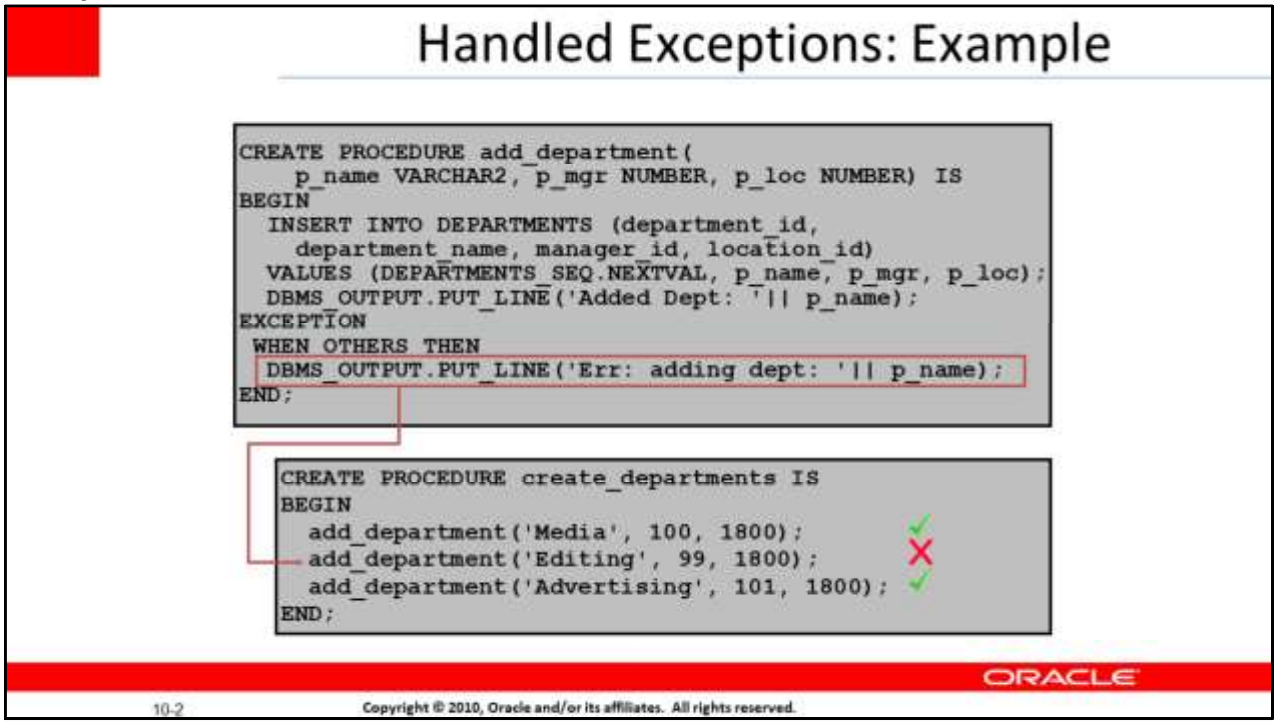
When you develop procedures that are called from other procedures, you should be aware of the effects that handled and unhandled exceptions have on the transaction and the calling procedure.

When an exception is raised in a called procedure, the control immediately goes to the exception section of that block. An exception is considered handled if the exception section provides a handler for the exception raised.

When an exception occurs and is handled, the following code flow takes place:

1. The exception is raised.
2. Control is transferred to the exception handler.
3. The block is terminated.
4. The calling program/block continues to execute as if nothing has happened.

If a transaction was started (that is, if any data manipulation language [DML] statements executed before executing the procedure in which the exception was raised), then the transaction is unaffected. A DML operation is rolled back if it was performed within the procedure before the exception. Note: You can explicitly end a transaction by executing a COMMIT or ROLLBACK operation in the exception section.



Handled Exceptions: Example

```
CREATE PROCEDURE add_department(
    p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS
BEGIN
  INSERT INTO DEPARTMENTS (department_id,
    department_name, manager_id, location_id)
  VALUES (DEPARTMENTS_SEQ.NEXTVAL, p_name, p_mgr, p_loc);
  DBMS_OUTPUT.PUT_LINE('Added Dept: '|| p_name);
EXCEPTION
 WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE('Err: adding dept: '|| p_name);
END;
```

```
CREATE PROCEDURE create_departments IS
BEGIN
  add_department('Media', 100, 1800);          ✓
  add_department('Editing', 99, 1800);         ✗
  add_department('Advertising', 101, 1800);    ✓
END;
```

ORACLE

10-2

Handled Exceptions: Example

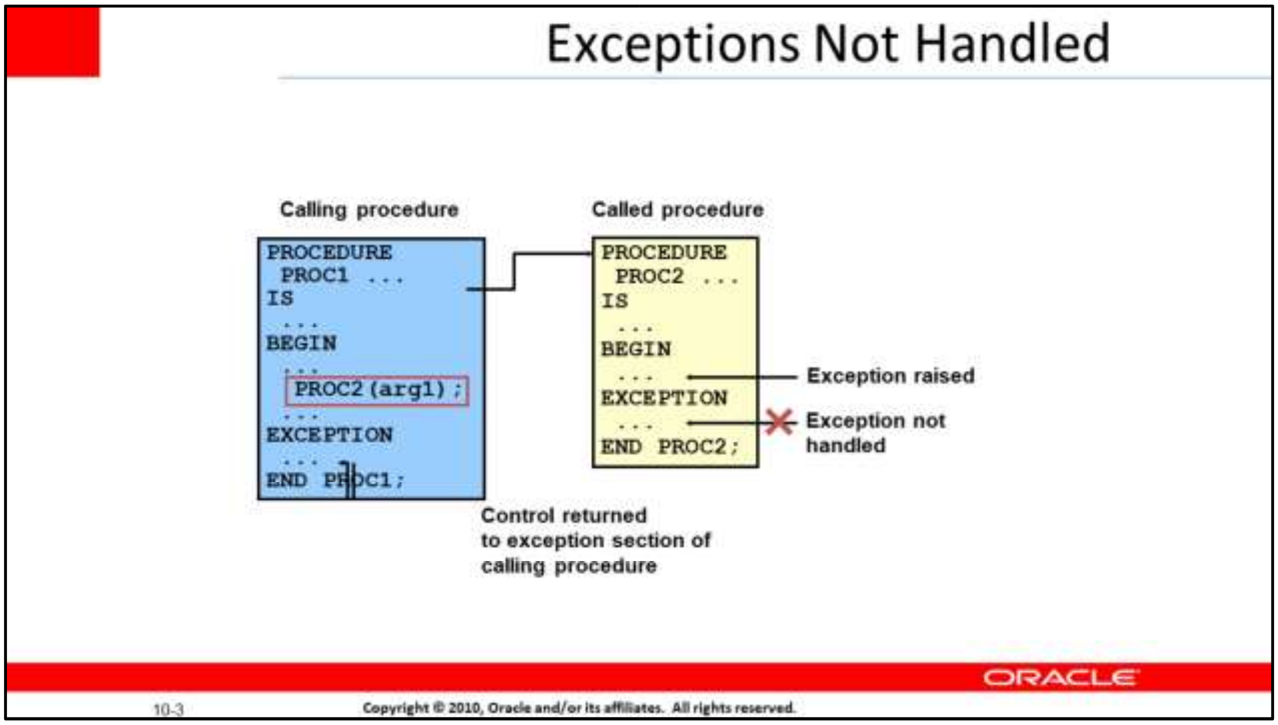
The two procedures in the slide are the following:

The add_department procedure creates a new department record by allocating a new department number from an Oracle sequence, and sets the department_name, manager_id, and location_id column values using the p_name, p_mgr, and p_loc parameters, respectively.

The create_departments procedure creates more than one department by using calls to the add_department procedure. The add_department procedure catches all raised exceptions in its own handler. When create_departments is executed, the following output is generated:

```
Added Dept: Media
Err: adding dept: Editing
Added Dept: Advertising
```

The Editing department with a manager_id of 99 is not inserted because a foreign key integrity constraint violation on manager_id ensures that no manager has an ID of 99. Because the exception was handled in the add_department procedure, the create_department procedure continues to execute. A query on the DEPARTMENTS table where the location_id is 1800 shows that Media and Advertising are added but the Editing record is not.



Exceptions Not Handled

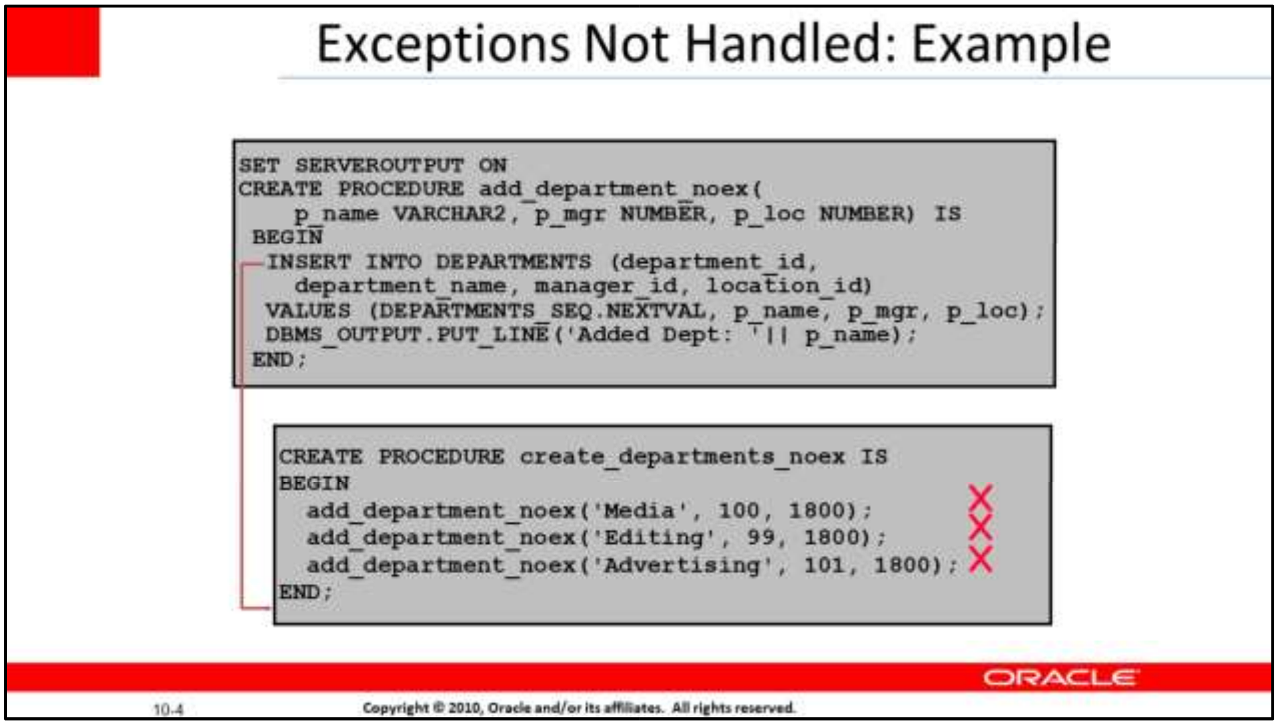
Calling procedure
Called procedure
PROCEDURE PROC1 ... IS ... BEGIN ... PROC2(arg1); ... EXCEPTION ... END PROC1;
PROCEDURE PROC2 ... IS ... BEGIN ... EXCEPTION ... END PROC2;
Exception raised
Exception not handled
Control returned to exception section of calling procedure
ORACLE

Exceptions Not Handled

As discussed earlier, when an exception is raised in a called procedure, control immediately goes to the exception section of that block. If the exception section does not provide a handler for the raised exception, then it is not handled. The following code flow occurs:

1. The exception is raised.
2. The block terminates because no exception handler exists; any DML operations performed within the procedure are rolled back.
3. The exception propagates to the exception section of the calling procedure—that is, control is returned to the exception section of the calling block, if one exists.

If an exception is not handled, then all the DML statements in the calling procedure and the called procedure are rolled back along with any changes to any host variables. The DML statements that are not affected are statements that were executed before calling the PL/SQL code whose exceptions are not handled.



Exceptions Not Handled: Example

```
SET SERVEROUTPUT ON
CREATE PROCEDURE add_department_noex(
    p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS
 BEGIN
  INSERT INTO DEPARTMENTS (department_id,
    department_name, manager_id, location_id)
  VALUES (DEPARTMENTS_SEQ.NEXTVAL, p_name, p_mgr, p_loc);
  DBMS_OUTPUT.PUT_LINE('Added Dept: '|| p_name);
 END;
```

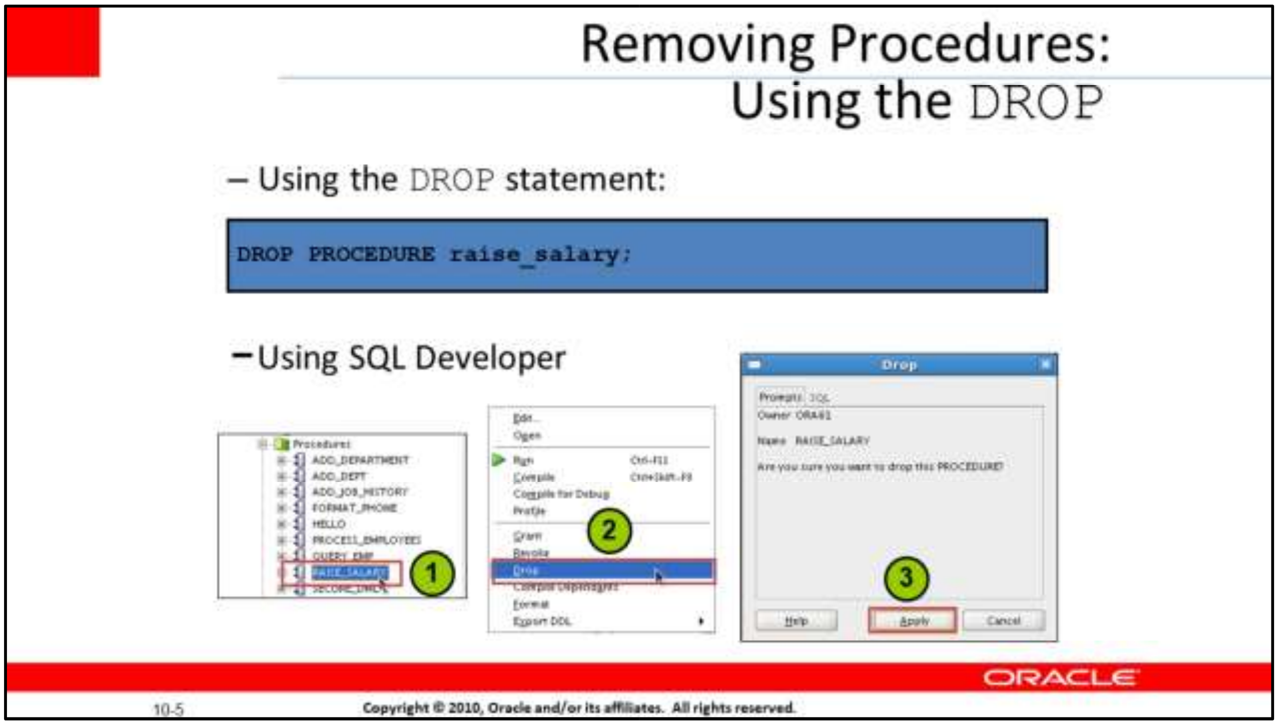
```
CREATE PROCEDURE create_departments_noex IS
BEGIN
  add_department_noex('Media', 100, 1800);
  add_department_noex('Editing', 99, 1800);
  add_department_noex('Advertising', 101, 1800);
END;
```

10-4 Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Exceptions Not Handled: Example

The code example in the slide shows add_department_noex, which does not have an exception section. In this case, the exception occurs when the Editing department is added. Because of the lack of exception handling in either of the subprograms, no new department records are added into the DEPARTMENTS table. Executing the create_departments_noex procedure produces a result that is similar to the following:

```
Error starting at line 8 in command:
EXECUTE create_departments_noex
Error report:
ORA-02291: integrity constraint (ORA62.DEPT_MGR_FK) violated - parent key not found
ORA-06512: at "ORA62.ADD_DEPARTMENT_NOEX", line 4
ORA-06512: at "ORA62.CREATE_DEPARTMENTS_NOEX", line 4
ORA-06512: at line 1
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"
*Cause:    A foreign key value has no matching primary key value.
*Action:   Delete the foreign key or add a matching primary key.
```


Removing Procedures: Using the DROP

– Using the DROP statement:

```
DROP PROCEDURE raise_salary;
```

–Using SQL Developer

Procedures
ADD_DEPARTMENT
ADD_DEPT
ADD_JOB_HISTORY
FORMAT_PHONE
HELLO
PROCESS_EMPLOYEES
QUERY_EMP
RAISE_SALARY
SECURE_DML
1
Edit...
Open
Run Ctrl-F11
Compile Ctrl+Shift-F9
Compile for Debug
Profile
Grant
Revoke
Drop
Compile Dependants
Format
Export DDL
2
Drop
Prompts SQL
Owner ORA61
Name RAISE_SALARY
Are you sure you want to drop this PROCEDURE?
3
Help
Apply
Cancel
ORACLE

Removing Procedures: Using the DROP SQL Statement or SQL Developer

When a stored procedure is no longer required, you can use the DROP PROCEDURE SQL statement followed by the procedure's name to remove it as follows:

DROP PROCEDURE procedure_name

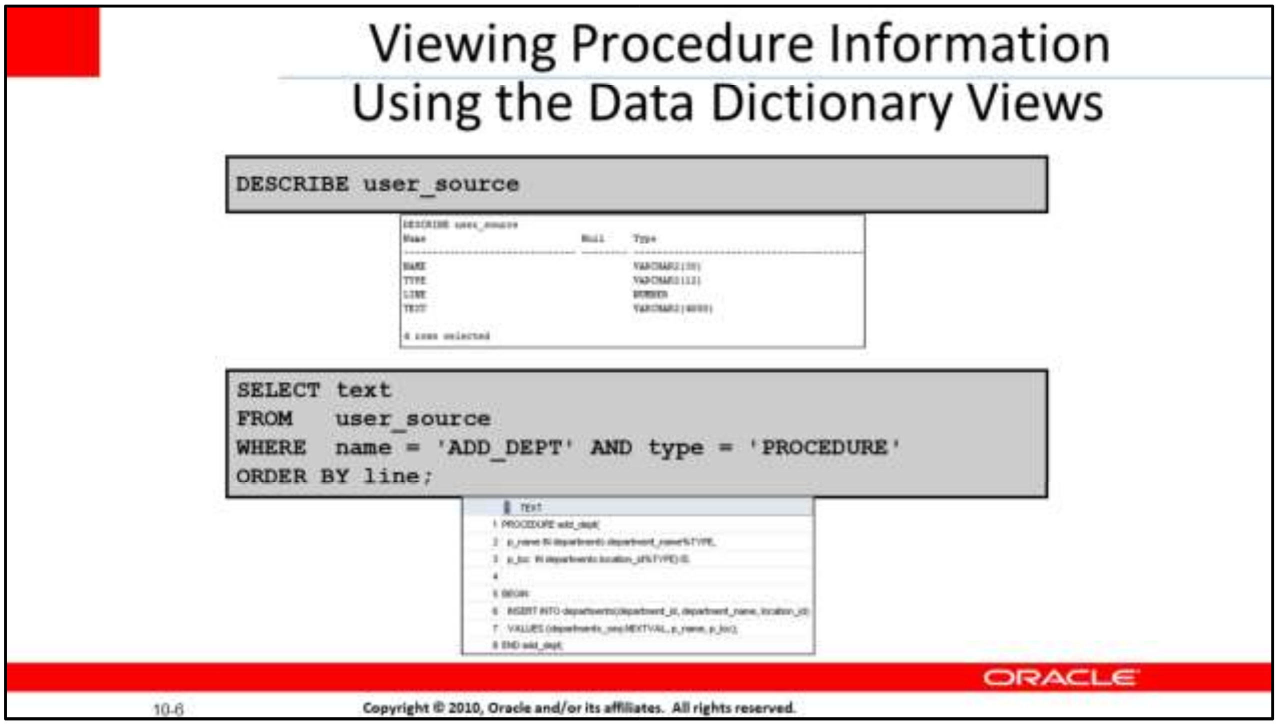
You can also use SQL Developer to drop a stored procedure as follows: 1. Right-click the procedure name in the Procedures node, and then click Drop. The Drop dialog box is displayed.

2. Click Apply to drop the procedure.

Note

Whether successful or not, executing a data definition language (DDL) command such as DROPPROCEDURE commits any pending transactions that cannot be rolled back.

You might have to refresh the Procedures node before you can see the results of the drop operation. To refresh the Procedures node, rightclick the procedure name in the Procedures node, and then click Refresh.



Viewing Procedure Information Using the Data Dictionary Views

```
DESCRIBE user_source
```

```
DESCRIBE user_source
Name                          Null     Type
----------------------------- -------- ---------------------
NAME                                   VARCHAR2(30)
TYPE                                   VARCHAR2(12)
LINE                                   NUMBER
TEXT                                   VARCHAR2(4000)

4 rows selected
```

```
SELECT text
FROM   user_source
WHERE  name = 'ADD_DEPT' AND type = 'PROCEDURE'
ORDER BY line;
```

```
  TEXT
1 PROCEDURE add_dept(
2   p_name IN departments.department_name%TYPE,
3   p_loc  IN departments.location_id%TYPE) IS
4
5 BEGIN
6   INSERT INTO departments(department_id, department_name, location_id)
7   VALUES (departments_seq.NEXTVAL, p_name, p_loc);
8 END add_dept;
```

10-6

Viewing Procedure in the Data Dictionary

The source code for PL/SQL subprograms is stored in the data dictionary tables. The source code is accessible to PL/SQL procedures that are successfully or unsuccessfully compiled. To view the PL/SQL source code stored in the data dictionary, execute a SELECT statement on the following tables:

The USER_SOURCE table to display PL/SQL code that you own The ALL_SOURCE table to display PL/SQL code to which you have been granted the EXECUTE right by the owner of that subprogram code The query example shows all the columns provided by the USER_SOURCE table:

The TEXT column holds a line of PL/SQL source code.

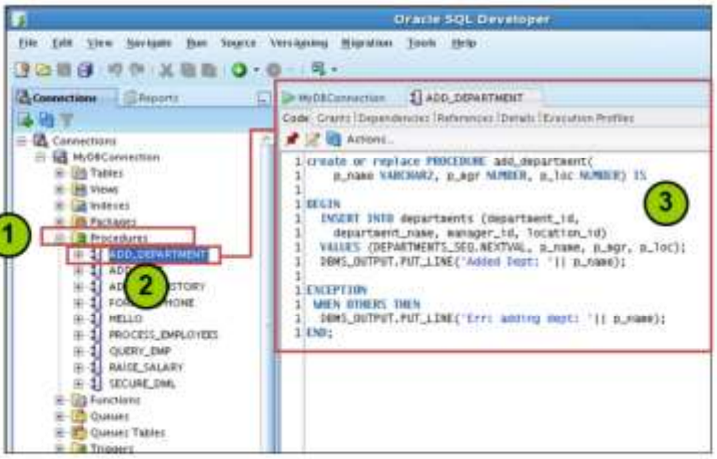
The NAME column holds the name of the subprogram in uppercase text. The TYPE column holds the subprogram type, such as PROCEDURE or

FUNCTION.

The LINE column stores the line number for each source code line. The ALL_SOURCE table provides an OWNER column in addition to the preceding columns.

Note: You cannot display the source code for Oracle PL/SQL built-in packages, or PL/SQL whose source code has been wrapped by using a WRAP utility. The WRAP utility converts the PL/SQL source code into a form that cannot be deciphered by humans.

Viewing Procedures Information Using the SQL Developer



```
create or replace PROCEDURE add_department(
    p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS

BEGIN
  INSERT INTO departments (department_id,
    department_name, manager_id, location_id)
  VALUES (DEPARTMENTS_SEQ.NEXTVAL, p_name, p_mgr, p_loc);
  DBMS_OUTPUT.PUT_LINE('Added Dept: '|| p_name);

EXCEPTION
 WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE('Err: adding dept: '|| p_name);
END;
```

Viewing Procedures Information Using SQL Developer

To view a procedure’s code in SQL Developer, use the following steps:

1. Click the Procedures node in the Connections tab.
2. Click the procedure’s name.
3. The procedure's code is displayed in the Code tab as shown in the slide.