Advanced PLSQL

Lesson 06: Best Practices of PLSQL and Dynamic SQL

Lesson Objectives

- In this lesson you will learn
- Best Practices of PLSQL code
  - Conditional Compilation
  - Using Selection Directives
  - Obfuscation
- Dynamic SQL

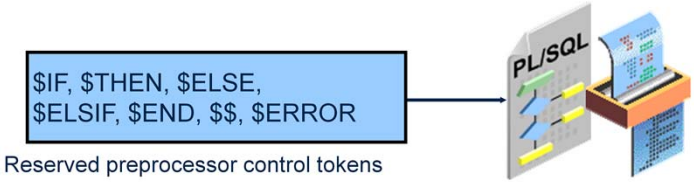
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2

6.1.1: Best practices: Conditional compilation

Conditional Compilation

- Enables you to customize the functionality in a PL/SQL application without removing any source code:
  - Utilize the latest functionality with the latest database release or disable the new features to run the application against an older release of the database.
  - Activate debugging or tracing functionality in the development environment and hide that functionality in the application while it runs at a production site.

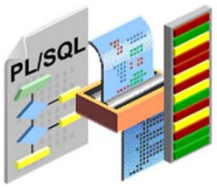


Reserved preprocessor control tokens

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

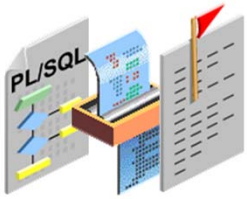
 3

How Does Conditional Compilation Work?



Selection directives:
Use the $IF token.

Inquiry directives:
Use the $$ token.

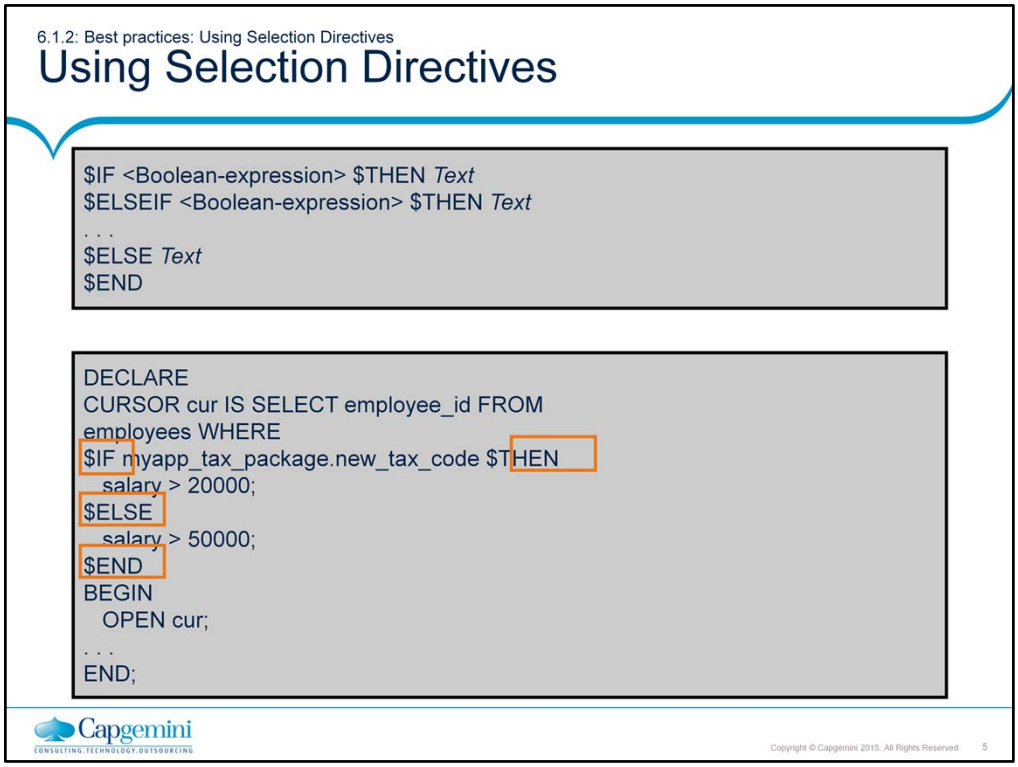


Error directives:
Use the $ERROR token.

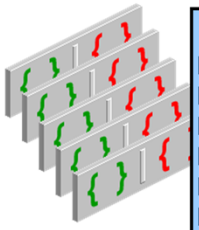
DBMS_PREPROCESSOR
package

DBMS_DB_VERSION
package

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING


6.1.2: Best practices: Using Selection Directives
Using Selection Directives
$IF <Boolean-expression> $THEN Text
$ELSEIF <Boolean-expression> $THEN Text
. . .
$ELSE Text
$END
DECLARE
CURSOR cur IS SELECT employee_id FROM
employees WHERE
$IF myapp_tax_package.new_tax_code $THEN
salary > 20000;
$ELSE
salary > 50000;
$END
BEGIN
OPEN cur;
. . .
END;
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved 5

Using Predefined and User-Defined Inquiry Directives



PLSQL_CCFLAGS
PLSQL_CODE_TYPE
PLSQL_OPTIMIZE_LEVEL
PLSQL_WARNINGS
NLS_LENGTH_SEMANTICS
PLSQL_LINE
PLSQL_UNIT

Predefined inquiry directives

```
PLSQL_CCFLAGS = 'plsql_ccflags:true,debug:true,debug:0';
```

User-defined inquiry directives

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

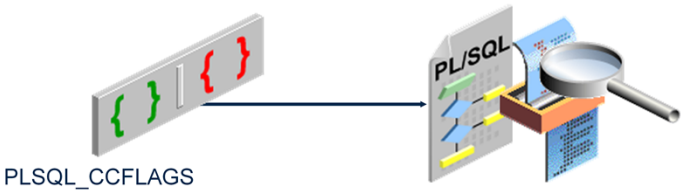
 6

The PLSQL_CCFLAGS Parameter and the Inquiry Directive

- Use the PLSQL_CCFLAGS parameter to control conditional compilation of each PL/SQL library unit independently.

```
PLSQL_CCFLAGS = '<v1>:<c1>,<v2>:<c2>,...,<vn>:<cn>'
```

```
ALTER SESSION SET
PLSQL_CCFLAGS = 'plsql_ccflags:true, debug:true, debug:0';
```



PLSQL_CCFLAGS initialization parameter

Inquiry directive

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 7

Displaying the PLSQL_CCFLAGS Initialization Parameter Setting

```
SELECT name, type, plsql_ccflags
FROM   user_plsql_object_settings
```

Results:

	NAME	TYPE	PLSQL_CCFLAGS
1	DEPT_PKG	PACKAGE	(null)
2	DEPT_PKG	PACKAGE BODY	(null)
3	TAXES_PKG	PACKAGE	(null)
4	TAXES_PKG	PACKAGE BODY	(null)
5	EMP_PKG	PACKAGE	(null)
6	EMP_PKG	PACKAGE BODY	(null)
7	SECURE_DML	PROCEDURE	(null)
8	SECURE_EMPLOYEES	TRIGGER	(null)
9	ADD_JOB_HISTORY	PROCEDURE	plsql_ccflags:true, debug:true, debug:0
10	UPDATE_JOB_HISTORY	TRIGGER	(null)

. . .

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8

The PLSQL_CCFLAGS Parameter and the Inquiry Directive: Example

```
ALTER SESSION SET PLSQL_CCFLAGS = 'Tracing:true';
CREATE OR REPLACE PROCEDURE P IS
BEGIN
  $IF $$tracing $THEN
    DBMS_OUTPUT.PUT_LINE ('TRACING');
  $END
END P;
```

```
ALTER SESSION SET succeeded.
PROCEDURE P Compiled.
```

```
SELECT name, plsql_ccflags
FROM USER_PLSQL_OBJECT_SETTINGS
WHERE name = 'P';
```

Results:

	NAME	PLSQL_CCFLAGS
1	P	Tracing:true

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 9

Using Conditional Compilation Error Directives to Raise User-Defined Errors

```
$ERROR varchar2_static_expression $END
```

```
ALTER SESSION SET Plsql_CCFlags = ' Trace_Level:3 '
/ CREATE PROCEDURE P IS
BEGIN
  $IF    $$Trace_Level = 0 $THEN ...;
  $ELSIF $$Trace_Level = 1 $THEN ...;
  $ELSIF $$Trace_Level = 2 $THEN ...;
  $else $error 'Bad: '||$$Trace_Level $END –- error
                        -- directive
  $END -- selection directive ends
END P;
```

```
SHOW ERRORS
Errors for PROCEDURE P:
LINE/COL ERROR
-------- -----------------------------------
6/9      PLS-00179: $ERROR: Bad: 3
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10

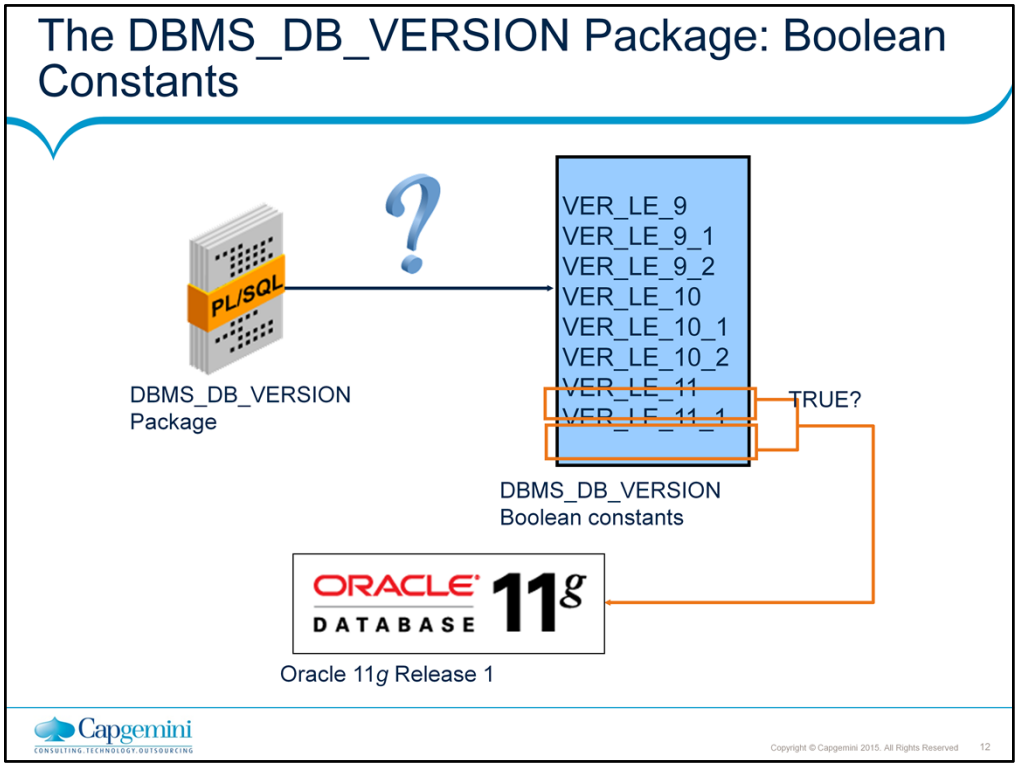
Using Static Expressions with Conditional Compilation

- Boolean static expressions:
- TRUE, FALSE, NULL, IS NULL, IS NOT NULL
- > , < , >= , <= , = , <>, NOT, AND, OR
- PLS_INTEGER static expressions:
- -2147483648 to 2147483647, NULL
- VARCHAR2 static expressions include:
- ||, NULL, TO_CHAR
- Static constants:

```
static_constant CONSTANT datatype := static_expression;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11


The DBMS_DB_VERSION Package: Boolean Constants
PL/SQL
DBMS_DB_VERSION
Package
VER_LE_9
VER_LE_9_1
VER_LE_9_2
VER_LE_10
VER_LE_10_1
VER_LE_10_2
VER_LE_11
TRUE?
DBMS_DB_VERSION
Boolean constants
ORACLE DATABASE 11g
Oracle 11g Release 1
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
12

The DBMS_DB_VERSION Package Constants

Name	Value	Description
VER_LE_9	FALSE	Version <= 9.
VER_LE_9_1	FALSE	Version <= 9 and release <= 1.
VER_LE_9_2	FALSE	Version <= 9 and release <= 2.
VER_LE_10	TRUE	Version <= 10.
VER_LE_10_1	FALSE	Version <= 10 and release <= 1.
VER_LE_10_2	TRUE	Version <=10 and release <= 2.
VER_LE_11	FALSE	Version <= 11.
VER_LE_11_1	TRUE	Version <=11 and release <= 1.

Using Conditional Compilation with Database Versions: Example

```
ALTER SESSION SET PLSQL_CCFLAGS = 'my_debug:FALSE, my_tracing:FALSE';
CREATE PACKAGE my_pkg AS
  SUBTYPE my_real IS
  -- Check the database version, if >= 10g, use BINARY_DOUBLE data type,
  -- else use NUMBER data type
    $IF DBMS_DB_VERSION.VERSION < 10 $THEN     NUMBER;
    $ELSE BINARY_DOUBLE;
    $END
  my_pi my_real; my_e my_real;
END my_pkg;
/
CREATE PACKAGE BODY my_pkg AS
BEGIN
  $IF DBMS_DB_VERSION.VERSION < 10 $THEN
      my_pi := 3.14016408289008292431940027343666863227;
      my_e  := 2.71828182845904523536028747135266249775;
  $ELSE
      my_pi := 3.14016408289008292431940027343666863227d;
      my_e  := 2.71828182845904523536028747135266249775d;
  $END
END my_pkg;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 14

Using Conditional Compilation with Database Versions: Example

```
CREATE OR REPLACE PROCEDURE circle_area(p_radius my_pkg.my_real) IS
  v_my_area my_pkg.my_real;
  v_my_datatype VARCHAR2(30);
BEGIN
  v_my_area := my_pkg.my_pi * p_radius * p_radius;
  DBMS_OUTPUT.PUT_LINE('Radius: ' || TO_CHAR(p_radius)
                  || ' Area: ' || TO_CHAR(v_my_area) );
  $IF $$my_debug $THEN  -- if my_debug is TRUE, run some debugging code
    SELECT DATA_TYPE INTO v_my_datatype FROM USER_ARGUMENTS
      WHERE OBJECT_NAME = 'CIRCLE_AREA' AND ARGUMENT_NAME = 'P_RADIUS';
    DBMS_OUTPUT.PUT_LINE('Datatype of the RADIUS argument is: ' || v_my_datatype);
  $END
END; /
```

```
PROCEDURE circle_area(p_radius Compiled.
```

```
CALL circle_area(50); -- Using Oracle Database 11g Release 2
```

```
CALL circle_area(50) succeeded.
Radius: 5.0E+001 Area: 7.8504102072252062E+003
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 15

Using DBMS_PREPROCESSOR Procedures to Print or Retrieve Source Text

```
CALL
    DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE('PACK
    AGE', 'ORA61', 'MY_PKG');
```

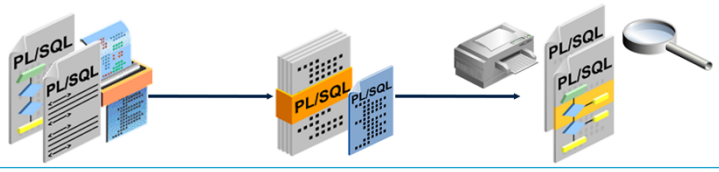
Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

```
CALL DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE('PACKAGE', succeeded.
PACKAGE my_pkg AS
  SUBTYPE my_real IS

-- Check the database version, if >= 10g, use BINARY_DOUBLE data type,
-- else use NUMBER data type


                BINARY_DOUBLE;

  my_pi my_real; my_e my_real;
END my_pkg;
```



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 16

6.1.3: Best practices: Using Selection Directives

Obfuscation

- Obfuscation (or wrapping) of a PL/SQL unit is the process of hiding the PL/SQL source code.
- Wrapping can be done with the wrap utility and `DBMS_DDL` subprograms.
- The Wrap utility is run from the command line and it processes an input SQL file, such as a SQL*Plus installation script.
- The `DBMS_DDL` subprograms wrap a single PL/SQL unit, such as a single `CREATE PROCEDURE` command, that has been generated dynamically.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 17

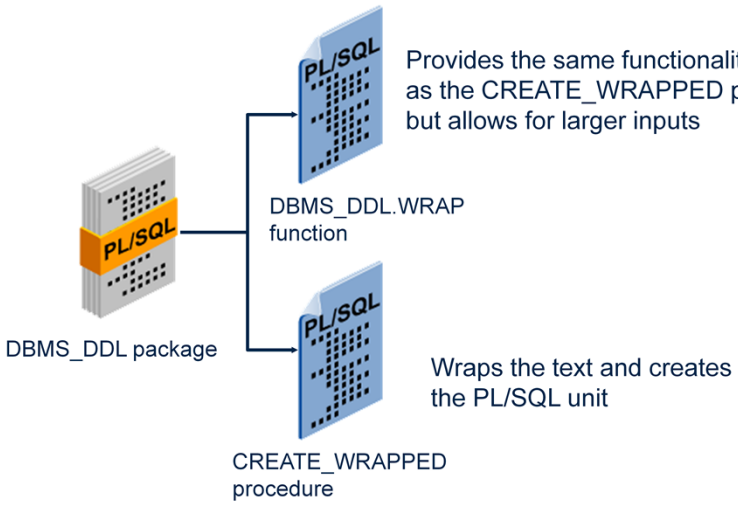
Benefits of Obfuscating

- It prevents others from seeing your source code.
- Your source code is not visible through the USER_SOURCE, ALL_SOURCE, or DBA_SOURCE data dictionary views.
- SQL*Plus can process the obfuscated source files.
- The Import and Export utilities accept wrapped files.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 18

What's New in Dynamic Obfuscating Since Oracle 10*g*?



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 19

Nonobfuscated PL/SQL Code: Example

```
SET SERVEROUTPUT ON
BEGIN -- The ALL_SOURCE view family shows source code
  EXECUTE IMMEDIATE '
    CREATE OR REPLACE PROCEDURE P1 IS
    BEGIN
      DBMS_OUTPUT.PUT_LINE ("I am not wrapped");
    END P1;
  ';
END;
/
CALL p1();
```

```
anonymous block completed
CALL p1() succeeded.
I'm not wrapped
```

```
SELECT text FROM user_source
WHERE name = 'P1' ORDER BY line;
```

	TEXT
1	PROCEDURE P1 IS
2	BEGIN
3	DBMS_OUTPUT.PUT_LINE ('I am not wrapped');
4	END P1;

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 20

Obfuscated PL/SQL Code: Example

```
BEGIN -- ALL_SOURCE view family obfuscates source code
  DBMS_DDL.CREATE_WRAPPED ( '
    CREATE OR REPLACE PROCEDURE P1 IS
    BEGIN
      DBMS_OUTPUT.PUT_LINE ("I am wrapped now");
    END P1;
  ' );
END;
/
CALL p1();
```

```
anonymous block completed
call p1() succeeded.
I am wrapped now
```

```
SELECT text FROM user_source
WHERE name = 'P1' ORDER BY line;
```

```
TEXT
--------------------------------------------------------------------------------
PROCEDURE P1 wrapped
a000000
b2
abcd
. . .
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 21

Dynamic Obfuscation: Example

```
SET SERVEROUTPUT ON

DECLARE
c_code CONSTANT VARCHAR2(32767) :=
' CREATE OR REPLACE PROCEDURE new_proc AS
  v_VDATE  DATE;
  BEGIN
    v_VDATE := SYSDATE;
    DBMS_OUTPUT.PUT_LINE(v_VDATE) ;
  END; ' ;
BEGIN
  DBMS_DDL.CREATE_WRAPPED (c_CODE);
END;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 22

The PL/SQL Wrapper Utility

- The PL/SQL wrapper is a stand-alone utility that hides application internals by converting PL/SQL source code into portable object code.
- Wrapping has the following features:
  - Platform independence
  - Dynamic loading
  - Dynamic binding
  - Dependency checking
  - Normal importing and exporting when invoked

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 23

Running the Wrapper Utility

```
WRAP INAME=input_file_name [ONAME=output_file_name]
```

- Do not use spaces around the equal signs.
- The INAME argument is required.
- The default extension for the input file is .sql, unless it is specified with the name.
- The ONAME argument is optional.
- The default extension for output file is .plb, unless specified with the ONAME argument.

Examples

```
WRAP INAME=demo_04_hello.sql
WRAP INAME=demo_04_hello
WRAP INAME=demo_04_hello.sql ONAME=demo_04_hello.plb
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Results of Wrapping

```
-- Original PL/SQL source code in input file:

CREATE PACKAGE banking IS
   min_bal := 100;
   no_funds EXCEPTION;
      ...
      END banking;
      /
```

```
-- Wrapped code in output file:

CREATE PACKAGE banking
      wrapped

012abc463e ...

/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 25

DBMS_DDL Package Versus the Wrap Utility

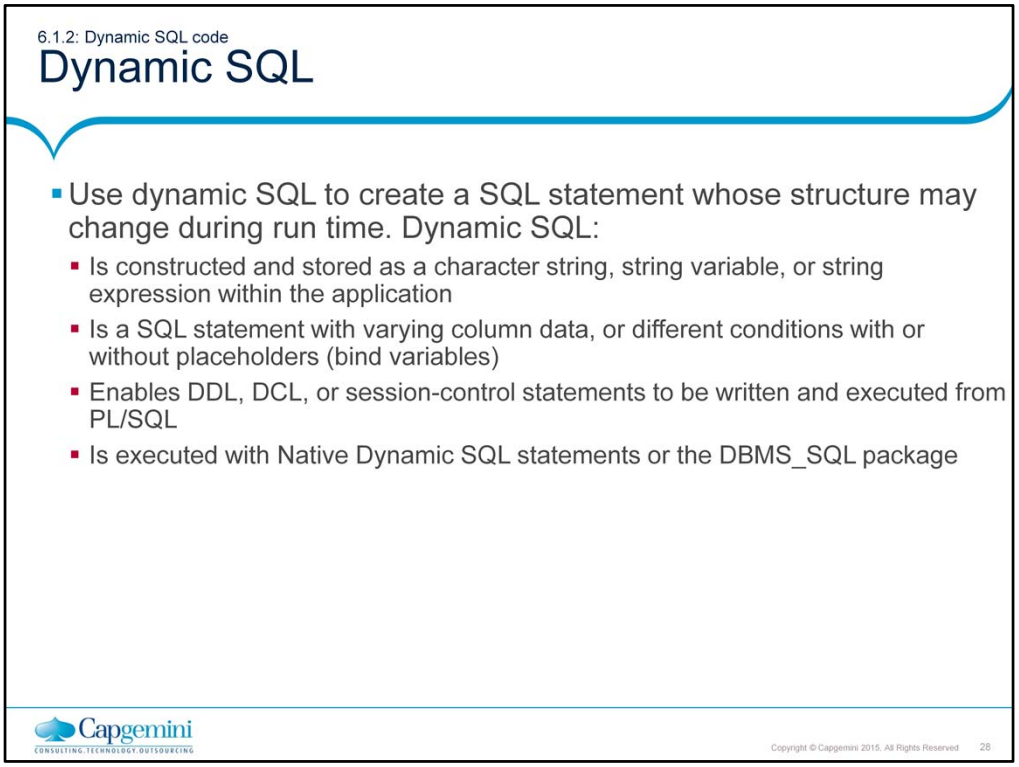
Functionality	DBMS_DDL	Wrap Utility
Code obfuscation	Yes	Yes
Dynamic Obfuscation	Yes	No
Obfuscate multiple programs at a time	No	Yes

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 26

Guidelines for Wrapping

- You must wrap only the package body, not the package specification.
- The wrapper can detect syntactic errors but cannot detect semantic errors.
- The output file should not be edited. You maintain the original source code and wrap again as required.
- To ensure that all the important parts of your source code are obfuscated, view the wrapped file in a text editor before distributing it.


6.1.2: Dynamic SQL code
Dynamic SQL
Use dynamic SQL to create a SQL statement whose structure may change during run time. Dynamic SQL:
Is constructed and stored as a character string, string variable, or string expression within the application
Is a SQL statement with varying column data, or different conditions with or without placeholders (bind variables)
Enables DDL, DCL, or session-control statements to be written and executed from PL/SQL
Is executed with Native Dynamic SQL statements or the DBMS_SQL package
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
28

Using Dynamic SQL

- Use dynamic SQL when the full text of the dynamic SQL statement is unknown until run time; therefore, its syntax is checked at *run time* rather than at *compile time*.
- Use dynamic SQL when one of the following items is unknown at precompile time:
  - Text of the SQL statement such as commands, clauses, and so on
  - The number and data types of host variables
  - References to database objects such as tables, columns, indexes, sequences, usernames, and views
- Use dynamic SQL to make your PL/SQL programs more general and flexible.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 29

Native Dynamic SQL (NDS)

- Provides native support for dynamic SQL directly in the PL/SQL language.
- Provides the ability to execute SQL statements whose structure is unknown until execution time.
- If the dynamic SQL statement is a SELECT statement that returns multiple rows, NDS gives you the following choices:
- Use the EXECUTE IMMEDIATE statement with the BULK COLLECT INTO clause
- Use the OPEN-FOR, FETCH, and CLOSE statements
- In Oracle Database 11g, NDS supports statements larger than 32 KB by accepting a CLOB argument.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 30

Using the EXECUTE IMMEDIATE Statement

- Use the EXECUTE IMMEDIATE statement for NDS or PL/SQL anonymous blocks:

```
EXECUTE IMMEDIATE dynamic_string
 [INTO {define_variable
    [, define_variable] ... | record}]
 [USING [IN|OUT|IN OUT] bind_argument
    [, [IN|OUT|IN OUT] bind_argument] ... ];
```

- INTO is used for single-row queries and specifies the variables or records into which column values are retrieved.
- USING is used to hold all bind arguments. The default parameter mode is IN.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 31

Available Methods for Using NDS

Method #	SQL Statement Type	NDS SQL Statements Used
Method 1	***Non-query without*** host variables	EXECUTE IMMEDIATE without the USING and INTO clauses
Method 2	***Non-query with*** known number of input host variables	EXECUTE IMMEDIATE with a USING clause
Method 3	***Query with known*** number of select-list items and input host variables	EXECUTE IMMEDIATE with the USING and INTO clauses
Method 4	***Query with unknown*** number of select-list items or input host variables	Use the DBMS_SQL package

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 32

Dynamic SQL with a DDL Statement: Examples

```
-- Create a table using dynamic SQL

CREATE OR REPLACE PROCEDURE create_table(
  p_table_name VARCHAR2, p_col_specs  VARCHAR2) IS
BEGIN
  EXECUTE IMMEDIATE 'CREATE TABLE ' || p_table_name ||
                    ' (' || p_col_specs || ')';
END;
/
```

```
-- Call the procedure

BEGIN
  create_table('EMPLOYEE_NAMES',
      'id NUMBER(4) PRIMARY KEY, name VARCHAR2(40)');
END;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Dynamic SQL with DML Statements

```
-- Delete rows from any table:
CREATE FUNCTION del_rows(p_table_name VARCHAR2)
RETURN NUMBER IS
BEGIN
  EXECUTE IMMEDIATE 'DELETE FROM '|| p_table_name;
  RETURN SQL%ROWCOUNT;
END;
/
BEGIN DBMS_OUTPUT.PUT_LINE(
  del_rows('EMPLOYEE_NAMES')|| ' rows deleted.');
END;
/
```

```
-- Insert a row into a table with two columns:
CREATE PROCEDURE add_row(p_table_name VARCHAR2,
   p_id NUMBER, p_name VARCHAR2) IS
BEGIN
  EXECUTE IMMEDIATE 'INSERT INTO '|| p_table_name ||
        ' VALUES (:1, :2)' USING p_id, p_name;
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 34

Dynamic SQL with a Single-Row Query: Example

```
CREATE FUNCTION get_emp( p_emp_id NUMBER )
RETURN employees%ROWTYPE IS
  v_stmt VARCHAR2(200);
  v_emprec employees%ROWTYPE;
BEGIN
  v_stmt := 'SELECT * FROM employees ' ||
            'WHERE employee_id = :p_emp_id';
  EXECUTE IMMEDIATE v_stmt INTO v_emprec USING p_emp_id ;
  RETURN v_emprec;
END;
/
DECLARE
  v_emprec employees%ROWTYPE := get_emp(100);
BEGIN
  DBMS_OUTPUT.PUT_LINE('Emp: '|| v_emprec.last_name);
END;
/
```

```
FUNCTION get_emp(p_emp_id Compiled.
anonymous block completed
Emp: King
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Executing a PL/SQL Anonymous Block Dynamically

```
CREATE FUNCTION annual_sal( p_emp_id NUMBER)
RETURN NUMBER IS
  v_plsql varchar2(200) :=
    'DECLARE '||
    ' rec_emp employees%ROWTYPE; '||
    'BEGIN '||
    ' rec_emp := get_emp(:empid); ' ||
    ' :res := rec_emp.salary  * 12; ' ||
    'END;';
  v_result NUMBER;
BEGIN
 EXECUTE IMMEDIATE v_plsql
        USING IN p_emp_id, OUT v_result;
  RETURN v_result;
END;
/
EXECUTE DBMS_OUTPUT.PUT_LINE(annual_sal(100))
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 36

Using Native Dynamic SQL to Compile PL/SQL Code

- Compile PL/SQL code with the ALTER statement:
  - ALTER PROCEDURE name COMPILE
  - ALTER FUNCTION name COMPILE
  - ALTER PACKAGE name COMPILE SPECIFICATION
  - ALTER PACKAGE name COMPILE BODY

```
CREATE PROCEDURE compile_plsql(p_name VARCHAR2,
 p_plsql_type VARCHAR2, p_options VARCHAR2 := NULL) IS
  v_stmt varchar2(200) := 'ALTER '|| p_plsql_type ||
                          ' '|| p_name || ' COMPILE';
BEGIN
 IF p_options IS NOT NULL THEN
   v_stmt := v_stmt || ' ' || p_options;
 END IF;
 EXECUTE IMMEDIATE v_stmt;
END;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 37

Using the DBMS_SQL Package

- The DBMS_SQL package is used to write dynamic SQL in stored procedures and to parse DDL statements.
- You must use the DBMS_SQL package to execute a dynamic SQL statement that has an unknown number of input or output variables, also known as Method 4.
- In most cases, NDS is easier to use and performs better than DBMS_SQL except when dealing with Method 4.
- For example, you must use the DBMS_SQL package in the following situations:
  - You do not know the SELECT list at compile time
  - You do not know how many columns a SELECT statement will return, or what their data types will be

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 38

Using the DBMS_SQL Package Subprograms

- Examples of the package procedures and functions:
  - OPEN_CURSOR
  - PARSE
  - BIND_VARIABLE
  - EXECUTE
  - FETCH_ROWS
  - CLOSE_CURSOR

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Using DBMS_SQL with a DML Statement: Deleting Rows

```
CREATE OR REPLACE FUNCTION delete_all_rows
  (p_table_name  VARCHAR2) RETURN NUMBER IS
   v_cur_id      INTEGER;
   v_rows_del    NUMBER;
BEGIN
  v_cur_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(v_cur_id,
    'DELETE FROM '|| p_table_name, DBMS_SQL.NATIVE);
  v_rows_del := DBMS_SQL.EXECUTE (v_cur_id);
  DBMS_SQL.CLOSE_CURSOR(v_cur_id);
  RETURN v_rows_del;
END;
/
```

```
CREATE TABLE temp_emp AS SELECT * FROM employees;
BEGIN
 DBMS_OUTPUT.PUT_LINE('Rows Deleted: ' ||
delete_all_rows('temp_emp'));
END;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

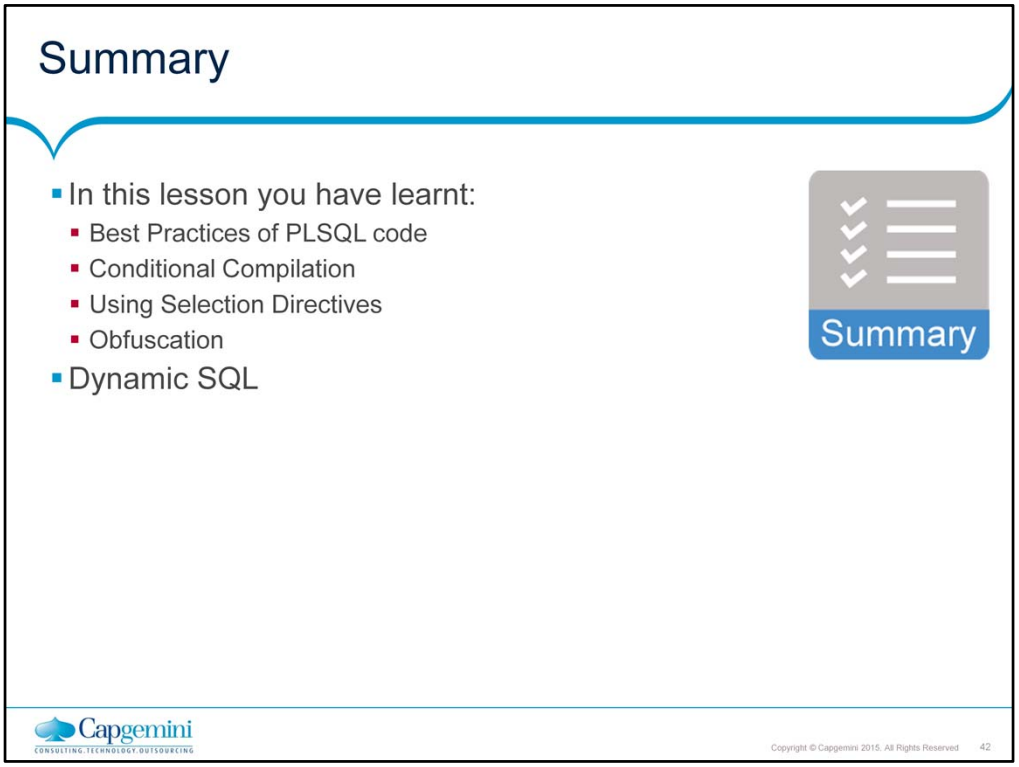
 40

Using DBMS_SQL with a Parameterized DML Statement

```
CREATE PROCEDURE insert_row (p_table_name VARCHAR2,
 p_id VARCHAR2, p_name VARCHAR2, p_region NUMBER) IS
  v_cur_id      INTEGER;
  v_stmt        VARCHAR2(200);
  v_rows_added  NUMBER;
BEGIN
  v_stmt := 'INSERT INTO '|| p_table_name ||
            ' VALUES (:cid, :cname, :rid)';
  v_cur_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(v_cur_id, v_stmt, DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(v_cur_id, ':cid', p_id);
  DBMS_SQL.BIND_VARIABLE(v_cur_id, ':cname', p_name);
  DBMS_SQL.BIND_VARIABLE(v_cur_id, ':rid', p_region);
  v_rows_added := DBMS_SQL.EXECUTE(v_cur_id);
  DBMS_SQL.CLOSE_CURSOR(v_cur_id);
  DBMS_OUTPUT.PUT_LINE(v_rows_added||' row added');
END;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 41



Add the notes here.

Review Question

- Question 1:_____ unit is the process of hiding the PL/SQL source code.

- Question 2: NDS provides the ability to execute SQL statements whose structure is unknown until execution time.
  - True/False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 43

Add the notes here.