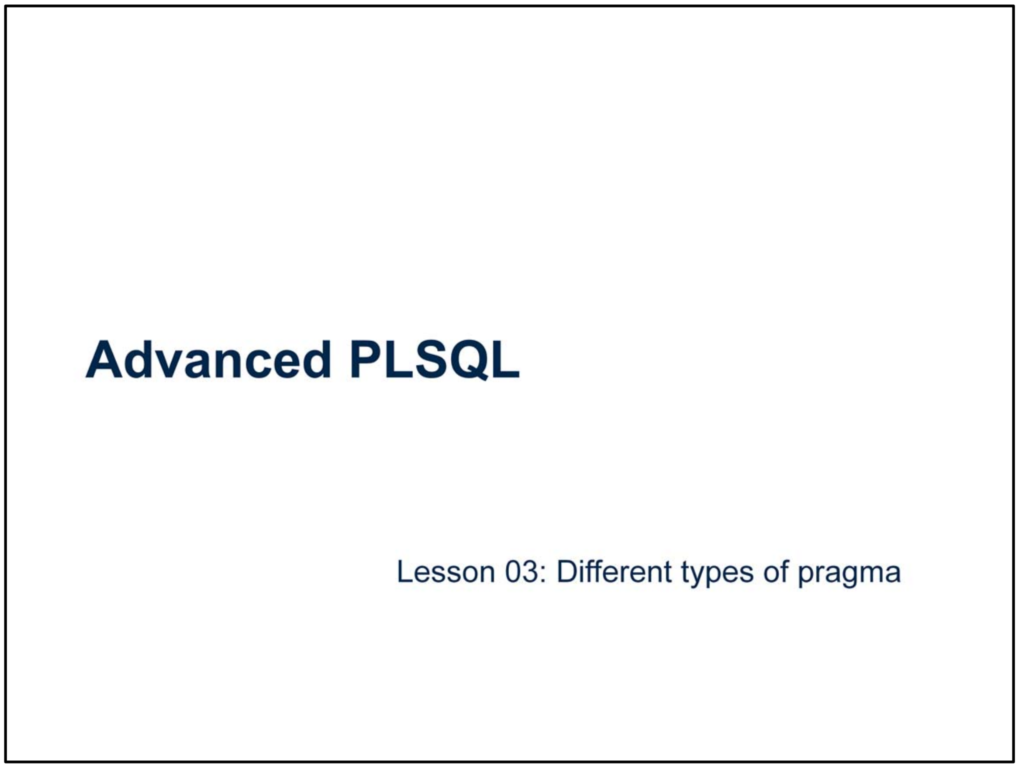
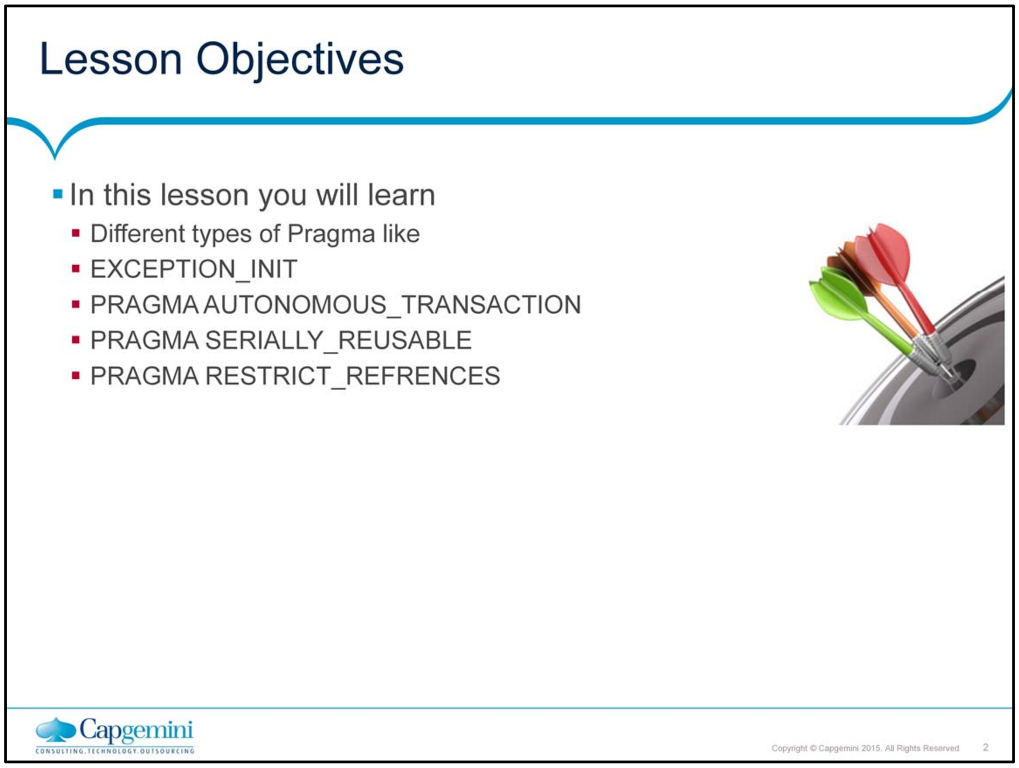


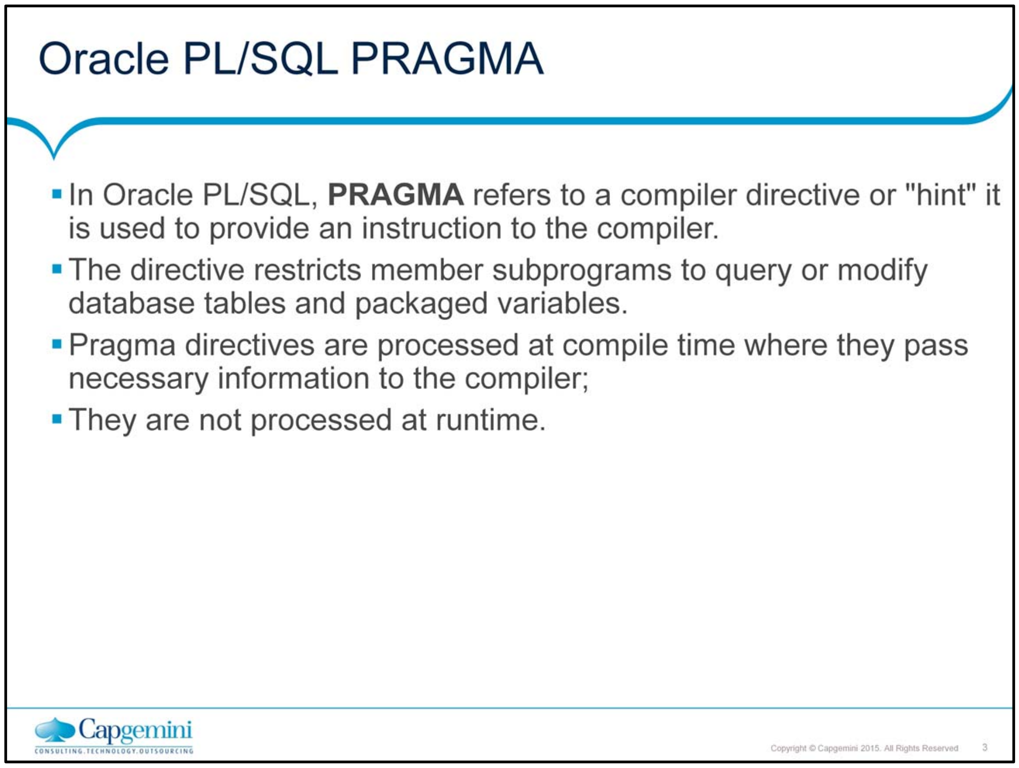
Advanced PLSQL

Lesson 03: Different types of pragma



Lesson Objectives

- In this lesson you will learn
  - Different types of Pragma like
  - EXCEPTION_INIT
  - PRAGMA AUTONOMOUS_TRANSACTION
  - PRAGMA SERIALLY_REUSABLE
  - PRAGMA RESTRICT_REFRENCES

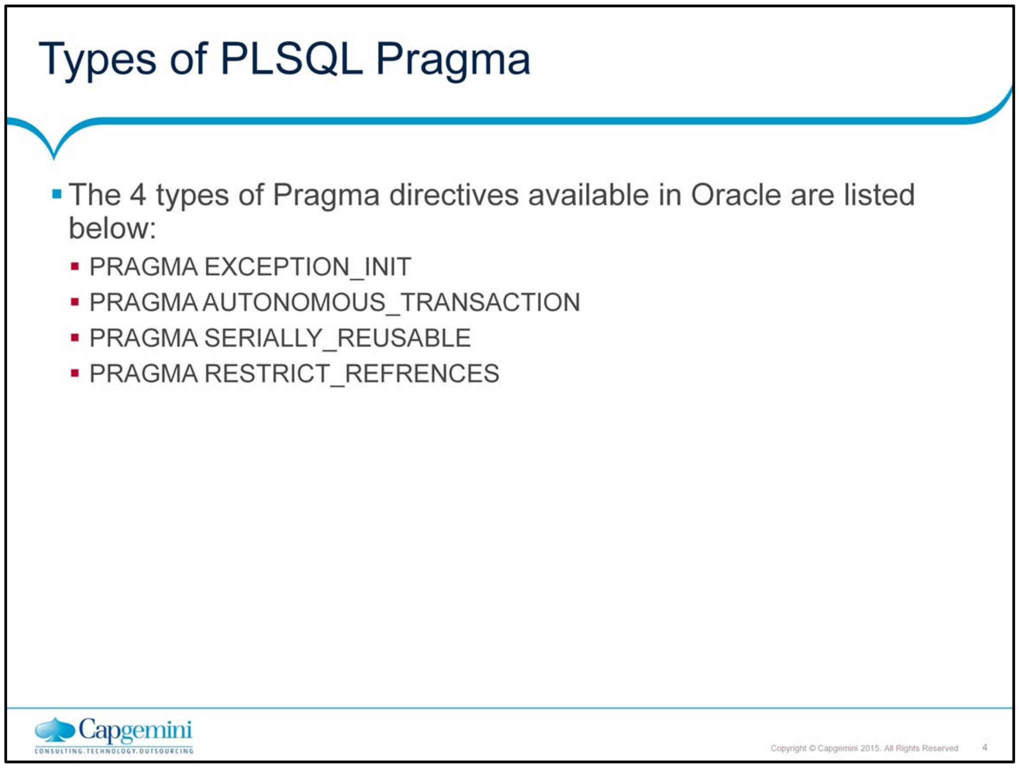


Oracle PL/SQL PRAGMA

- In Oracle PL/SQL, **PRAGMA** refers to a compiler directive or "hint" it is used to provide an instruction to the compiler.
- The directive restricts member subprograms to query or modify database tables and packaged variables.
- Pragma directives are processed at compile time where they pass necessary information to the compiler;
- They are not processed at runtime.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

3

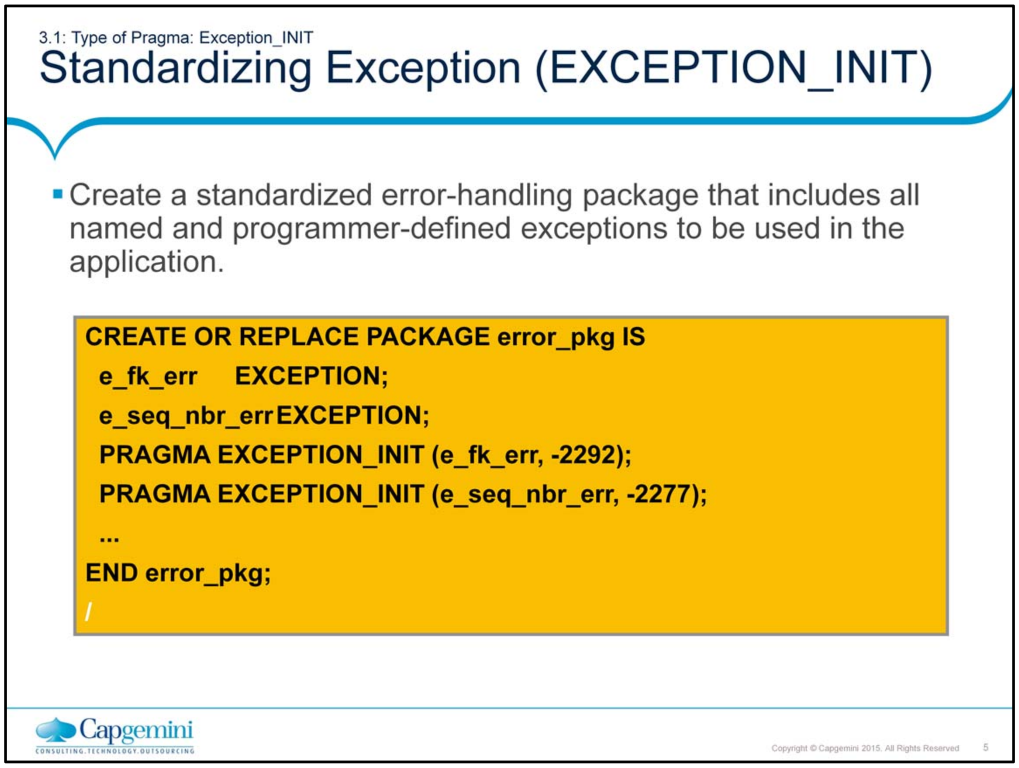


Types of PLSQL Pragma

- The 4 types of Pragma directives available in Oracle are listed below:
  - PRAGMA EXCEPTION_INIT
  - PRAGMA AUTONOMOUS_TRANSACTION
  - PRAGMA SERIALLY_REUSABLE
  - PRAGMA RESTRICT_REFRENCES

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

4

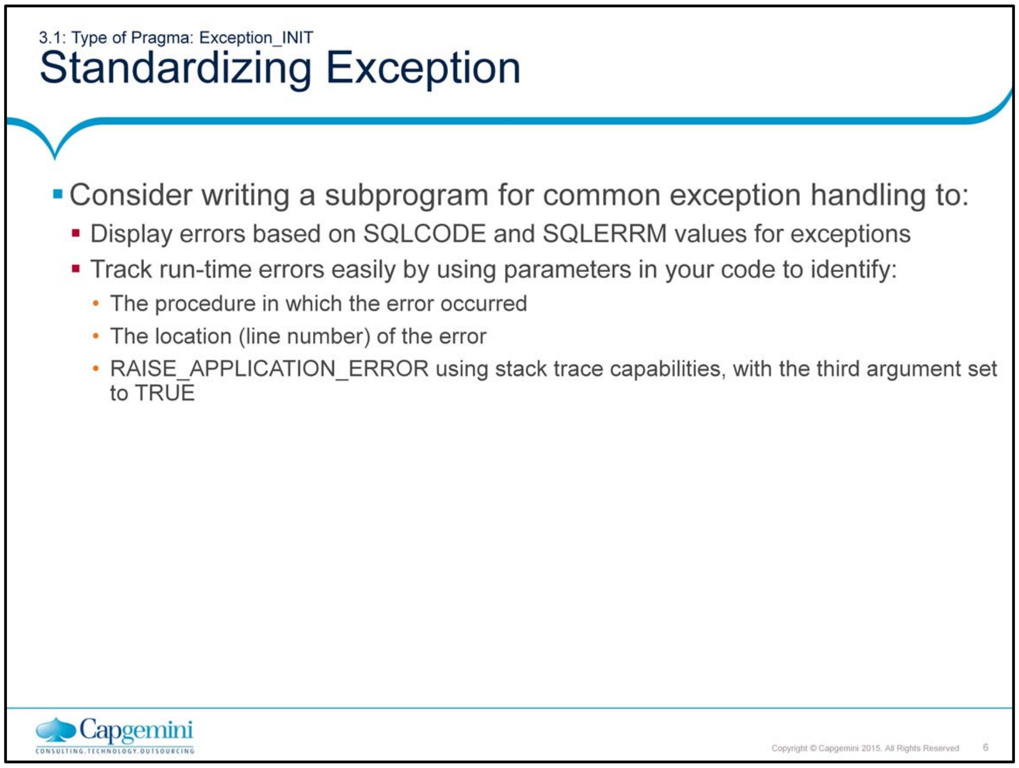


3.1: Type of Pragma: Exception_INIT

Standardizing Exception (EXCEPTION_INIT)

- Create a standardized error-handling package that includes all named and programmer-defined exceptions to be used in the application.

```
CREATE OR REPLACE PACKAGE error_pkg IS
  e_fk_err      EXCEPTION;
  e_seq_nbr_err EXCEPTION;
  PRAGMA EXCEPTION_INIT (e_fk_err, -2292);
  PRAGMA EXCEPTION_INIT (e_seq_nbr_err, -2277);
  ...
END error_pkg;
/
```

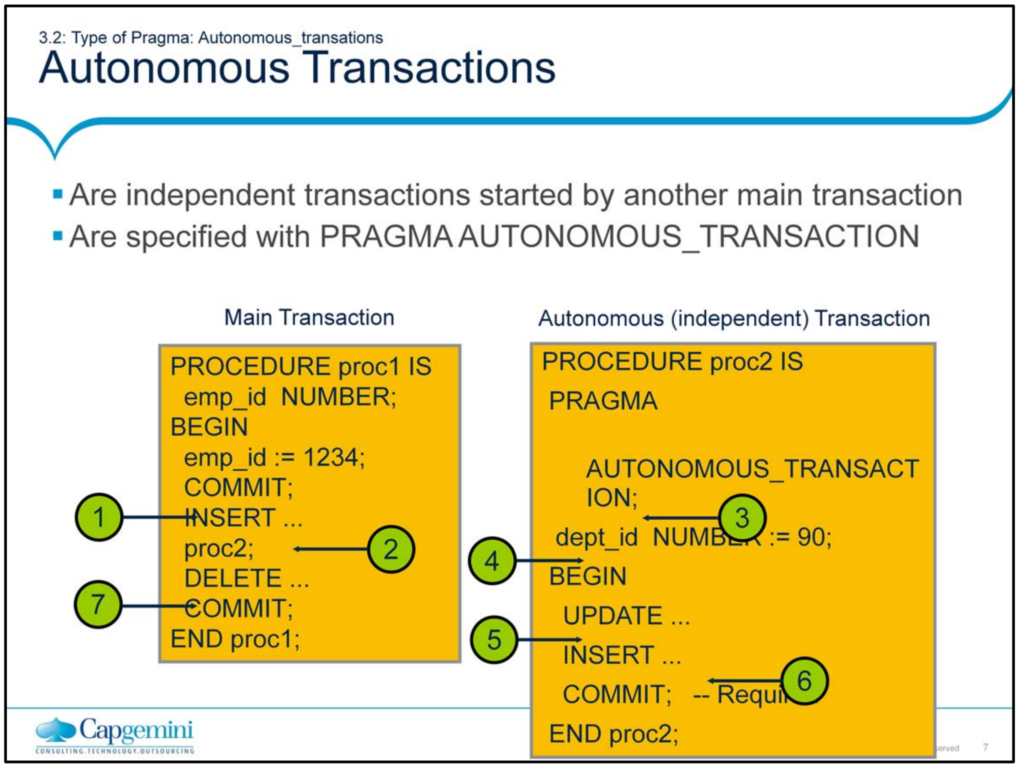


3.1: Type of Pragma: Exception_INIT

Standardizing Exception

- Consider writing a subprogram for common exception handling to:
  - Display errors based on SQLCODE and SQLERRM values for exceptions
  - Track run-time errors easily by using parameters in your code to identify:
    - The procedure in which the error occurred
    - The location (line number) of the error
    - RAISE_APPLICATION_ERROR using stack trace capabilities, with the third argument set to TRUE

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING



3.2: Type of Pragma: Autonomous_transations

Autonomous Transactions

- Are independent transactions started by another main transaction
- Are specified with PRAGMA AUTONOMOUS_TRANSACTION

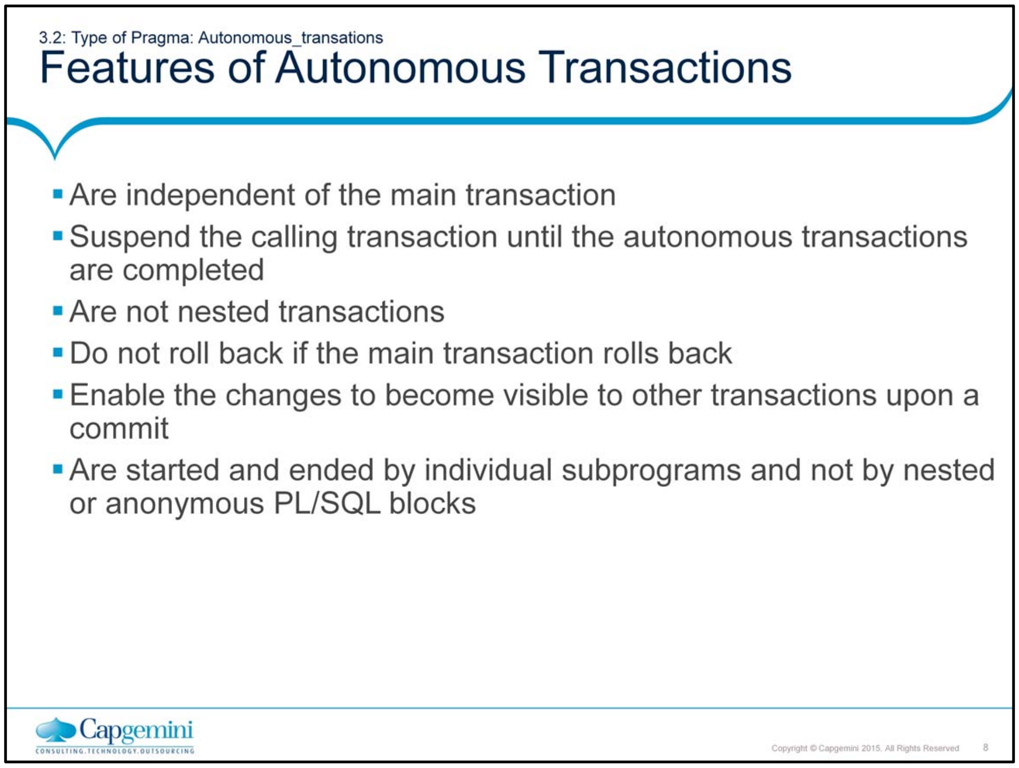
Main Transaction

```
PROCEDURE proc1 IS
  emp_id  NUMBER;
BEGIN
  emp_id := 1234;
  COMMIT;
  INSERT ...
  proc2;
  DELETE ...
  COMMIT;
END proc1;
```

Autonomous (independent) Transaction

```
PROCEDURE proc2 IS
 PRAGMA

    AUTONOMOUS_TRANSACT
    ION;
  dept_id  NUMBER := 90;
 BEGIN
   UPDATE ...
   INSERT ...
   COMMIT;   -- Requi
 END proc2;
```



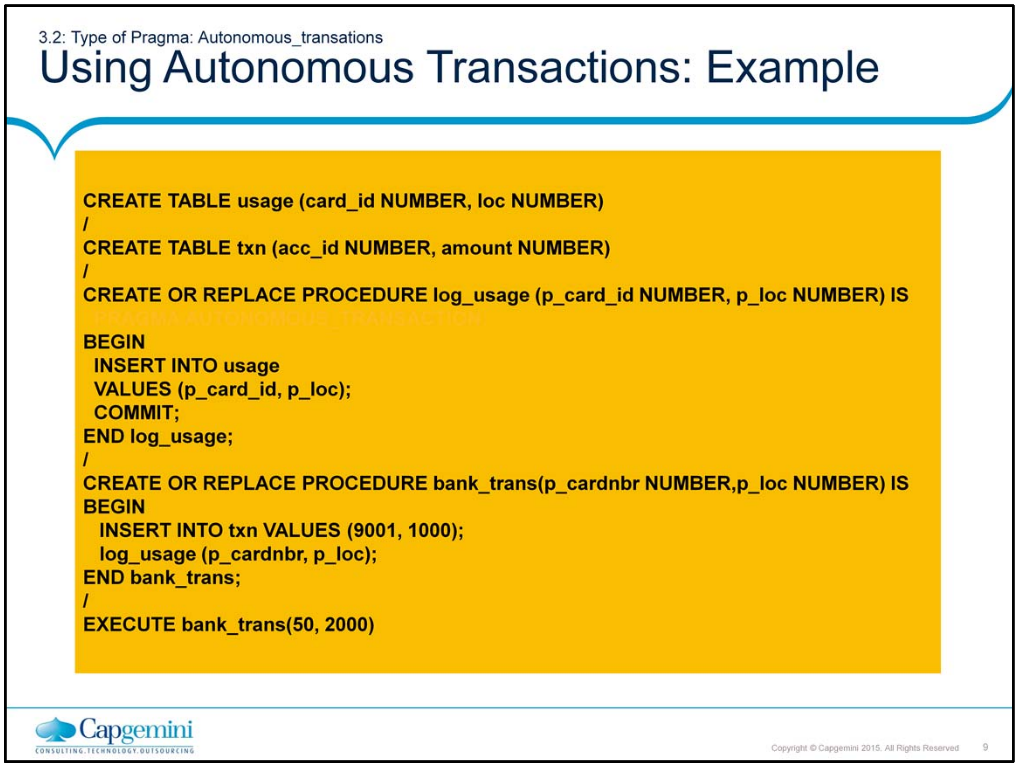
3.2: Type of Pragma: Autonomous_transations

Features of Autonomous Transactions

- Are independent of the main transaction
- Suspend the calling transaction until the autonomous transactions are completed
- Are not nested transactions
- Do not roll back if the main transaction rolls back
- Enable the changes to become visible to other transactions upon a commit
- Are started and ended by individual subprograms and not by nested or anonymous PL/SQL blocks

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

8



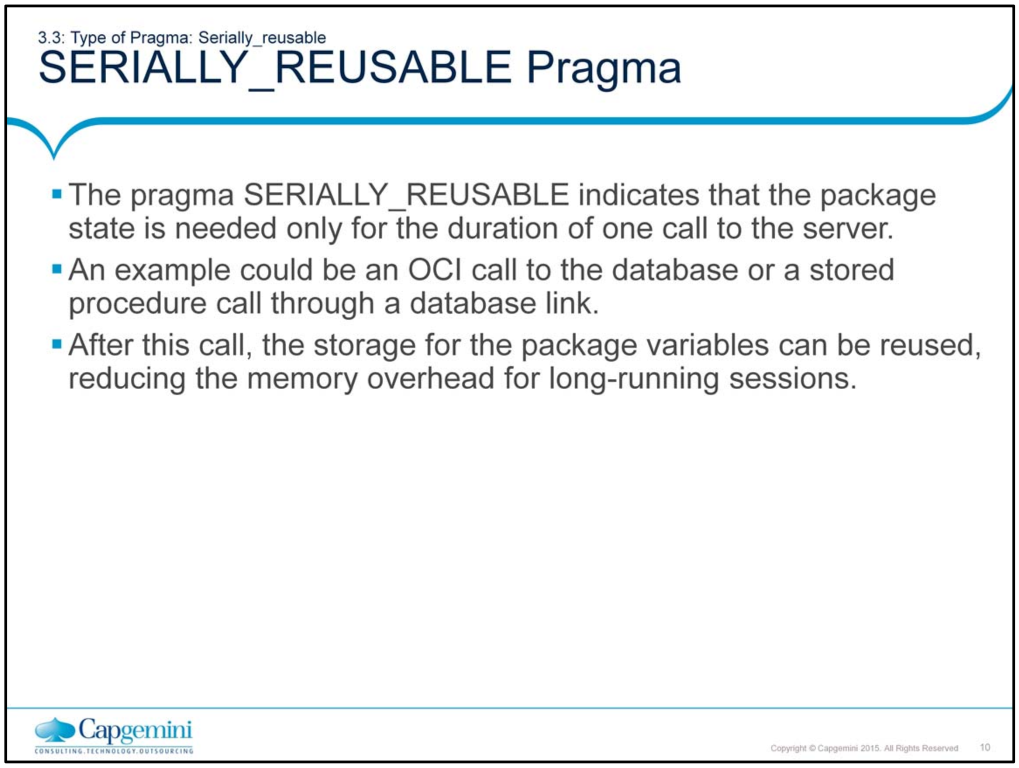
3.2: Type of Pragma: Autonomous_transations

Using Autonomous Transactions: Example

```
CREATE TABLE usage (card_id NUMBER, loc NUMBER)
/
CREATE TABLE txn (acc_id NUMBER, amount NUMBER)
/
CREATE OR REPLACE PROCEDURE log_usage (p_card_id NUMBER, p_loc NUMBER) IS
  PRAGMA AUTONOMOUS_TRANSACTION
BEGIN
  INSERT INTO usage
  VALUES (p_card_id, p_loc);
  COMMIT;
END log_usage;
/
CREATE OR REPLACE PROCEDURE bank_trans(p_cardnbr NUMBER,p_loc NUMBER) IS
BEGIN
   INSERT INTO txn VALUES (9001, 1000);
   log_usage (p_cardnbr, p_loc);
END bank_trans;
/
EXECUTE bank_trans(50, 2000)
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

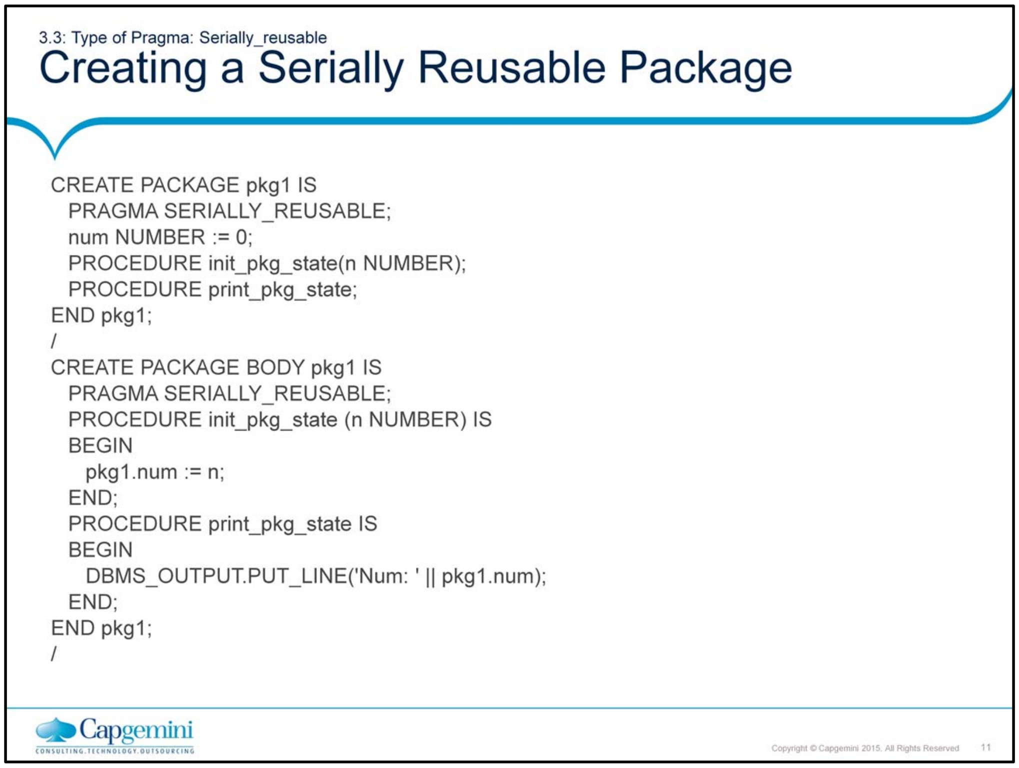
9



3.3: Type of Pragma: Serially_reusable

SERIALLY_REUSABLE Pragma

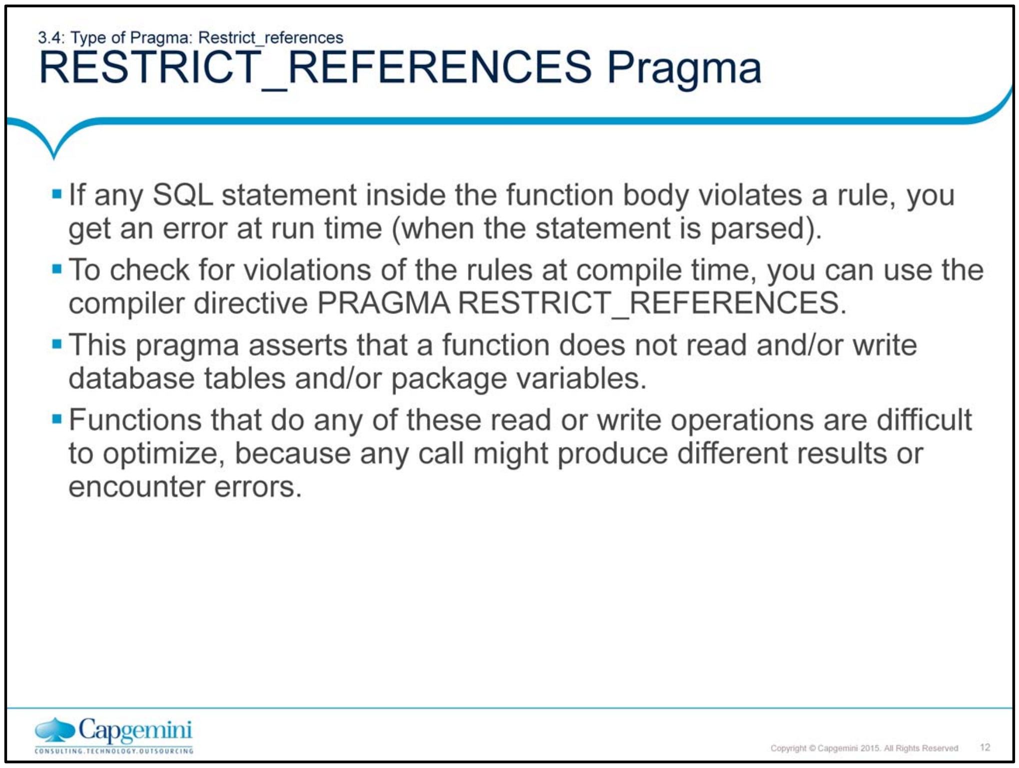
- The pragma SERIALLY_REUSABLE indicates that the package state is needed only for the duration of one call to the server.
- An example could be an OCI call to the database or a stored procedure call through a database link.
- After this call, the storage for the package variables can be reused, reducing the memory overhead for long-running sessions.



3.3: Type of Pragma: Serially_reusable

Creating a Serially Reusable Package

```
CREATE PACKAGE pkg1 IS
  PRAGMA SERIALLY_REUSABLE;
  num NUMBER := 0;
  PROCEDURE init_pkg_state(n NUMBER);
  PROCEDURE print_pkg_state;
END pkg1;
/
CREATE PACKAGE BODY pkg1 IS
  PRAGMA SERIALLY_REUSABLE;
  PROCEDURE init_pkg_state (n NUMBER) IS
  BEGIN
    pkg1.num := n;
  END;
  PROCEDURE print_pkg_state IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Num: ' || pkg1.num);
  END;
END pkg1;
/
```



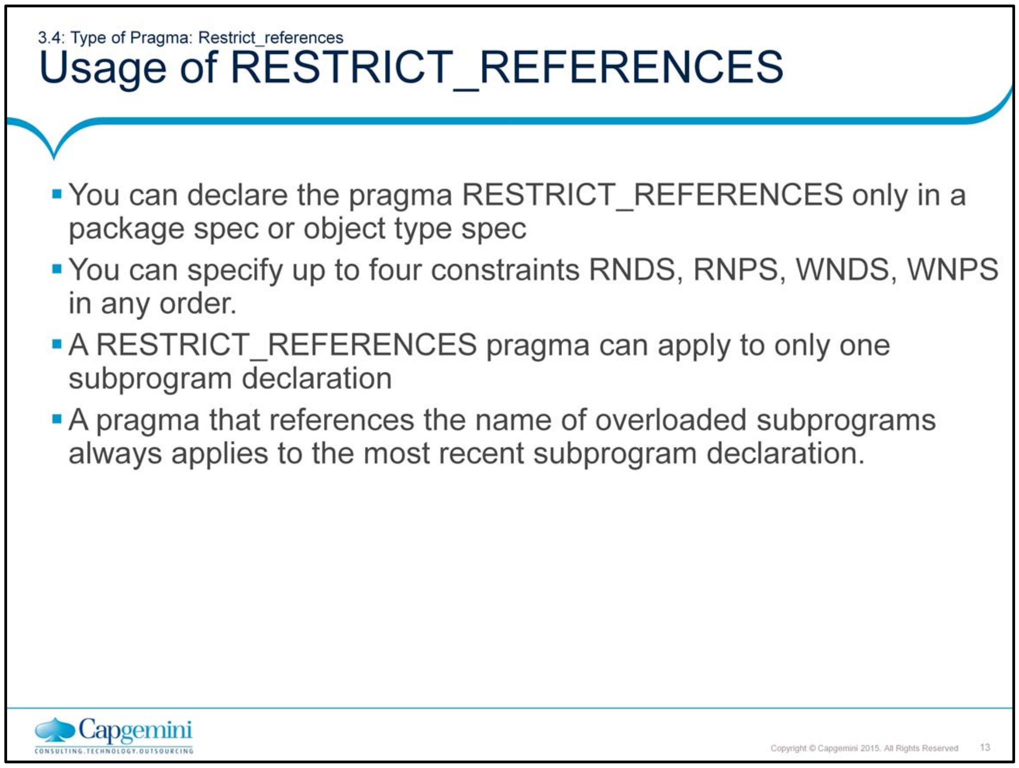
3.4: Type of Pragma: Restrict_references

RESTRICT_REFERENCES Pragma

- If any SQL statement inside the function body violates a rule, you get an error at run time (when the statement is parsed).
- To check for violations of the rules at compile time, you can use the compiler directive PRAGMA RESTRICT_REFERENCES.
- This pragma asserts that a function does not read and/or write database tables and/or package variables.
- Functions that do any of these read or write operations are difficult to optimize, because any call might produce different results or encounter errors.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

12

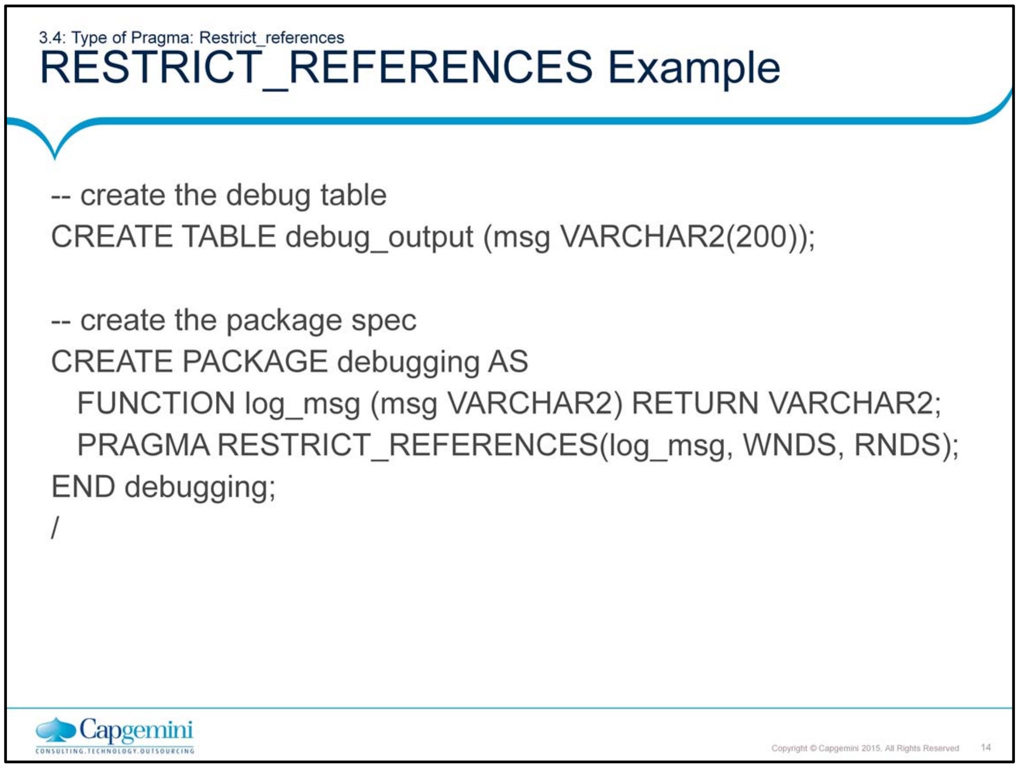


3.4: Type of Pragma: Restrict_references

Usage of RESTRICT_REFERENCES

- You can declare the pragma RESTRICT_REFERENCES only in a package spec or object type spec
- You can specify up to four constraints RNDS, RNPS, WNDS, WNPS in any order.
- A RESTRICT_REFERENCES pragma can apply to only one subprogram declaration
- A pragma that references the name of overloaded subprograms always applies to the most recent subprogram declaration.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



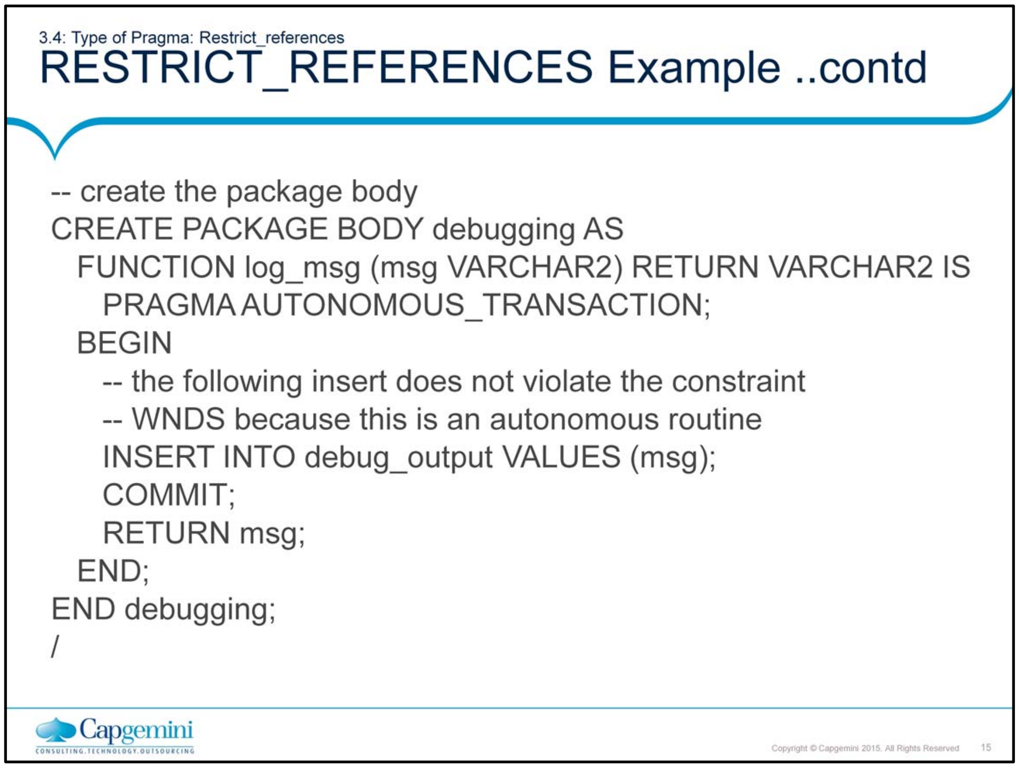
3.4: Type of Pragma: Restrict_references

RESTRICT_REFERENCES Example

```
-- create the debug table
CREATE TABLE debug_output (msg VARCHAR2(200));

-- create the package spec
CREATE PACKAGE debugging AS
   FUNCTION log_msg (msg VARCHAR2) RETURN VARCHAR2;
   PRAGMA RESTRICT_REFERENCES(log_msg, WNDS, RNDS);
END debugging;
/
```

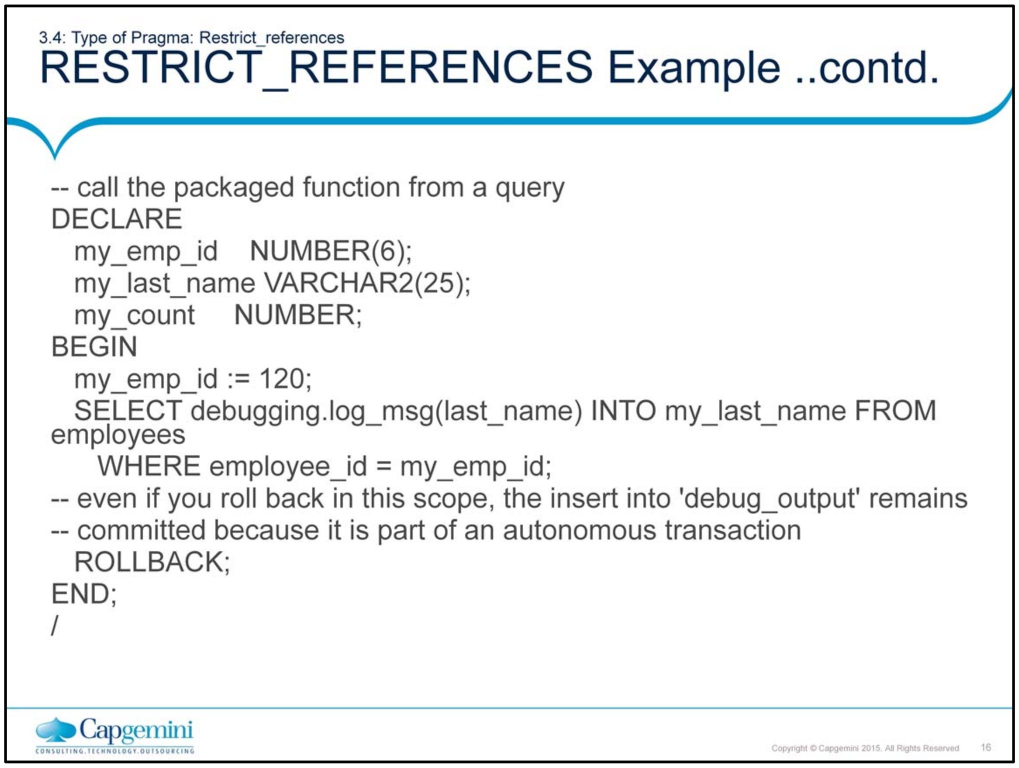
14



3.4: Type of Pragma: Restrict_references

RESTRICT_REFERENCES Example ..contd

```
-- create the package body
CREATE PACKAGE BODY debugging AS
  FUNCTION log_msg (msg VARCHAR2) RETURN VARCHAR2 IS
    PRAGMA AUTONOMOUS_TRANSACTION;
  BEGIN
    -- the following insert does not violate the constraint
    -- WNDS because this is an autonomous routine
    INSERT INTO debug_output VALUES (msg);
    COMMIT;
    RETURN msg;
  END;
END debugging;
/
```

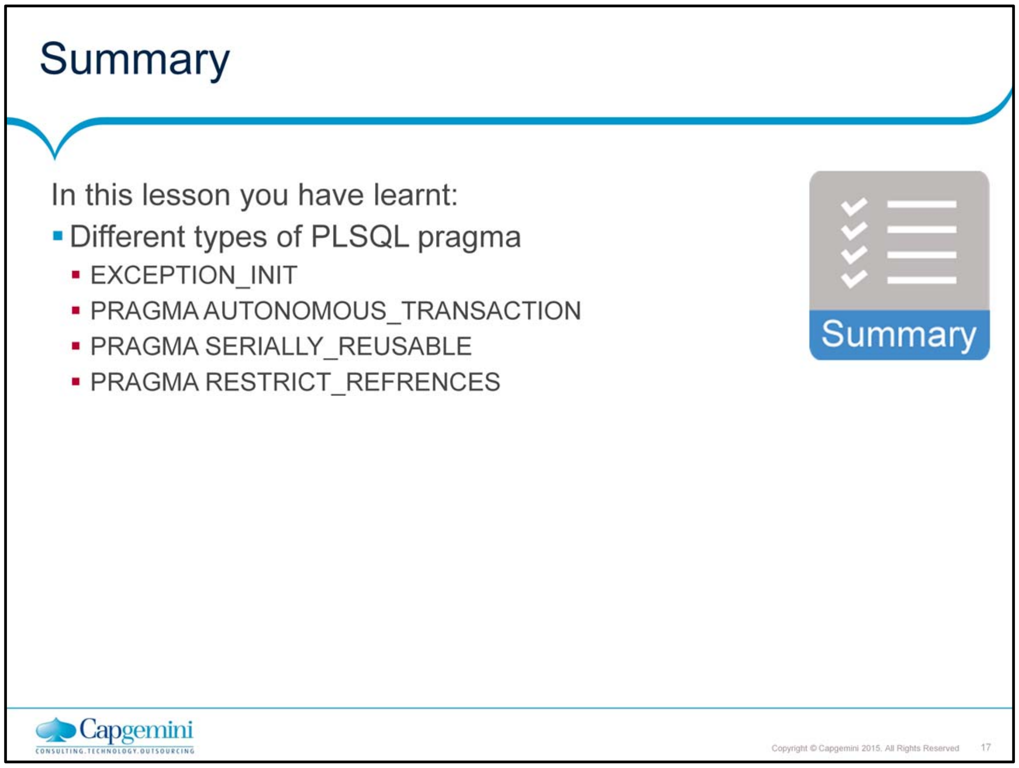


3.4: Type of Pragma: Restrict_references

RESTRICT_REFERENCES Example ..contd.

```
-- call the packaged function from a query
DECLARE
  my_emp_id    NUMBER(6);
  my_last_name VARCHAR2(25);
  my_count     NUMBER;
BEGIN
  my_emp_id := 120;
  SELECT debugging.log_msg(last_name) INTO my_last_name FROM
employees
    WHERE employee_id = my_emp_id;
-- even if you roll back in this scope, the insert into 'debug_output' remains
-- committed because it is part of an autonomous transaction
  ROLLBACK;
END;
/
```

16



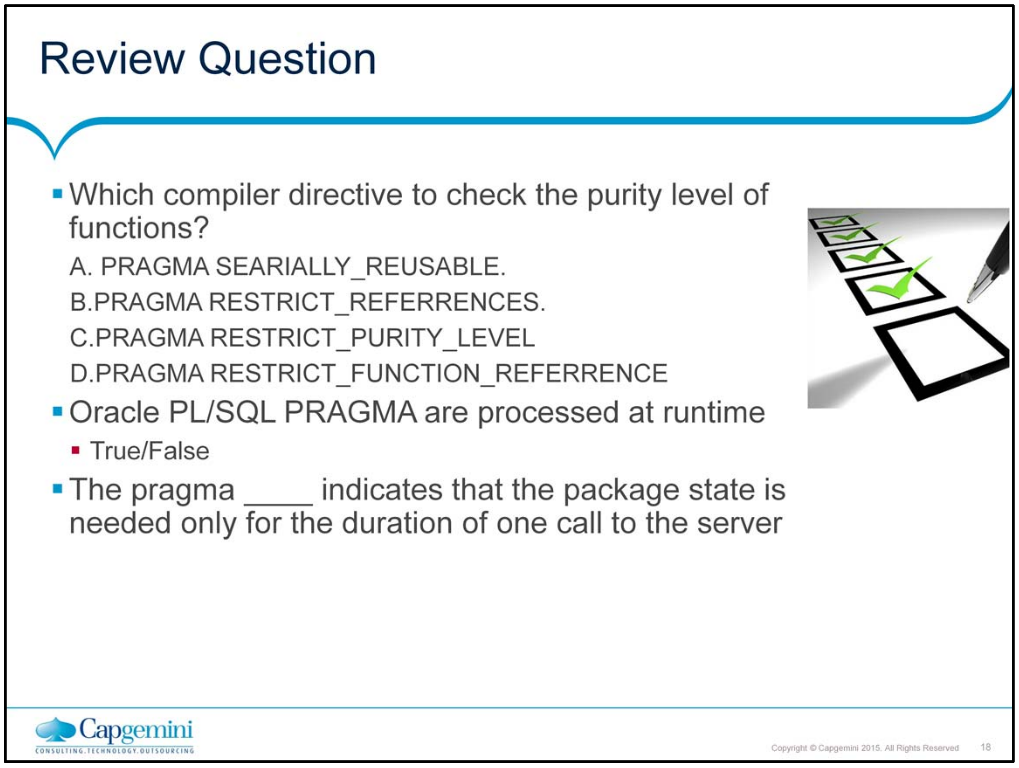
Summary

In this lesson you have learnt:

- Different types of PLSQL pragma
  - EXCEPTION_INIT
  - PRAGMA AUTONOMOUS_TRANSACTION
  - PRAGMA SERIALLY_REUSABLE
  - PRAGMA RESTRICT_REFRENCES

17

Add the notes here.



Review Question

- Which compiler directive to check the purity level of functions?
  A. PRAGMA SEARIALLY_REUSABLE.
  B.PRAGMA RESTRICT_REFERRENCES.
  C.PRAGMA RESTRICT_PURITY_LEVEL
  D.PRAGMA RESTRICT_FUNCTION_REFERRENCE
- Oracle PL/SQL PRAGMA are processed at runtime
  - True/False
- The pragma ____ indicates that the package state is needed only for the duration of one call to the server

Add the notes here.