Oracle PL/SQL Intermediate

Most Important Features

www.lingaro.com

lingaro

The Value of Business Intelligence

Training Notes

- Course contains Lingaro PL/SQL Standard rules
  - Rule is indicated by blue stars
  - More stars means more important rule

Training Agenda

- Exceptions
- PL/SQL Composite Types
- Processing Cursors
- Creating Packages
- Dependencies
- Oracle Supplied Packages
- Database Triggers
- PL/SQL Security
- Manage PL/SQL Units

Topic Agenda

Exceptions

- Exception Types
- From Basic Training
  - Predefined Exceptions
  - OTHERS Keyword
  - RAISE_APPLICATION_ERROR
- Programmer Exceptions
  - Implicitly Raised
  - Explicitly Raised
- Stored Procedure Transactions

Exception Types

- Exception are errors generated during PL/SQL code runtime
- Named Exceptions
  - Names predefined by Oracle

```
RDBMS errors                              - ORA error prefix
PL/SQL engine (in database) errors        - PLS error prefix
Forms Server engine errors                - FRM error prefix
Reports Server engine errors              - RPT error prefix
```

  - Declared by programmer in block

```
Raised implicitly
   - Exception name is assigned to error by programmer
   - Exception is raised by server
Raised explicitly
   - Exception is raised by RAISE statement in code
```

- Unnamed Exceptions
  - Errors not included in predefined or declared exceptions

Predefined Exceptions

- Names

```
DUP_VAL_ON_INDEX
INVALID_CURSOR
NO_DATA_FOUND
TOO_MANY_ROWS
ZERO_DIVIDE
```

full list from documentation

- Are assigned to server error code
- When assigned error happen – exception is raised
- Next statements in block section are not executed
- If exception is not handled then propagate outside block
- Indent statements below WHEN using 5 characters ★★
- Handling Example

```
EXCEPTION
  WHEN NO_DATA_FOUND OR TO_MANY_ROWS THEN
       pro_error_message(in_msg => 'Query must return one row');
  WHEN ZERO_DIVIDE THEN
       pro_error_message(in_msg => 'Division by zero');
END;
```

OTHERS Keyword

- You can handle all errors by using OTHERS keyword
- RAISE statement must be used to propagate other errors ★★★
- Do not hide other errors using NULL statement ★★★
- Rollback transaction ★★★
- Use SQLERRM and FORMAT_ERROR_BACKTRACE functions ★★★
  - To get standard error message for this unknown error

- Example

```
EXCEPTION
  WHEN NO_DATA_FOUND OR TO_MANY_ROWS THEN
       pro_errmsg('Query must return one row');
  WHEN OTHERS THEN
       l_sql_errmsg := SQLERRM;
       l_err_stack  := dbms_utility.format_error_backtrace;
       ROLLBACK;
       pro_errmsg(in_mdg => l_sql_errmsg || l_err_stack);
       RAISE;
END;
```

RAISE_APPLICATION_ERROR Procedure

- This procedure is used to generate custom application level server error
- For this kind of errors server reserved codes between -20000 and -20999
- Syntax

```
RAISE_APPLICATION_ERROR( -<numeric code>, 'error message');
```

- Can be used in exception handling instead of RAISE statement

```
EXCEPTION
 ...
  WHEN OTHERS THEN
       RAISE_APPLICATION_ERROR(-20154, 'Process ' ||
         l_pprcss_name || ' has unexpected error: ' ||
         SQLERRM || dbms_utility.format_error_backtrace);
END;
```

- Can be used in block executable section on application error

```
IF ((l_month_num < 1) OR (l_month_num >12)) THEN
  RAISE_APPLICATION_ERROR(-20154, 'Wrong month number');
END IF;
```

Programmer Exception

Implicitly Raised

- Declare exception
- Assign exception do server error

```
PROCEDURE pro_add_cust(in_cust_name IN VARCHAR2)
IS
  e_insert_null EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_insert_null, -01400);
BEGIN
  INSERT INTO cust_dim (cust_skid, cust_name)
    VALUES (cust_seq.NEXTVAL, in_cust_name);
EXCEPTION
  WHEN e_insert_null THEN
       dbms_output.put_line('INSERT OPERATION FAILED');
       dbms_output.put_line(SQLERRM);
END;
```

NOT NULL constraint violated

Programmer Exception

Explicitly Raised

- Declare exception
- Raise exception using RAISE statement

```
...
  e_month_num EXCEPTION;
BEGIN
  ...
  IF ((l_month_num < 1) OR (l_month_num >12)) THEN
    RAISE e_month_num;
  END IF;
  ...
EXCEPTION
  WHEN e_month_num THEN
       dbms_output.put_line('Wrong month number');
END;
```

Stored Units Transactions

- Implicit savepoint is auto-created when stored subprogram is started
- IF exception is not handled transaction is rolled back to savepoint
- IF exception is handled in stored subprogram
  - then implicit rollback to subprogram savepoint is not executed

```
CREATE OR REPLACE PROCEDURE pro_add_work(in_work_code IN VARCHAR2)
IS
BEGIN
  INSERT INTO work_code_prc (work_code) VALUES(in_work_code);
  IF ...
    RAISE no_data_found;
  END IF;
EXCEPTION
  WHEN no_data_found THEN
       pro_errmsg(in_msg => 'Work template is not found');
END;
```

Implicite savepoint

Topic Agenda

PL/SQL Composite Types

- Records
- Associated Array Collections

PL/SQL Composite Types

- RECORD
- Indexed TABLE collection

- Can be declared in PL/SQL block declarations section
- Can not be created as database schema types
- Unlike scalar type composite type contains components
- Advantages
  - Operation one composite variable is easier and faster then on many scalar variables
  - PL/SQL only types are faster in PL/SQL engine then SQL types
  - Can use composite type as type for another composite type component
  - Nesting level number is not limited

RECORD Type

prod_skid	mkt_name	end_date	sales_amt	unit_cnt	rep_ind
NUMBER(38)	VARCHAR2(90)	DATE	NUMBER(38,2)	NUMBER(38)	BOOLEAN
4529917	Total XAOC	10-MAY-2014	3023099.12	326443	TRUE

- Use **t_** prefix in all declared type names ★★★
- Field like variable can be NOT NULL or initialized
  - Can be composite type (block declared or created in database)

```
TYPE t_sales_record IS RECORD (prod_skid NUMBER,
                               mkt_name  VARCHAR2(90),
                               end_date  DATE,
                               sales_amt NUMBER(38,2),
                               unit_cnt  NUMBER(38),
                              rep_ind   BOOLEAN NOT NULL := FALSE,
                               flags_rec t_flags_rec);

l_sales_record t_sales_record;
```

RECORD Type

- %ROWTYPE Attribute
  - Used to declare record based on table or view definition
  - Fields will be auto-modified after table columns change
  - Such variable typically is filled using `SELECT * INTO l_record_variable`
  - Not recommended if small part from many columns needed ★★

```
l_sales_record sales_fct%ROWTYPE;
```

- Using record variables

```
SELECT *
  INTO l_sales_record
  FROM sales_fct
  WHERE ...
...
l_sales_record_copy := l_sales_record;
l_sales_record.rep_ind := (l_sales_record.sales_amt > 1000000);
...
INSERT INTO sales_fct
  VALUES l_sales_record;
...
UPDATE sales_fct
   SET ROW = l_sales_record;
WHERE ...
```

select only one row

RECORD Variables

- Can be used in
  - Parameters of procedures and function (local and in package subprograms)

```
DECLARE
  l_sales_record sales_fct%ROWTYPE;
  PROCEDURE pro_put_salary(in_sales_record IN sales_fct%ROWTYPE)
  ...
BEGIN
  ...
  pro_put_salary(in_sales_record => l_sales_record)
```

  - Function type (local and in package functions)

```
...
  FUNCTION fn_get_salary(...) RETURN t_sales_record
  ...
BEGIN
  ...
  l_sales_record := fn_get_salary(...
```

Associated Array Collections

INDEX BY TABLE Type

- Contains unbounded collection of same type elements
- Index is used to access one element
- Similar to other 3-gen lang. arrays
  - No need to initialize
  - No need to use index values in order

Index key	Value
PLS_INTEGER or VARCHAR2	scalar or composite type e.g. VARCHAR2
62	Walmart
107	CVS
2	Target

```
TYPE t_cntry_list IS TABLE OF VARCHAR2(100) INDEX BY VARCHAR2(2);
l_cntry_list t_cntry_list;
...
l_cntry_list('pl')  := 'Poland';


TYPE t_name_list IS TABLE OF VARCHAR2(100) INDEX BY PLS_INTEGER;
l_num_list t_num_list;
...
l_num_list(62)  := 'Walmart';
l_num_list(107) := 'CVS';
l_num_list(2)   := 'Target';
```

- PLS_INTEGER index can be positive, negative and zero but not NULL

Associated Arrays Collections

INDEX BY TABLE Type

- Collection of records

Index key	Record cust_code | cust_name | ...
WM	WM | Walmart | ...
TG	TG | Target | ...

```
DECLARE
  TYPE t_cust_list IS TABLE OF cust_dim%ROWTYPE INDEX BY VARCHAR2(10);
  l_cust_list t_cust_list;
BEGIN
  SELECT *
    INTO l_cust_list('WM')
    FROM cust_dim
   WHERE cust_code = 'WM';
  ...
  l_cust_code := 'WM';
  l_cust_name := l_cust_list(l_cust_code).cust_name;
```

Associated Arrays

Methods

```
EXISTS(index)   COUNT    FIRST        LAST
DELETE(index)   DELETE   NEXT(index)  PRIOR(index)
```

- Usage examples

```
pro_send_msg('List contains ' || l_cust_list.COUNT || ' customers. ');

...
IF l_cust_list.EXISTS(l_index) THEN
  l_cust_name := l_cust_list(l_index).cust_name;
...
```

- Access all elements using loop

```
l_index := l_cust_list.FIRST;
LOOP
  l_cust_name := l_cust_list(l_index).cust_name;
  l_index := l_cust_list.NEXT(l_index);
  EXIT WHEN (l_index IS NULL);
END LOOP;
```

Associated Arrays

Capabilities

- Order of elements during reading is not dependent of inserting order
- Elements are ascending ordered by index key value during read

e.g.

```
DECLARE
   TYPE t_num_list IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
   l_num_list t_num_list;
 BEGIN
   l_num_list(0) := 0;
   l_num_list(7) := 7;
   l_num_list(-1) := -1;
   dbms_output.put_line(l_num_list.FIRST);
   dbms_output.put_line(l_num_list.LAST);
 END;
Results:
  -1
  7
```

- Index is used to find element very fast

Associated Arrays

Named and Unnamed Exceptions

- NO_DATA_FOUND – access to not existence element

```
l_num_list.DELETE(-5);
l_num := l_num_list(-5);
```

- VALUE_ERROR – index is NULL

```
l_num_list(NULL) := 8;
```

- ORA-01426: numeric overflow
  - index outside PLS_INTEGER limit

```
l_num_list(2147483648) := 8;
```

Topic Agenda

Processing Cursors

- Implicit Cursor
- Cursor Attributes
- Explicit Cursor
  - Processing Inside Loop
  - FETCH More Then One Row
- Cursor Parameters
- Locking Processed Rows

Implicit Cursor

- All statements in PL/SQL which uses SQL engine are implicit cursors
  - e.g. SELECT, UPDATE, COMMIT, SAVEPOINT, ROLLBACK, EXECUTE IMMEDIATE, …
- Cursor attributes

  `%ISOPEN, %NOTFOUND, %FOUND, %ROWCOUNT`
- Implicit cursors for attribute uses SQL keyword instead of cursor name
- Can check attribute only for last executed implicit cursor
  - during PL/SQL engine current call
- Save attribute into variable if you need to use it later

```
BEGIN
  DELETE sales_fct
   WHERE start_date < SYSDATE - 90;
  l_rows_cnt := SQL%ROWCOUNT;
  COMMIT;
  pro_send_msg(in_msg => l_rows_cnt || ' rows deleted. ');
END;
```

Explicit Cursor

- Only query can be used as explicit cursor
- Use ORDER BY in query if specific row order is needed during fetch
- FETCH gets one row from cursor and moves position to next row
- Cursor position can't be moved back without reopen
- Use **c_** prefix and **_cur** suffix in cursor name ★★★

```
DECLARE
  CURSOR c_task_list_cur IS
    SELECT task_code, task_name, task_actions
      FROM task_list_prc;
  l_task_record c_task_list_cur%ROWTYPE;
 ...
BEGIN
  ...
  OPEN c_task_list_cur;
  ...
  FETCH c_task_list_cur INTO l_code, l_name, l_actions;
  FETCH c_task_list_cur INTO l_task_record;
  ...
  CLOSE c_task_list_cur;
  ...
END;
```

Explicit Cursor

Processing Inside Loop

- Basic loop

```
    l_task_record c_task_list_cur%ROWTYPE;
  BEGIN
    OPEN c_task_list_cur;
    LOOP
      FETCH c_task_list_cur INTO l_task_record;
      EXIT WHEN c_task_list_cur%NOTFOUND;
      pro_run_task(in_task_record => l_task_record);
  END LOOP;
    CLOSE c_task_list_cur;
```

- FOR loop

```
  BEGIN
    FOR l_task_record IN c_task_list_cur LOOP
      pro_run_task(in_task_record => l_task_record);
    END LOOP;
```

Explicit Cursor

FETCH More Then One Row

- Collection of record type variable is needed

```
IS
  TYPE t_task_list IS TABLE OF task_list_prc%ROWTYPE INDEX BY PLS_INTEGER;
  l_task_list t_task_list;
  ...
```

- Fetch limited number of rows
  - Collection index values are auto-assigned starting from 1 and are incremented by 1

```
BEGIN
  OPEN c_task_list_cur;
  LOOP
    FETCH c_task_list_cur BULK COLLECT INTO l_cust_list LIMIT in_limit;
    EXIT WHEN l_task_list.COUNT < in_limit;
    pro_run_task_list(in_task_list => l_task_list);
END LOOP;
CLOSE employees_cur;
```

  - Similar limit for SELECT stamement is available in 12c

```
FETCH FIRST
```

Explicit Cursor

Parameters

- Use parameters instead of variables in cursor query ★
- It makes code easer, more readable and less error prone

```
DECLARE
  CURSOR c_task_list_cur(in_cust_code VARCHAR2) IS
    SELECT task_code, task_name, task_actions
      FROM task_list_prc
     WHERE cust_code = in_cust_code;
  ...
BEGIN
  OPEN c_task_list_cur('WM');
  ...
  CLOSE c_task_list_cur;
  OPEN c_task_list_cur('TG');
  ...
```

Explicit Cursor

Locking Processed Rows

- Used to prevent from rows modifications by other transactions
- Locks only rows from cursor working set selected by query filter
- Rows remains locked until current transaction finished
- Not needed to get read consistent rows snapshot
- But if cursor rows are updated or deleted inside cursor processing
  - rows should be locked by cursor FOR UPDATE option with NOWAIT, WAIT or WAIT *seconds* option

```
DECLARE
  CURSOR c_stts_cur IS
    SELECT *.s, task_name
      FROM task_sttus_prc s
      JOIN task_list_prc l
        ON (l.task_code = s.task_code)
     WHERE cust_code = in_cust_code
    FOR UPDATE OF task_flags_val NOWAIT;
  ...
  FOR l_stts_record IN c_stts_cur LOOP
   ...
   UPDATE task_sttus_prc
      SET task_flags_val = l_flags
    WHERE CURRENT OF c_stts_cur;
  END LOOP;
  ...
```

Topic Agenda

PL/SQL Package

- Overview
- Architecture
- Example
- Description Header Comments
- Features
  - Initialization
  - Session Persistent State
  - Overloading
  - Forward Declaration
  - Bodiless Package

Overview

- Package is a schema object that groups logically related
  - PL/SQL types, cursors, exceptions, variables, and subprograms
  - Oracle server reads all objects from package into memory at once at first content call
  - Subsequent call to any package component not require disk I/O - performance
  - Less object in database to create, migrate and maintain - simplification
- Have two parts: mandatory specification, optional body
- Creation and compilation are separate for parts
- Hiding information – only specification is public
- Cannot be called and parameterized itself
- Cannot be nested/declared in another PL/SQL code
  - only created as database schema object (exceptions are Forms or Reports modules)
- Maximum size 64kB on Unix (check PL/SQL limits)
- Bigger then 10kB are not recommended

Architecture

- Specification and body content is similar to PL/SQL block declaration section
- Specification
  - Is public interface to package
  - Use **g_** prefix for public/global variables names ★★★
  - Use public declarations only if abs. necessary ★★★ (dependencies)
  - Subprograms are declared here without body (only header finished with ;)
- Body
  - Is private package implementation
  - Subprograms are declared here with body
  - All declared in specification subprograms have to be implemented in body

```
Package Specification
  public/global types, exceptions, cursors and variables declarations
  public subprograms declarations without body

Package Body
  private types, exceptions, cursors and variables
  private subprograms implementation with body
  public subprograms implementations with body
```

Example

Specification

```
CREATE OR REPLACE PACKAGE pkg_appl_log
IS
  TYPE t_num_list IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
  CURSOR c_log_globl_cur IS SELECT * FROM appl_log_plc;
  g_num_list t_num_list;

  ...

  PROCEDURE pro_add_log_msg(in_prcss_name IN VARCHAR2,
                            in_msg        IN VARCHAR2);

  FUNCTION fn_delet_old(in_rtnsn_days IN NUMBER) RETURN NUMBER;

END pkg_appl_log;
```

Example

Body

```
CREATE OR REPLACE PACKAGE BODY pkg_appl_log
IS
  TYPE t_name_list IS TABLE OF VARCHAR2(100) INDEX BY PLS_INTEGER;
  CURSOR c_parm_cur IS
    SELECT * FROM appl_parm_prc
     WHERE parm_class = 'LOG';
  l_name_list t_name_list;
  l_parm_record c_parm_cur%ROWTYPE;

  PROCEDURE pro_add_log_msg(in_prcss_name IN VARCHAR2,
                            in_msg        IN VARCHAR2)
  IS
  BEGIN
    ...
  END;

  FUNCTION fn_delet_old(in_rtnsn_days IN NUMBER) RETURN NUMBER
  IS
  BEGIN
    ...
  END;

END pkg_appl_log;
```

Description Header Comments ★★★

- Are mandatory in package specification and body
  - With change history
- All package subprograms should have description headers as well
  - With change history, parameters and exit codes (described in Basic Training)

```
CREATE OR REPLACE PACKAGE BODY pkg_appl_log IS
-----------------------------------------------------------------------------------------------
-- Package      : pkg_appl_log                                                                --
-- Author       : Jan Kowalski (jan.kowalski@pg.com)                                          --
-- Date         : 01-JAN-2014                                                                 --
-- Purpose      : Logging messages to database table                                          --
-- Version      : 1.0.1                                                                       --
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-- Change History                                                                             --
-- Date         Programmer         Description                                                --
-- ----------- ----------------- ------------------------------------------------ --
-- 01-JAN-2014 Jan Kowalski       Initial Version 1.0.0                                      --
-- 23-MAY-2014 Piotr Nowak        Version 1.0.1 - Corrected bug in SELECT stmt               --
-----------------------------------------------------------------------------------------------
...
```

Features

Session Persistent State

- State of cursor and variable is persistent on session
  - If declared inside package specification or body but only outside subprograms
  - Subsequent package call do not reinitialize this objects
  - State is stored on session – reinitialized after reconnect
  - Can consume a lot of memory for session
  - Can be removed or reinitialized during session time using one from

```
dbms_session.reset_package; --frees memory allocated for all packages states in session
dbms_session.modify_package_state(dbms_session.free_all_resources) --frees all PL/SQL session memory
dbms_session.modify_package_state(dbms_session.reinitialize) --only variables reinit - performance
```

  - Can be turned off by adding into package after IS keyword

```
PRAGMA SERIALLY_REUSABLE;
```

Features

Session Persistent State

- Example

```
CREATE OR REPLACE PACKAGE BODY pkg_appl_log
IS
  ...
  l_log_rowid ROWID;

  PROCEDURE pro_new_log
   ...
    INSERT INTO appl_log_plc ...
      RETURNING ROWID INTO l_log_rowid;
  END;

  PROCEDURE pro_add_log_msg(in_msg IN VARCHAR2)
    ...
    UPDATE appl_log_plc
       SET log_msg = log_msg || CHR(10) || in_msg)
     WHERE ROWID = l_log_rowid;
  END;
  ...
END pkg_appl_log;
```

PL/SQL Package

Features

Initialization

- Variable initialization when value known only on runtime

```
CREATE OR REPLACE PACKAGE BODY pkg_appl_log
IS
  l_log_tbl_name VARCHAR2(30);
  PROCEDURE pro_add_log_msg(in_prcss_name IN VARCHAR2,
                            in_msg        IN VARCHAR2)
  IS
  BEGIN
    EXECUTE IMMEDIATE 'UPDATE ' || l_log_tbl_name ||
    ...
BEGIN
  SELECT parm_val
    INTO l_log_tbl_name
    FROM appl_parm_prc
   WHERE parm_name = 'log_table_name';
END pkg_appl_log;
/
```

Features

Overloading

- Many subprogram versions with the same name in one namespace
- Available inside package or subprogram declarations section
- Not available inside database schema
- Overloaded subprogram versions headers must have min. 1 difference
  - Number of parameters
  - Type of parameter
  - Returned value type (for functions)
- Overloaded subprogram must be unambiguously called
  - Parameters providing form must be enough to choose one from many declarations
  - If one version can't be chosen server generate runtime error

```
PLS-00307: too many declarations of … match this call
```

Features

Overloading

- Example - difference in number of parameters

```
CREATE OR REPLACE PACKAGE BODY pkg_cust_pkg
IS

  PROCEDURE pro_add_cust(in_cust_name cust_dim.cust_name%TYPE,
                         in_cust_skid cust_dim.cust_skid%TYPE)
  IS
  BEGIN
    INSERT INTO cust_dim (cust_skid, cust_name)
      VALUES (in_cust_skid, in_cust_name);
  END;

  PROCEDURE pro_add_cust(in_cust_name cust_dim.cust_name%TYPE)
  IS
  BEGIN
     INSERT INTO cust_dim (cust_skid, cust_name)
      VALUES (cust_dim_skid_seq.NEXTVAL, in_cust_name);
  END;

END dept pkg;
/

BEGIN
  pro_add_cust('Walmart',8);
  pro_add_cust('Kroger');
END;
/
```

Features

Forward Declaration

- Called subprogram must be declared before calling
- If its not possible (cross recursion or name order)
- Use forwarding (not needed for public subprograms)

```
CREATE OR REPLACE PACKAGE BODY ...
...
  PROCEDURE pro_calc_rating(...);

  PROCEDURE pro_award_bonus(...)
    ...
    pro_calc_rating(...)
    ...
  END;

  PROCEDURE pro_calc_rating(...)
    ...
    pro_award_bonus(...);
    ...
  END;
```

Features

Bodiless Package

- Do not created body if no subprograms in specification
- Bodiless packages
  - Can be shared between many applications
  - Can contain public constants and types – best if never modified
  - Use **g_const_** prefix in global constant name

```
CREATE OR REPLACE PACKAGE pkg_global_const
IS
  g_const_mile_2_kilo  CONSTANT NUMBER  :=  1.6093;
  g_const_kilo_2_mile  CONSTANT NUMBER  :=  0.6214;
  g_const_yard_2_meter CONSTANT NUMBER  :=  0.9144;
  g_const_meter_2_yard CONSTANT NUMBER  :=  1.0936;
END pkg_global_const;
```

  - e.g. dbms_types used in dbms_sql.describe_columns

Topic Agenda

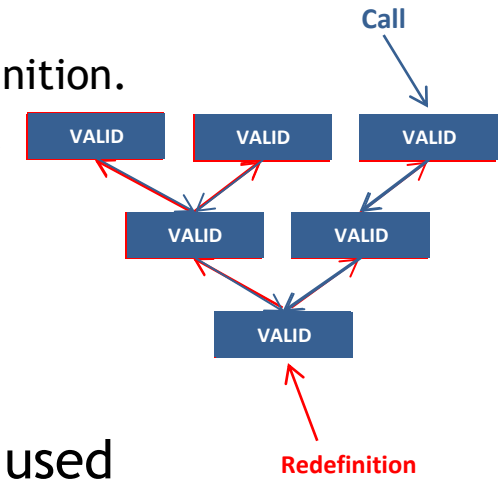
Dependencies

- Overview
- Checking
- When Using Packages
- Fine grained in 11g

Overview

- If database object references another object than is dependent of it
- If both are in the same database – local dependency
- Local dependency is automatically handled
  - Object is invalidated automatically after master object redefinition.
  - Invalid object is recompiled automatically on subsequent call.
  - If recompilation is successful object is used without any error.
- Recompilation raises CPU load
  - do nod modify base object during work hours
    or all directly and indirectly dependent object will be invalidated
- For remote dependency compilation timestamp is used
  - to detect change during call and invalidate in first call when error message is generated.
  - Second call recompile invalidated object
- Can be changed to signature mode

```
EXECUTE IMMEDIATE 'ALTER SESSION SET REMOTE_DEPENDENCIES_MODE = SIGNATURE';
```



Checking

- INVALID status

```
SELECT status FROM user_objects WHERE object_name = 'PRO_LOAD_TBL';
```

- Direct dependency

```
SELECT name, type, referenced_name, referenced_type
  FROM user_dependencies
 WHERE referenced_name = 'APPL_LOG_PLC';
```

- All dependent on selected

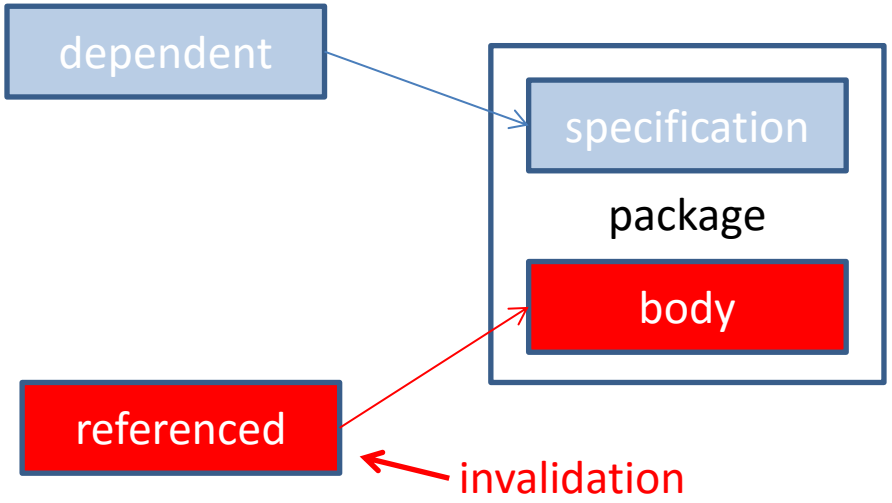
```
SQL> EXECUTE deptree_fill('TABLE', 'schema', 'APPL_LOG_PLC')
SQL> @?/rdbms/admin/utldtree.sql

SQL> SELECT nested_level, type, name
       FROM deptree
      ORDER BY seq#;
```

Package

Simplified Dependency Hierarchy

- References to package are based on specification
- Body dependency invalidates only body
- So cascade invalidation are stopped on package
- Use packages to break dependency hierarchy
- Put into specification only what is absolutely needed



11g Fine Grained Dependencies

- Object referencing table is not invalided if
  - references only no modified columns
- Reference tables trough the view ★
  - to minimize invalidation
- Package specification modification
  - do not invalidate dependent objects if adding
  - new subprogram as last in package specification ★

Topic Agenda

Oracle Supplied Packages

- List
- Usage Examples

Oracle Provided Database Packages List

- 239 can be used by applications in Oracle 11g R2
  - Database PL/SQL Packages and Types Reference
- Not all installed by default
- Most frequently used packages have DBMS_ or UTL_ name prefix

DBMS_ALERT, **DBMS_APPLICATION_INFO**, DBMS_AQ, DBMS_ASSERT, DBMS_OBFUSCATION_TOOLKIT, **DBMS_OUTPUT**, DBMS_PARALLEL_EXECUTE, DBMS_RLS, DBMS_FGA, DBMS_LDAP, DBMS_LDAP_UTL, DBMS_LOB, **DBMS_LOCK**, DBMS_RANDOM, DBMS_SCHEDULER, DBMS_SESSION, DBMS_STAT_FUNCS, DBMS_STATS, DBMS_UTILITY

DBMS_DATAPUMP, DBMS_COMPARISON, DBMS_COMPRESSION, DBMS_FILE_TRANSFER, DBMS_METADATA, DBMS_METADATA_DIFF, DBMS_PIPE, DBMS_PREPROCESSOR, DBMS_REDEFINITION, DBMS_REDACT, DBMS_RESUMABLE, DBMS_SQL, DBMS_SQLTUNE, DBMS_TRACE, DBMS_TRANSACTION, DBMS_WM, DBMS_XA, DBMS_XML..., DBMS_XPLAN

UTL_COMPRESS, UTL_ENCODE, UTL_FILE, UTL_HTTP, UTL_I18N, UTL_INADDR, UTL_LMS, UTL_MAIL, UTL_MATCH, UTL_NLA, UTL_RAW,
UTL_RECOMP, UTL_REF, UTL_SMTP, UTL_SPADV, UTL_TCP, UTL_URL

Oracle Provided Database Packages

Usage Examples

- dbms_output

  Debug buffer is collection with maximum 2147483647 elements of VARCHAR2(32767) type

  SQL> set serveroutput on -- on sqlplus and sqldeveloper to see buffer

```
dbms_output.enable(100000);
```
← NULL - no bytes limit - default 20000 bytes

```
dbms_output.put( TO_CHAR(SYSDATE, ' YYYY-MM-DD HH24:MI:SS')|| 'STEP 040 - ');
dbms_output.put_line('Loading Table. ');
dbms_output.new_line;
dbms_output.disable;
...
dbms_output.get_line(lines => l_msgs, numlines => l_line_cnt);
dbms_output.get_lines(line => l_msg, status => l_sttus);
```
←(0 - ok, 1 - no lines)

- dbms_lock

```
dbms_lock.allocate_unique(lockname => 'APPLCK-SC88', lockhandle => l_handle);
dbms_lock.request(lockhandle => l_handle, lockmode => dbms_lock.x_mode, timeout => 60);
...
dbms_lock.release(l_lock_handle);
dbms_lock.sleep(30);
```

- dbms_application_info

```
dbms_application_info.set_module(module_name =>'PRO_LOAD_TBL', action_name => NULL);
dbms_application_info.set_action(action_name => 'STEP050');
dbms_application_info.set_client_info
```

Topic Agenda

Database Triggers

- Overview
- DML Trigger Examples

Overview

- Automatic executed on defined event PL/SQL block
- Place header comments into triggers ★★
- Use object name in name and _trbaiiad sufix ★★
- Trigger types:

  – DML trigger - for statement or for each row
    - before or after INSERT, UPDATE, DELETE on table
    - instead of INSERT, UPDATE, DELETE on view

  use DML compound trigger to define all actions on table in one trigger

  – Event trigger - database or schema level
    - before or after selected from many DDL command types
    - before or after startup, shutdown, login, logoff, after error

DML Trigger Examples

- Automatic data supplementation

```
CREATE OR REPLACE TRIGGER prod_dim_trbi
  BEFORE INSERT ON prod_dim FOR EACH ROW
  WHEN (NEW.prod_skid IS NULL)
BEGIN
  :NEW.prod_skid :=  prod_dim_skid.NEXTVAL;
END;
```

- Related data auto-modification or replication

```
CREATE OR REPLACE TRIGGER inven_item_prc_trbiud
  BEFORE INSERT OR UPDATE OR DELETE ON inven_item_prc FOR EACH ROW
BEGIN
  IF DELETING THEN
    UPDATE inven_tot_prc SET item_cnt = item_cnt - :OLD.item_cnt
     WHERE item_categ = :OLD.item_categ;
  ELSE
    UPDATE inven_tot_prc SET item_cnt = item_cnt + :NEW.item_cnt
     WHERE item_categ = :NEW.item_categ;
END;
```

DML Trigger Examples

- Access Control

```
CREATE OR REPLACE  TRIGGER user_uprc_trbiu
  BEFORE INSERT OR UPDATE ON user_uprc
  WHEN (TO_CHAR(SYSDATE, ' DAY ') IN (' SAT ', ' SUN '))
BEGIN
  RAISE_APPLICATION_ERROR(-20321, ' Do not define user on weekend. ');
END;
```

- Auditing

```
CREATE OR REPLACE TRIGGER config_prc_traiud
  AFTER INSERT OR UPDATE OR DELETE ON config_prc FOR EACH ROW
BEGIN
  INSERT INTO audit_plc (time_stamp, usr, name, old_val, new_val, ...)
    VALUES (SYSDATE, USER, :OLD.parm_name, :NEW.parm_val, :NEW.param_val, ...);
END;
```

Topic Agenda

PL/SQL security

- Execution Access
- Secured Application Roles

Execution Access

- Object execution privilege needed for other users

```
GRANT EXECUTE ON pkg_sale_fct_load TO adwg_cba_etl;
```

- Anonymous block units executed with invoker's rights
- Stored units by default executed with definer's rights
  - If security need it change to invoker's rights, e.g.

```
CREATE OR REPLACE PACKAGE pkg_name
  AUTHID CURRENT_USER
IS
```

  - If object name in code is not prefixed with schema name
    - Invoker rights unit uses object from invoker schema
    - Definer rights unit uses object from definer schema

Secured Application Roles

- Use secured database roles in invoker rights model
  - Create role secured by PL/SQL role security package

```
CREATE ROLE cba_role IDENTIFIED USING pkg_role_scrty;
```

  - Create role security package

```
CREATE OR REPLACE PACKAGE BODY pkg_role_scrty
IS
  PROCEDURE set_cba_role
  ...
    IF ... THEN
      dbms_session.set_role('cba_role');
    ...
```

  - Security package procedure turn on role conditionally
  - Grant definer object privileges to role and create synonyms to objects
  - Turn on role before invoker rights access to other schema objects

```
pkg_role_scrty.set_cba_role;
```

Topic Agenda

Manage PL/SQL Units

- Maintain Source Code
- Resolving INVALID Status
- Compilation Flags

Source Code

- Is stored in database even for objects created with compilation errors
- Can be restored from database

```
SELECT text FROM user_source
 WHERE name = '<name>' AND type = '<type>'
 ORDER BY line;

SELECT dbms_metadata.get_ddl(object_type => 'PROCEDURE'
                             object_name => 'PRO_LOAD_TBL')
  FROM dual;
```

- Can be transported between schemas using

  `dbms_metadata.get_xml dbms_metadata.put or expdp inpdp`

- Storing in files ★★
  - plsql_*<unit name>*_ vX_XXX_XXX.sql (X-numbers: version nr, maintenance nr, component nr)
  - Each unit in separate files, package specification and body in one file.

Resolving INVALID Status

- Immediately after CREATE execute ★★

```
SHOW ERRORS
```

  - Later you can execute

```
SHOW ERRORS PROCEDURE|FUNCTION|PACKAGE|PACKAGE BODY|TRIGGER <name>
SELECT text, line, position FROM user_errors
 WHERE name = '<name>' AND type = '<type>'
 ORDER BY sequence;
```

- Find problematic units

```
SELECT object_name, object_type, status
  FROM user_objects
 WHERE status = 'INVALID';
```

  - Compiled with errors
  - Invalided because of dependency
  - Object privilege revoked

- Unit recompilation

```
ALTER PROCEDURE|FUNCTION|TRIGGER <name> COMPILE;
ALTER PACKAGE <name> COMPILE SPECIFICATION|BODY|PACKAGE;
```

  - Compile all invalid in schema

```
utl_recmp.recomp_parallel(threads => 8, schema => 'ADWG_CBA');
```

  - Avoid during processing – to save performance

Compilation Flags

- Get from database dictionary

```
SELECT name, type,
       plsql_code_type, plsql_optimize_level
       plsql_debug, plsql_warnings, nls_length_semantics
       plsql_ccflags, plscope_settings
FROM   user_plsql_object_settings;
```

- ALTER SESSION SET parameter before compilation

```
PLSQL_CODE_TYPE = INTERPRETED|NATIVE
PLSQL_OPTIMIZE_LEVEL = 0|1|2|3
PLSQL_DEBUG = TRUE;
PLSQL_WARNINGS = 'ENABLE:SEVERE','DISABLE:PERFORMANCE', 'ERROR:05003';
NLS_LENGTH_SEMANTICS = 'CHAR‘;
PLSQL_CCFLAGS = ‘platform:ADW3U‘
PLSCOPE_SETTINGS='IDENTIFIERS:ALL';
```

- To recompile procedure without flags modification

```
ALTER PROCEDURE pro_load_tbl COMPILE REUSE SETTINGS;
```

Q & A

www.lingaro.com

PL/SQL Resources

- Oracle Database Documentation Library

  http://docs.oracle.com/cd/E11882_01/index.htm

  – PL/SQL Language Reference

    http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/toc.htm

  – PL/SQL Packages and Types Reference

    http://docs.oracle.com/cd/E11882_01/appdev.112/e40758/toc.htm

- OTN

  http://www.oracle.com/technetwork/database/application-development/index-101230.html

- O'Reilly

  – **Oracle PL/SQL Programming**

    ISBN:978-0-596-51446-48