Oracle PL/SQL IV

Exceptions

Packages

Exception Handling

- Remember it is optional

[DECLARE]

BEGIN

[EXCEPTION]

END;

More on Exception Handling

- Exceptions are pre-defined or designer-specified occurrences during the operation of a PL/SQL block
- Exceptions are named by user or by Oracle
- Exceptions are raised internally by PL/SQL or explicitly by designer using RAISE keyword
- A routine in the Exception section will then be called

User-defined

- Declare a name for the exception
- Identify point to raise exception
- Defining code to fire when raised

```
DECLARE
Salary_too_high     EXCEPTION;
Invalid_tax_code    EXCEPTION;
….
```

RAISING

- Once control has passed to the Exception section it cannot be returned to the block that raised it

```
RAISE exception_name
```

```
IF v_salary>v_max then
        RAISE salary_too_high;
End if
```

Jump to the exception section user defined as salary_too_high

EXCEPTION Section

```
EXCEPTION
  WHEN salary_too_high
       then "PL/SQL statements here";
  WHEN another_error OR yet_another
       then "PL/SQL statements here";
  WHEN OTHERS                      -- Oracle defined
       then "PL/SQL statements here";
END;
```

Note OTHERS will trap any other error that you have not accounted for

Example

```
set serveroutput on
DECLARE
v_bonus   number;
null_bonus_alert  exception;
BEGIN
select bonus into v_bonus from personnel where snum=3813;
if v_bonus is null then
                raise null_bonus_alert;
end if;
EXCEPTION
  when null_bonus_alert then
  dbms_output.put_line('This exmployee really should get a bonus!');
END;
/
```

RAISE_APPLICATION _ERROR

- We have seen this already
- Error_number is a negative integer in the range -20000 to -20999
- Error_message is a character string up to 512 bytes

Raise_application_error(*error_number*,*error_message*)

Example and error generated

```
DECLARE
v_bonus   number;
BEGIN
select bonus into v_bonus from personnel where snum=3813;
if v_bonus is null then
  raise_application_error(-20111,'For goodness sake give him a
    bonus!!');
end if;
END;
/
declare
*
ERROR at line 1:
ORA-20111: For goodness sake give him a bonus!!
ORA-06512: at line 6
```

Common Pre-defined Exceptions

Oracle Error	Exception	description
ORA-00001	DUP_VAL_ON_INDEX	PK violation
ORA-01403	NO_DATA_FOUND	No records
ORA-01422	TOO_MANY_ROWS	> 1 row
ORA-01476	ZERO_DIVIDE	
ORA-01722	INVALID_NUMBER	Can't convert

Note: there are others too!

Example using pre-defined method

```
--This example returns salaries for branch 10
DECLARE
v_salary personnel.salary%type;
BEGIN
select salary into v_salary from personnel where div=10;
EXCEPTION
when too_many_rows  -- this is the pre-defined exception
then raise_application_error (-20001, ' Did not expect so many');
END;
/
declare
*
ERROR at line 1:
ORA-20001:  Did not expect so many
ORA-06512: at line 7
```

If you don't trap it Oracle takes over!

```
--in this example branch 40 has no staff yet!
DECLARE
v_salary personnel.salary%type;
BEGIN
select salary into v_salary from personnel where div=40;
EXCEPTION
when too_many_rows  -- this is the pre-defined exception
then raise_application_error (-20001, ' Did not expect so many');
END;
/
declare
*
ERROR at line 1:
ORA-01403: no data found – occurs as div 40 has no staff!
ORA-06512: at line 4
```

This is NO_DATA_FOUND exception

PACKAGES

What are they?

- A collection of PL/SQL objects that are logically grouped together to form one unit
- They can contain:
  - Procedures, functions
  - Cursors, variables, Constants
  - Tables
  - Exceptions
- Typically, they may contain all routines to process purchase orders, for example.

Package structure

- Has 2 parts:
  - Package Specification
    - Declares public procedures etc
    - Other programs can access them outside the package
  - Package Body
    - Implements the public procedures etc but also may specify private procedures, functions etc
    - The private units are only accessible within the scope of the package itself
    - All units declared in specification MUST be implemented in the body

Package Specification example

```
CREATE OR REPLACE PACKAGE package_name
IS
    PROCEDURE sal_raise
              (p1 NUMBER, p2 NUMBER);
  ----------------------------------------
    FUNCTION div_bal
              (div_no IN NUMBER)
    RETURN NUMBER;
  ----------------------------------------
END package_name;     -- not necessary to name package here
                                -- just for clarity
```

Note there is no PL/SQL executable code

Package Body

```
CREATE OR REPLACE PACKAGE BODY package_name
IS
    PROCEDURE sal_raise (p1 NUMBER, p2 NUMBER)
    IS
    BEGIN
            update staff set salary=salary*1.1 where div=p2;
    END sal_raise;
 ----------------------------------------
    FUNCTION div_bal (div_no IN NUMBER)
    RETURN NUMBER
    IS
    bal       number;
   BEGIN
            select sum(salary) into bal from staff where div=div_no;
    RETURN bal;
    END div_bal;
END package_name;
```

How do we use them?

- DROP PACKAGE package_name
  - Will remove specification and body
- DROP PACKAGE BODY package_name
  - Will only remove the body
- To run/access an element of a package body

```
Execute package_name.element
```

```
empName:=package_name.getName(empID);
```

The package The function element The parameter

Global variables

```
CREATE OR REPLACE PACKAGE BODY stdcomp
IS
gcompany  NUMBER;  -- global to the package
   PROCEDURE setcomp (xcompany IN NUMBER) IS
   BEGIN
            gcompany:=xcompany;
   END setcomp;
  ----------------------------------------
   FUNCTION getcomp
   RETURN NUMBER IS
   BEGIN
            RETURN NVL(gcompany,0);
   END getcomp;
  ----------------------------------------
END stdcomp;
```

Instantiation and persistence

- Instantiation occurs each time you connect to the database
- So any state of your current session is lost when this happens
- And packages are instantiated again
- The default behaviour of a package is to maintain its state once it has been instantiated throughout the life of the session

Persistence

```
CREATE OR REPLACE PACKAGE pack1 IS
V1 NUMBER:=1;
    Procedure proc1;
End pack1;

CREATE OR REPLACE PACKAGE BODY pack1 IS
V2 NUMBER:=2;
    Procedure proc1 IS
    V3        NUMBER:=3;
    BEGIN
              v1:=v1+1;
              v2:=v2+2;
              v3:=v3+3;
              DBMS_OUTPUT.PUT_LINE('v1 = '||v1);
              DBMS_OUTPUT.PUT_LINE('v2 = '||v2);
              DBMS_OUTPUT.PUT_LINE('v3 = '||v3);
    END proc1;
END pack1;
```

execution →

	1st	2nd	3rd
v1	2	3	4
v2	4	6	8
v3	6	6	6

Execute pack1.proc1 – do it 3 times!

Pragma SERIALLY_REUSABLE

- Causes the PL/SQL runtime to discard the state of the package. So instantiation occurs each time it is invoked
- Pragma serially_reusable needs to be applied to both the SPECIFICATION and the BODY
- Now, execution 3 times of the previous code would be as follows

v1	2	2	2
v2	4	4	4
v3	6	6	6

```
CREATE OR REPLACE PACKAGE pack1
IS
Pragma serially_reusable;
V1          NUMBER:=1;
            Procedure proc1;
End pack1;
```

Overloading

- Sub-programs in a package body can have the same names so long as parameter lists are not the same.
- E.g. the TO_CHAR function in SQL takes either a number or a date.
- The appropriate function is called depending on what the user enters.

Overload example

```
CREATE OR REPLACE PACKAGE overload IS
      Function sal_return (p_detail NUMBER)
            Return NUMBER;
      Function sal_return (p_detail VARCHAR2)
            Return NUMBER;
End overload;

CREATE OR REPLACE PACKAGE BODY overload IS
      Function sal_return (p_detail NUMBER)
            Return NUMBER IS
    v_salary NUMBER;
    BEGIN
                  Select salary into v_salary from staff where snum=p_detail;
                  Return v_salary;
    END sal_return;
```


Code Continues on next slide

Continued …..

Same name different parameter datatype

Different attribute

```
Function sal_return (p_detail VARCHAR2)
            Return NUMBER IS
      v_salary NUMBER;
BEGIN
            Select salary into v_salary from staff where surname=p_detail;
            Return v_salary;
END sal_return;
END overload;
```

```
SELECT overload.sal_return(3488) from dual ;    -- this would call the first
                                                    sal_return
SELECT overload.sal_return('STYLES') from dual;  -- this would call the second
                                                    sal_return
```

Legal and illegal packages!

```
CREATE OR REPLACE PACKAGE ovtest
IS
Function cat  (n1 NUMBER, c2 VARCHAR2)
        Return VARCHAR2;
----------------------------------------
Function cat (c1 VARCHAR2, n2 NUMBER)
        Return VARCHAR2;
End ovtest;
```

LEGAL as positions are different

```
CREATE OR REPLACE PACKAGE ovtest
IS
Function cat  (n1 NUMBER, c2 VARCHAR2)
        Return VARCHAR2;
----------------------------------------------------
    Function cat (n1 INTEGER, c2 CHAR)
        Return VARCHAR2;
End ovtest;
```

ILLEGAL as parameters in both are compatible

Summary

- Exceptions
  - RAISE_APPLICATION_ERROR
  - RAISE
  - User or pre-defined
  - Exception WHEN ……
- Packages
  - Specification and Body
  - Executing and calling
  - Instantiation and persistence
  - Global and local variables

READING

- Connolly/Begg (4th ed) 8.2.5, 8.2.6
- Shah (Ch 10,12)
- Morrison/Morrison “Guide to ORACLE 10g” Ch.4 & 9 – selected bits
- Casteel, J (2003). Oracle 9i Developer: PL/SQL Programming – chapter 3 & 6.