Programming in Oracle with PL/SQL

1

Why/When PL/SQL

- PL/SQL allows SQL to be combined with programming language constructs (e.g., if/else, loops, function declarations)
- This is generally "lighter-weight" than connecting with JDBC, since it is run within the database
- PL/SQL functions can even be called from a query!!

2

PL/SQL Blocks

- There are two types of block structures for PL/SQL.
- Anonymous blocks: have no name
  - can be written and executed immediately in SQLPLUS
  - can be used in a trigger
- Named PL/SQL blocks:
  - functions
  - procedures
- Important: Always put a new line with only a / at the end of a block, so that Oracle will compile it.

3

Block Structure for Anonymous PL/SQL Blocks

DECLARE (optional)
Declare PL/SQL objects to be used within this block

BEGIN (mandatory)
Define the executable statements

EXCEPTION (optional)
Define the actions that take place if an error arises

END; (mandatory)

/

4

Declaring PL/SQL Variables

Syntax

```
identifier [CONSTANT] datatype [NOT NULL]
    [:= | DEFAULT expr];
```

Examples

```
Declare
  birthday   DATE;
  age        NUMBER(2) NOT NULL := 27;
  name       VARCHAR2(13) := 'Levi';
  magic      CONSTANT NUMBER := 77;
  valid      BOOLEAN NOT NULL := TRUE;
```

5

Declaring Variables with the %TYPE Attribute

Examples

```
...
  sname          Sailors.sname%TYPE;
  fav_boat       VARCHAR2(30);
  my_fav_boat    fav_boat%TYPE := 'Pinta';
...
```

6

Creating a PL/SQL Record

Declare variables to store the name, id, age and rating of a new sailor.

Example

```
...
  TYPE sailor_record_type IS RECORD
    (sname     VARCHAR2(10),
     sid       VARCHAR2(9),
     age       NUMBER(3),
    rating     NUMBER(3));
  sailor_record   sailor_record_type;
...
```

7

The %ROWTYPE Attribute

Declare a variable to store the same information about a reservation as it is stored in the Reserves table.

```
reserves_record   reserves%ROWTYPE;
```

8

SELECT Statements in PL/SQL

```
DECLARE
  v_sname  VARCHAR2(10);
  v_rating NUMBER(3);
BEGIN
  SELECT  sname, rating
    INTO  v_sname, v_rating
    FROM  Sailors
   WHERE  sid = '112';
...
END;
```

- INTO clause is required.
- Query must return exactly one row.
- Otherwise, a NO_DATA_FOUND or TOO_MANY_ROWS exception is thrown

9

Suppose we have the following table:

```
create table mylog(
     who varchar2(30),
     logon_num number
);
```

- Want to keep track of how many times someone logged on
- When running, increment logon_num, if user is already in table. Otherwise, insert user into table

10

Solution

```
declare
 cnt  NUMBER;
begin
select count(*)
 into cnt
 from mylog
where who = user;

if cnt > 0 then
   update mylog
      set logon_num = logon_num + 1
    where who = user;
else
   insert into mylog values(user, 1);
end if;
commit;
end;
/
```

11

Some Notes

- We used commit at the end, since the actions taken should form a "single unit"
- Can also use rollback if we encounter an exception
- PL/SQL does not commit by default
- Note syntax of IF:
  - IF *condition* THEN
  - ELSIF …
  - ELSE …
  - END IF

MUST BE MISSING AN E!!!

12

IF-THEN-ELSIF Statements

```
. . .
IF rating > 7 THEN
  v_message := 'You are great';
ELSIF rating >= 5 THEN
  v_message := 'Not bad';
ELSE
  v_message := 'Pretty bad';
END IF;
. . .
```

13

SQL Cursor Attributes

Using SQL cursor attributes, you can test the outcome of your SQL statements.

SQL%ROWCOUNT	Number of rows affected by the most recent SQL statement (an integer value).
SQL%FOUND	Boolean attribute that evaluates to TRUE if the most recent SQL statement affects one or more rows.
SQL%NOTFOUND	Boolean attribute that evaluates to TRUE if the most recent SQL statement does not affect any rows.
SQL%ISOPEN	Always evaluates to FALSE because PL/SQL closes implicit cursors immediately after they are executed.

14

Solution (2)

```
begin

update mylog
   set logon_num = logon_num + 1
 where who = user;

if SQL%ROWCOUNT = 0 then
   insert into mylog values(user, 1);
end if;

commit;
end;
/
```

15

Simple Loop (Similar to While Until)

```
create table number_table(
     num NUMBER(10)
);
```

```
DECLARE
  i      number_table.num%TYPE := 1;
BEGIN
  LOOP
    INSERT INTO number_table
      VALUES(i);
    i := i + 1;
    EXIT WHEN i > 10;
  END LOOP;
END;
```

16

FOR Loop

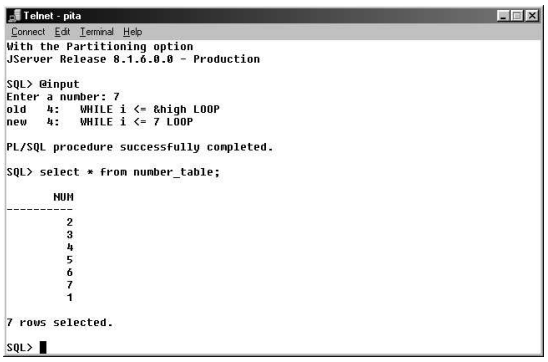
```
DECLARE
  i      number_table.num%TYPE;
BEGIN
  FOR i IN 1..10 LOOP
    INSERT INTO number_table
      VALUES(i);
  END LOOP;
END;
```

17

WHILE Loop

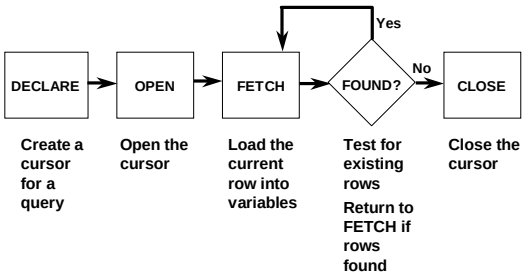
```
ACCEPT high PROMPT 'Enter a number: '

DECLARE
i               number_table.num%TYPE:=1;
BEGIN
  WHILE i <= &high LOOP
     INSERT INTO number_table
     VALUES(i);
     i := i + 1;
  END LOOP;
END;
```

18



19

Looping over results of a query using an Explicit Cursor



20

Explicit Cursor Attributes

Obtain status information about a cursor.

Attribute	Type	Description
%ISOPEN	Boolean	Evaluates to TRUE if the cursor is open.
%NOTFOUND	Boolean	Evaluates to TRUE if the most recent fetch does not return a row.
%FOUND	Boolean	Evaluates to TRUE if the most recent fetch returns a row; complement of %NOTFOUND
%ROWCOUNT	Number	Evaluates to the total number of rows returned so far.

21

Example

```
DECLARE
   num  number_table.num%TYPE;
   cursor c is
      select * from number_table;

BEGIN
   open c;
   fetch c into num;
   loop
     dbms_output.put_line(c%ROWCOUNT ||
                          '-th Value: ' ||
                          num);
     fetch c into num;
     exit when c%NOTFOUND;
   end loop;
   close c;
end;
```

22

Printing Output

- You need to use a function in the DBMS_OUTPUT package in order to print to the output
- The output is actually buffered
- If you want to see the output on the screen, you must type the following (before starting):
  - set serveroutput on format wrapped size 1000000
- Then print using
  - dbms_output. put_line*(your_string);*
  - dbms_output.put(*your_string*);

23

Cursor Looping

```
DECLARE
   num_row  number_table%ROWTYPE;
   cursor c is select * from number_table;

BEGIN
   for num_row in c loop -- opens and fetches

     dbms_output.put_line(c%ROWCOUNT ||
                          '-th Value: ' ||
                          num_row.num);
   end loop; -- closes
end;
/
```

24

Trapping Oracle Server Errors

- Reference the standard name in the exception-handling routine.
- Sample predefined exceptions:
  - NO_DATA_FOUND
  - TOO_MANY_ROWS
  - ZERO_DIVIDE
- When handling an exception, consider performing a rollback

25

```
DECLARE
   num_row  number_table%ROWTYPE;
BEGIN
   select *
   into num_row
   from number_table;
   dbms_output.put_line(1/num_row.num);

EXCEPTION
   WHEN NO_DATA_FOUND THEN
     dbms_output.put_line('No data!');
   WHEN TOO_MANY_ROWS THEN
     dbms_output.put_line('Too many!');
   WHEN OTHERS THEN
     dbms_output.put_line(SQLERRM);
end;
```

26

User-Defined Exception

```
DECLARE
   e_number1  EXCEPTION;
   cnt        NUMBER;
BEGIN
   select count(*)
   into cnt
   from number_table;

   IF cnt = 1 THEN RAISE e_number1;
   ELSE dbms_output.put_line(cnt);
   END IF;

EXCEPTION
   WHEN e_number1 THEN
     dbms_output.put_line('Count = 1');
end;
```

27

Functions and Procedures

- Up until now, our code was in an anonymous block
- It was run immediately
- Useful to put code in a function or procedure so it can be called several times

28

Creating Procedures

```
CREATE [OR REPLACE] PROCEDURE
procedure_name
 [(parameter1 [mode1] datatype1,
  parameter2 [mode2] datatype2,
  . . .)]
IS|AS
PL/SQL Block;
```

29

Modes

- Modes:
  - IN: procedure must be called with a value for the parameter. Value cannot be changed
  - OUT: procedure must be called with a variable for the parameter. Changes to the parameter are seen by the user (i.e., call by reference)
  - IN OUT: value can be sent, and changes to the parameter are seen by the user
- Default Mode is: IN

30

Example

```
create or replace procedure
num_logged
(person IN mylog.who%TYPE DEFAULT USER,
 num OUT mylog.logon_num%TYPE)
IS
BEGIN
    select logon_num
    into num
    from mylog
    where who = person;
null;
END;
/
```

31

Errors in a Procedure

- If there are errors in the procedure definition, they will not be shown
- To see the errors of a procedure called *proc*, type
  - SHOW ERRORS PROCEDURE *proc*

  in the SQLPLUS prompt
- For functions, type
  - SHOW ERRORS FUNCTION *fun_name*

32

Calling a Procedure

```
declare
    howmany  mylog.logon_num%TYPE;
begin
    -- parameters supplied by position
    num_logged('SAM',howmany);
    dbms_output.put_line(howmany);

    -- parameters supplied by name
    num_logged(num => howmany);
    dbms_output.put_line(howmany);
end;
/
```

33

Creating a Function

- Almost exactly like creating a procedure, but you supply a return type

```
CREATE [OR REPLACE] FUNCTION
function_name
 [(parameter1 [mode1] datatype1,
  parameter2 [mode2] datatype2,
  . . .)]
RETURN datatype
IS|AS
PL/SQL Block;
```

34

Calling a Function

- You can call a function similarly to calling a procedure, in a PL/SQL block
- A function can also be called from a query, if it only has IN parameters, and the function does not execute insert/delete/update statements

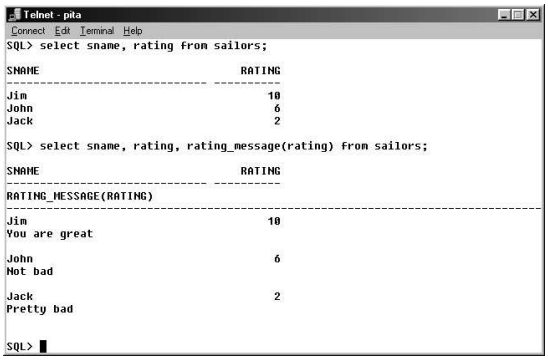
35

A Function

```
create or replace function
rating_message(rating IN NUMBER)
return VARCHAR2
AS
BEGIN
IF rating > 7 THEN
  return 'You are great';
ELSIF rating >= 5 THEN
  return  'Not bad';
ELSE
  return 'Pretty bad';
END IF;
END;
/
```

NOTE THAT YOU DON'T SPECIFY THE SIZE

36



37

Packages

- Functions, Procedures, Variables can be put together in a package
- In a package, you can allow some of the members to be "public" and some to be "private"
- There are also many predefined Oracle packages
- Won't discuss packages in this course

38