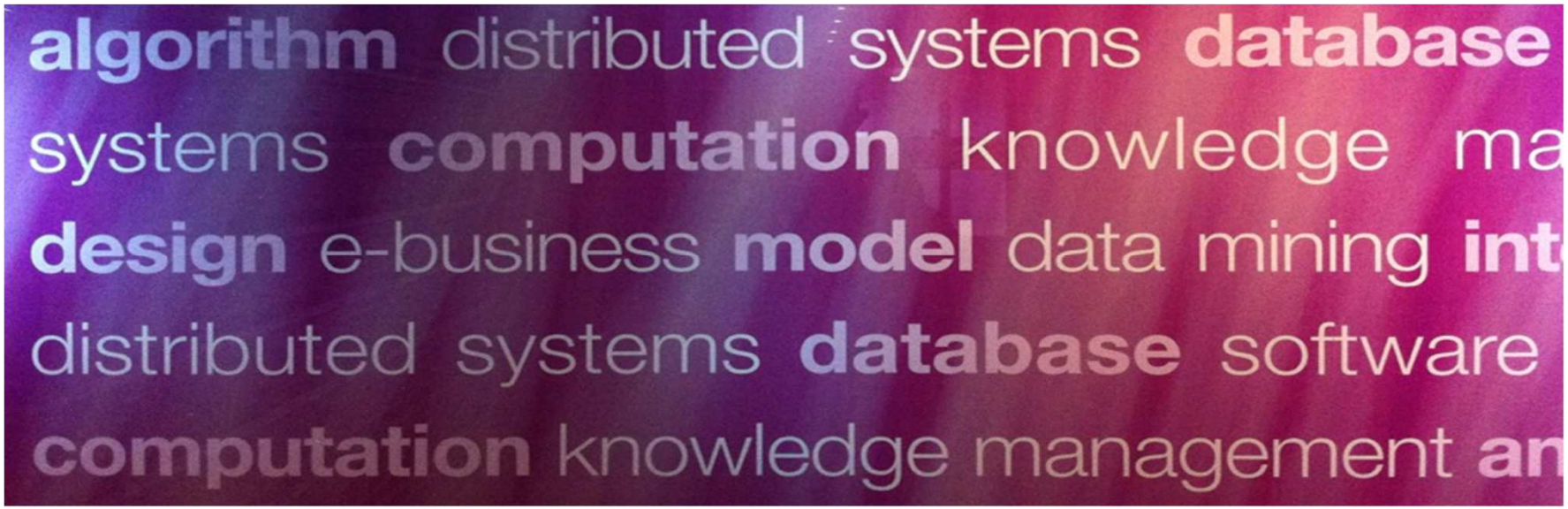
MONASH University

Information Technology

FIT3176 Advanced Database Design

Dr Minh Le
Minh.Le@monash.edu

Topic 5: PL/SQL Procedures and Functions



*Adapted from slides developed by Lindsay Smith

Learning objectives

By the end of this week you should be able to:

- Code different structures of PL/SQL Loops
- Code and use anonymous PL/SQL Block and a stored procedure
- Code and use a PL/SQL stored function
- Define and use PL/SQL cursors: implicit cursors and explicit cursors
- Define and use cursor variables: REF Cursor and SYS_REFCURSOR

References

1. Coronel, & Morris, Database Systems: Design, Implementation & Management, 11th Edition 2015, Thomson Course Technology. Chapter 8, Sections 8.7.2 - 8.7.4 inclusive
2. Oracle Database PL/SQL Language Reference, 12c Release (12.1), E50727-06, May 2017.
3. http://www.plsql-tutorial.com/index.htm.

PL/SQL Loop Structures

- LOOP-EXIT

```
DECLARE
  MaxNum CONSTANT int := 10;
  I int := 1;
BEGIN
  /* print 1 to 10 using a LOOP-EXIT */
  LOOP
    dbms_output.put_line(to_char(I));
    I := I + 1;
    EXIT WHEN I > MaxNum;
  END LOOP;
  /* end of the loop */
END;
/
```

PL/SQL Loop Structures cont'd

- WHILE-LOOP

```
/* print 1 to 10 using a WHILE-LOOP */
WHILE I <= MaxNum LOOP
  dbms_output.put_line(to_char(I));
  I := I + 1;
END LOOP;
/* end of while */
```

- FOR-LOOP

```
/* print 1 to 10 using a FOR-LOOP */
FOR I in 1..MaxNum LOOP
  dbms_output.put_line(to_char(I));
END LOOP;
/* end of for */
```

PL/SQL Procedures

- Same concept as other programming languages
  - Accepts parameters (optional), performs some operation/s and returns to caller
- General form

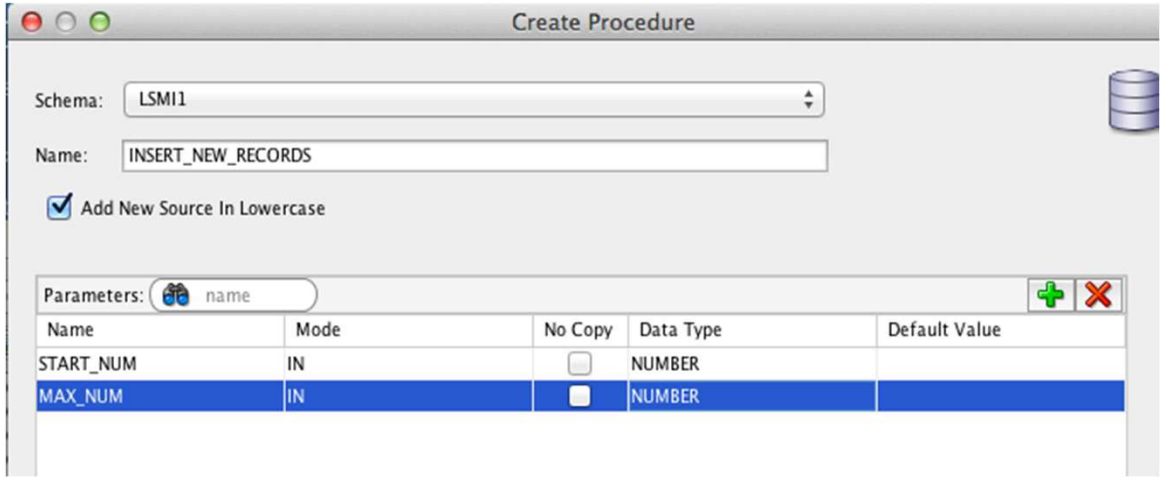
```
PROCEDURE name (parameter, .. parameter) AS
        local variables
BEGIN
        statement ... statement;
        ...
        ...
END name;
```

Parameters

- The format for parameters
  - parameter1 type1, parameter2 type2, …
- Between the name and type a mode *can* be inserted
  - IN – input parameter which cannot be changed by procedure or function
  - OUT – output parameter, will have a value returned to the caller
    - In a PROCEDURE assign value to OUT parameter, calling program then reads this value
  - IN OUT – mixture between IN and OUT
- If parameter mode not explicitly defined it is treated as IN

Procedure to insert rows into a table

- Started via SQL Developer GUI
  - Right click Procedures – Select New Procedure
  - complete definition:



  - select OK and use PL/SQL Editor to complete coding

Procedure to insert rows into a table

- Coded direct from SQL Worksheet

Worksheet | Query Builder

```
CREATE OR REPLACE PROCEDURE insert_new_records(
    start_num IN NUMBER,
    max_num   IN NUMBER )
AS

  j NUMBER := 0;

BEGIN

  FOR j IN start_num .. max_num LOOP

    INSERT INTO test
      (recordno,
       currentdate)
     VALUES
      (j,
       sysdate);

  END LOOP;

END insert_new_records;
/
```

Calling a procedure

- The procedure can be called from with an anon PL/SQL block
  - In a simple form as:

```
BEGIN
  INSERT_NEW_RECORDS( 10, 20 );
END;
/
```

  - If you wish to modify the parameters before the call:

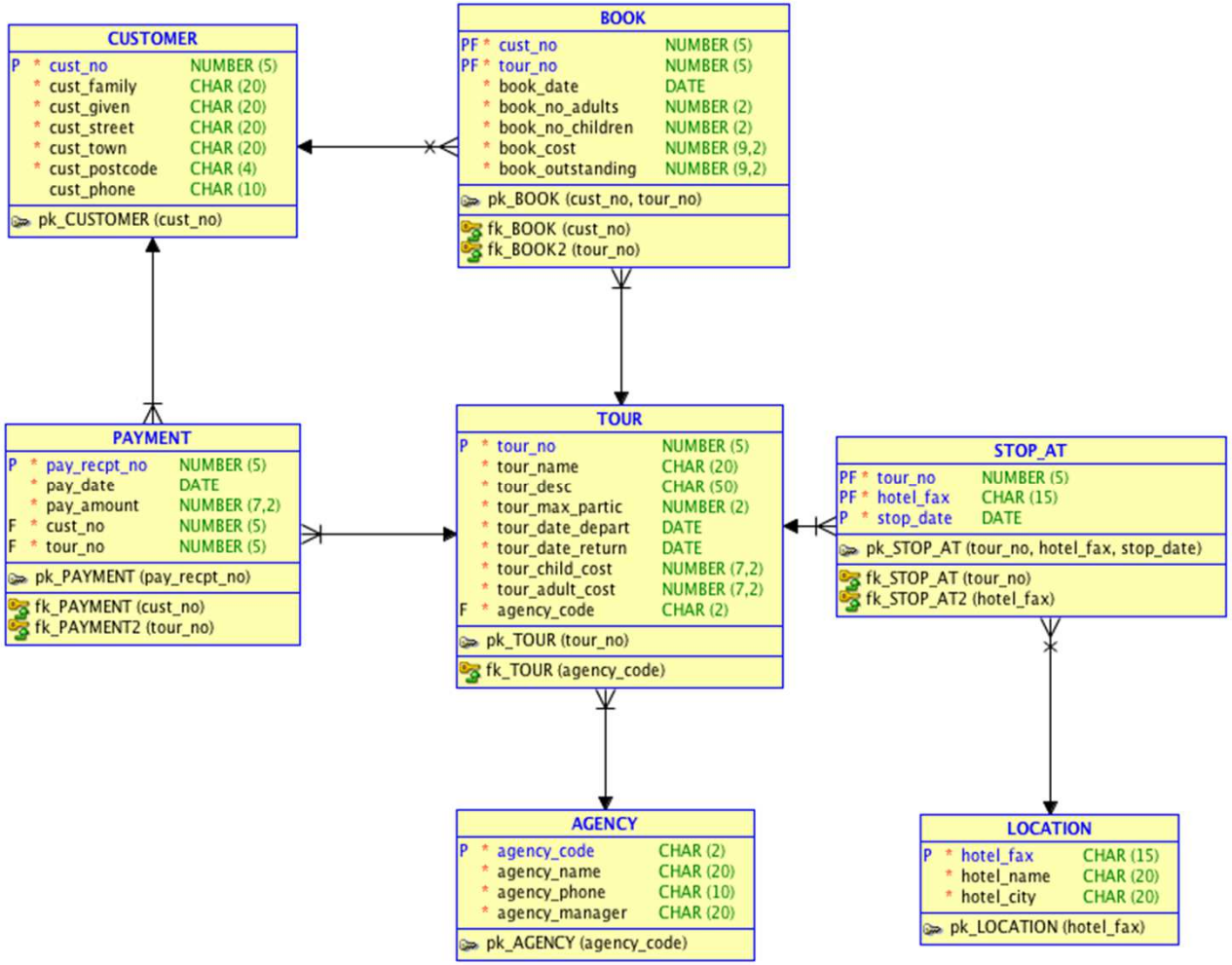
```
DECLARE
  SNUM NUMBER;
  MNUM NUMBER;
BEGIN
  SNUM := 10;
  MNUM := 20;
  INSERT_NEW_RECORDS( SNUM, MNUM );
END;
/
```

- OR via the SQL EXECUTE command:
  - EXEC INSERT_NEW_RECORDS(10, 20);

Calling a Procedure - parameters

- Parameters can be listed in one of three ways
  - Positional: compact notation but the parameters must be in the correct order
  - Named: more verbose, each parameter value and name listed, connect with =>
    - order does not matter
  - Mixed: a combination of positional and named, start with positional and then swap to named
- These are all equivalent:
  - exec cust_add (arg_no, arg_name)
  - exec cust_add (cust_no => arg_no, cust_name => arg_name)
  - exec cust_add (cust_name => arg_name, cust_no => arg_no);
  - exec cust_add (arg_no, cust_name => arg_name)

Fly-By-Night Relational Model Case Study



Q1. The declaration line for a procedure to insert a new customer could have the form:

```
procedure add_customer (
        arg_cust_no          customer.cust_no%type,
        arg_cust_family      customer.cust_family%type,
        arg_cust_given       customer.cust_given%type,
        arg_cust_street      customer.cust_street%type,
        arg_cust_town        customer.cust_town%type,
        arg_cust_postcode customer.cust_postcode%type,
        arg_cust_phone       customer.cust_phone%type);
```

The parameters as listed in this declaration are:

A. IN parameters

B. OUT parameters

C. IN OUT parameters

D. None of the above

Q2. Given the sequence cust_no_seq to provide customer numbers automatically, the declaration line for a procedure to insert a new customer has the form:

```
procedure add_customer (
        arg_cust_family     customer.cust_family%type,
        arg_cust_given      customer.cust_given%type,
        arg_cust_street     customer.cust_street%type,
        arg_cust_town       customer.cust_town%type,
        arg_cust_postcode customer.cust_postcode%type,
        arg_cust_phone      customer.cust_phone%type,
        arg_cust_no         customer.cust_no%type);
```

The arg_cust_no parameter should be an

A. IN parameter

B. OUT parameter

C. IN OUT parameter

D. It does not matter what it is declared as

Procedure to add a new CUSTOMER

```
procedure add_Customer (
      arg_cust_family      IN customer.cust_family%type,
      arg_cust_given       IN customer.cust_given%type,
      arg_cust_street      IN customer.cust_street%type,
      arg_cust_town        IN customer.cust_town%type,
      arg_cust_postcode    IN customer.cust_postcode%type,
      arg_cust_phone       IN customer.cust_phone%type,
      arg_cust_no          OUT customer.cust_no%type)
  as

  begin

    . . .

  end add_customer;
```

Q3. To add a new CUSTOMER the ADD_CUSTOMER procedure will need to carry out:

A. An SQL SELECT command

B. An SQL UPDATE command

C. An SQL INSERT command

D. Several of the above

Procedure to add a new CUSTOMER

```
procedure add_Customer (
      arg_cust_family      IN customer.cust_family%type,
      arg_cust_given       IN customer.cust_given%type,
      arg_cust_street      IN customer.cust_street%type,
      arg_cust_town        IN customer.cust_town%type,
      arg_cust_postcode    IN customer.cust_postcode%type,
      arg_cust_phone       IN customer.cust_phone%type,
      arg_cust_no          OUT customer.cust_no%type)
  as

  begin
        insert into customer values (
                cust_no_seq.NextVal,
                arg_cust_family,
                arg_cust_given,
                arg_cust_street,
                arg_cust_town,
                arg_cust_postcode,
                arg_cust_phone);
        . . .

  end add_customer;
```

Q4. To return the value of the newly assigned CUST_NO via arg_cust_no the procedure should use:

A. arg_cust_no := cust_no;

B. arg_cust_no := cust_no_seq.NextVal;

C. arg_cust_no := cust_no_seq.CurrVal;

D. select cust_no into arg_cust_no from customer;

Procedure to add a new CUSTOMER

```
procedure add_Customer (
      arg_cust_family      IN customer.cust_family%type,
      arg_cust_given       IN customer.cust_given%type,
      arg_cust_street      IN customer.cust_street%type,
      arg_cust_town        IN customer.cust_town%type,
      arg_cust_postcode    IN customer.cust_postcode%type,
      arg_cust_phone       IN customer.cust_phone%type,
      arg_cust_no          OUT customer.cust_no%type)
  as

  begin
        insert into customer values (
                cust_no_seq.NextVal,
                arg_cust_family,
                arg_cust_given,
                arg_cust_street,
                arg_cust_town,
                arg_cust_postcode,
                arg_cust_phone);

        arg_cust_no := cust_no_seq.CurrVal;

        commit;

  end add_customer;
```

Test ADD_CUSTOMER

- Suggest that during development you comment out COMMIT statements
- First test from within the procedure editor
  - Enter parameters, run, check
  - Debug is available
    - Compile for debug
    - set break points and use  to run

```
20          arg_cust_phone);
21
22      arg_cust_no := customerno.CurrVal;
23
24      commit;
25
```

Debugging: IdeConnections%23Eragon+Winx64+VM+-+lsmi1.jpr - Log Breakpo

Connecting to the database Eragon Winx64 VM - lsmi1
Executing PL/SQL: ALTER SESSION SET PLSQL_DEBUG=

Step Over (⌘-F8)

Test ADD_CUSTOMER continued

- BIND variables
  - a placeholder in a SQL statement that must be replaced with a valid value or value address
    - declare first via VAR
    - address in PL/SQL with a : preface
    - show via print
- Run from SQL command window via BIND VARIABLE/S (here new_custno) and EXEC (wrapper for begin .. end):

```
var new_custno number
exec  add_customer('Smith','Lindsay','123 Wide Rd',
   'Melbourne','3000','1234567890',:new_custno);
print new_custno
```

  - NOTE: exec must be on a single line, for multiple lines use begin .. end (don't forget the / in column 1 below end)

PL/SQL Functions

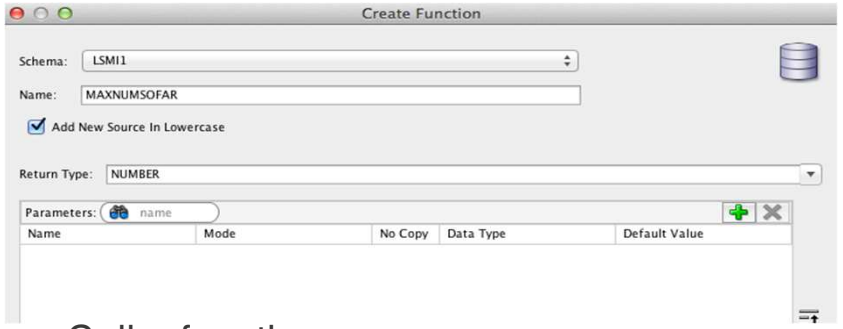
- Functions return a single value via a RETURN statement
- Functions have the general form:

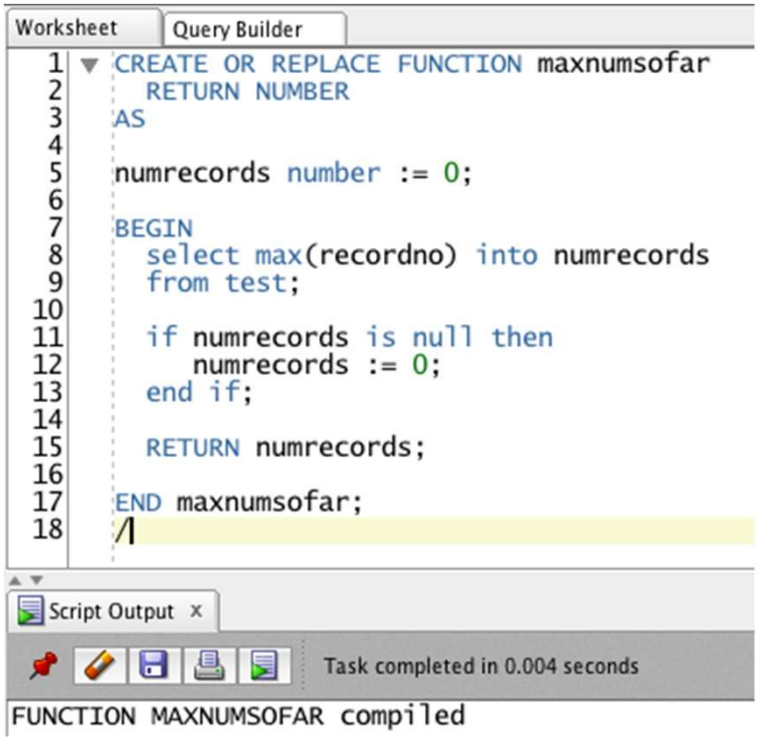
```
FUNCTION name (parameter, .. parameter)
        RETURN datatype IS
        local variables
BEGIN
        statement ... statement;
        ...
        RETURN value;
END;
```

- Call function
  - SQL: select name (parameter, .. parameter) from dual;
  - PL/SQL: name (parameter, .. parameter) eg
    - dbms_output.put_line(name (parameter, .. parameter));

FUNCTION to determine max recordno

- Declare a function
  - Create same as a procedure: GUI or SQL Worksheet





- Call a function
  1. select maxnumsofar() from dual; OR
  2. DECLARE

```
  i number := 0;
  range number := 15;
BEGIN
  i := maxnumsofar() + 1;
  INSERT_NEW_RECORDS( i, i + range );
END;
/
```

Note: the procedure insert_new_records() has been defined in SLIDE 7.

PL/SQL Cursors

- Cursor: is a pointer to a private SQL area that stores information about processing a specific SELECT or DML statement. That is, it is designed to hold data rows returned by a SELECT or DML statement.
- A cursor can be classified into two types: implicit cursors and explicit cursors
- Implicit cursor
  - A session cursor that is constructed and managed by PL/SQL.
  - PL/SQL opens an implicit cursor every time a SELECT or DML statement is run.
  - It cannot be controlled but its attributes can be accessed.
  - It closes after its associated statement runs; however, its attribute values remain available until another SELECT or DML statement runs.

Implicit Cursor Attributes

- The implicit cursor attributes include:
  - SQL%ISOPEN Attribute: Is the Cursor Open?
  - SQL%FOUND Attribute: Were Any Rows Affected?
  - SQL%NOTFOUND Attribute: Were No Rows Affected?
  - SQL%ROWCOUNT Attribute: How Many Rows Were Affected?
  - SQL%BULK_ROWCOUNT (see “Getting Number of Rows Affected by FORALL Statement” on page 12-23 in [2] for further details).
  - SQL%BULK_EXCEPTIONS (see “Handling FORALL Exceptions After FORALL Statement Completes” on page 12-20 in [2] for further details).

Example 6-3 SQL%FOUND Implicit Cursor Attribute

```
DROP TABLE dept_temp;
CREATE TABLE dept_temp AS
  SELECT * FROM departments;

CREATE OR REPLACE PROCEDURE p (
  dept_no NUMBER
) AUTHID CURRENT_USER AS
BEGIN
  DELETE FROM dept_temp
  WHERE department_id = dept_no;

  IF SQL%FOUND THEN
    DBMS_OUTPUT.PUT_LINE (
      'Delete succeeded for department number ' || dept_no
    );
  ELSE
    DBMS_OUTPUT.PUT_LINE ('No department number ' || dept_no);
  END IF;
END;
/
BEGIN
  p(270);
  p(400);
END;
/
```

Result:

Delete succeed for department number 270

No department number 400

Explicit Cursor

- Explicit cursor
  - A user-defined cursor lives within its session.
  - It must be declared, defined, given a name and associating it with a query. Usually, the query result set consists of multiple rows.
- The query result set can be processed in either of two ways:
  - Open the explicit cursor (with the OPEN statement), fetch rows from the result set (with the FETCH statement), and close the explicit cursor (with the CLOSE statement).
  - Use the explicit cursor in a cursor FOR LOOP statement (see "Processing Query Result Sets With Cursor FOR LOOP Statements" on page 6-25 in [2]).

General Form Of An Explicit Cursor [3]

```
DECLARE
   variables;
   records;
   create a cursor;
 BEGIN
  OPEN cursor;
  FETCH cursor;
    process the records;
  CLOSE cursor;
 END;
```

```
1> DECLARE
2>    emp_rec emp_tbl%rowtype;
3>    CURSOR emp_cur IS
4>    SELECT *
5>    FROM
6>    WHERE salary > 10;
7> BEGIN
8>    OPEN emp_cur;
9>    FETCH emp_cur INTO emp_rec;
10>      dbms_output.put_line (emp_rec.first_name || '  ' || emp_rec.last_name);
11>   CLOSE emp_cur;
12> END;
```

Explicit Cursor Attributes

- The explicit cursor attributes are:
  - %ISOPEN Attribute: Is the Cursor Open?
  - %FOUND Attribute: Has a Row Been Fetched?
  - %NOTFOUND Attribute: Has No Row Been Fetched?
  - %ROWCOUNT Attribute: How Many Rows Were Fetched?

Example 6-14 %ISOPEN Explicit Cursor Attribute

```
DECLARE
  CURSOR c1 IS
    SELECT last_name, salary FROM employees
    WHERE ROWNUM < 11;

  the_name employees.last_name%TYPE;
  the_salary employees.salary%TYPE;
BEGIN
  IF NOT c1%ISOPEN THEN
    OPEN c1;
  END IF;

  FETCH c1 INTO the_name, the_salary;

  IF c1%ISOPEN THEN
    CLOSE c1;
  END IF;
END;
/
```

%FOUND & %ROWCOUNT Explicit Cursor Attributes

Example 6-15 %FOUND Explicit Cursor Attribute

```
DECLARE
  CURSOR c1 IS
    SELECT last_name, salary FROM employees
    WHERE ROWNUM < 11
    ORDER BY last_name;

  my_ename   employees.last_name%TYPE;
  my_salary  employees.salary%TYPE;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO my_ename, my_salary;
    IF c1%FOUND THEN  -- fetch succeeded
      DBMS_OUTPUT.PUT_LINE('Name = ' || my_ename || ', salary = ' || my_salary);

      ELSE  -- fetch failed
        EXIT;
      END IF;
    END LOOP;
  END;
  /
```

Result:

```
Name = Austin, salary = 4800
Name = De Haan, salary = 17000
Name = Ernst, salary = 6000
Name = Faviet, salary = 9000
Name = Greenberg, salary = 12008
Name = Hunold, salary = 9000
Name = King, salary = 24000
Name = Kochhar, salary = 17000
Name = Lorentz, salary = 4200
Name = Pataballa, salary = 4800
```

Example 6-17 %ROWCOUNT Explicit Cursor Attribute

```
DECLARE
  CURSOR c1 IS
    SELECT last_name FROM employees
    WHERE ROWNUM < 11
    ORDER BY last_name;

  name  employees.last_name%TYPE;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO name;
    EXIT WHEN c1%NOTFOUND OR c1%NOTFOUND IS NULL;
    DBMS_OUTPUT.PUT_LINE(c1%ROWCOUNT || '. ' || name);
    IF c1%ROWCOUNT = 5 THEN
      DBMS_OUTPUT.PUT_LINE('--- Fetched 5th row ---');
    END IF;
  END LOOP;
  CLOSE c1;
END;
/
```

Result:

```
1. Abel
2. Ande
3. Atkinson
4. Austin
5. Baer
--- Fetched 5th row ---
6. Baida
7. Banda
8. Bates
9. Bell
10. Bernstein
```

FIGURE 8.49 A simple PRC_CURSOR_EXAMPLE

SQL Plus

```
SQL> CREATE OR REPLACE PROCEDURE PRC_CURSOR_EXAMPLE IS
  2  W_P_CODE PRODUCT.P_CODE%TYPE;
  3  W_P_DESCRIPT PRODUCT.P_DESCRIPT%TYPE;
  4  W_TOT NUMBER(3);
  5  CURSOR PROD_CURSOR IS
  6     SELECT P_CODE, P_DESCRIPT
  7       FROM PRODUCT
  8       WHERE P_QOH > (SELECT AVG(P_QOH) FROM PRODUCT);
  9  BEGIN
 10     DBMS_OUTPUT.PUT_LINE ('PRODUCTS WITH P_QOH > AVG(P_QOH)');
 11     DBMS_OUTPUT.PUT_LINE ('====================================');
 12     OPEN PROD_CURSOR;
 13     LOOP
 14        FETCH PROD_CURSOR INTO W_P_CODE, W_P_DESCRIPT;
 15        EXIT WHEN PROD_CURSOR%NOTFOUND;
 16        DBMS_OUTPUT.PUT_LINE (W_P_CODE || ' -> ' || W_P_DESCRIPT);
 17     END LOOP;
 18     DBMS_OUTPUT.PUT_LINE ('====================================');
 19     DBMS_OUTPUT.PUT_LINE ('TOTAL PRODUCTS PROCESSED ' || PROD_CURSOR%ROWCOUNT);
 20     DBMS_OUTPUT.PUT_LINE ('--- END OF REPORT ---');
 21     CLOSE PROD_CURSOR;
 22  END;
 23  /

Procedure created.

SQL> EXEC PRC_CURSOR_EXAMPLE;
PRODUCTS WITH P_QOH > AVG(P_QOH)
====================================
PVC23DRT -> PVC pipe, 3.5-in., 8-ft
SM-18277 -> 1.25-in. metal screw, 25
SW-23116 -> 2.5-in. wd. screw, 50
====================================
TOTAL PRODUCTS PROCESSED 3
--- END OF REPORT ---

PL/SQL procedure successfully completed.

SQL> _
```

SOURCE: Course Technology/Cengage Learning

User objects in the database

- Drop objects – tables, indexes, procedures, functions, packages etc
  - List all the objects a user owns:

```
select rtrim(object_name) as objname,
       rtrim(object_type) as objtype
from user_objects;
```

  - When dropping a table, the drop may be refused by Oracle due to FK constraints (type 'R' = restraints)
    - List FK constraints a user has in place:

```
select rtrim(constraint_name) as conname,
       rtrim(table_name) as tabname
from user_constraints
where rtrim(constraint_type) = 'R';
```

CLEANOUT Procedure

Code | Dependencies | Details | Errors | Profiles | Grants | References

Find

```
1  create or replace procedure cleanout as
2
3    -- Drop FK restraints (type = ''R') so tables can be dropped in any order
4    cursor con_cursor is
5       select rtrim(constraint_name) as conname,
6        rtrim(table_name) as tabname
7     from user_constraints
8     where rtrim(constraint_type) = 'R';
9
10   -- Leave some objects to prevent problems when objects are dropped out of order
11   -- Do not drop indexes, they will go with tables
12   -- Do not drop package bodies, they will go with packages
13   -- Do not drop triggers, they will go with tables
14   -- Do not drop this procedure (CLEANOUT)
15   cursor obj_cursor is
16      select rtrim(object_name) as objname,
17           rtrim(object_type) as objtype
18    from user_objects
19    where object_type <> 'INDEX'
20          and
21        object_type <> 'PACKAGE BODY'
22        and
23        object_type <> 'TRIGGER'
24        and
25        object_name <> 'CLEANOUT'
26        and
27        object_name not like ('BIN%');
28
29   con_value con_cursor%ROWTYPE;
30   obj_value obj_cursor%ROWTYPE;
31
32 BEGIN
```

Handling Cursor Output- REF CURSOR

- PL/SQL uses row approach
  - OPEN
    - FETCH until exhausted
  - CLOSE
- How to handle a multi row data from a cursor out of a procedure?
  - use the dbms_output package
  - use a new data type – REF CURSOR
- A REF CURSOR is a PL/SQL *data type* which is a **pointer** to a result set from the database. That is, its value is the address of an item, not the item itself.
- The basic syntax of a REF CURSOR type definition is:

TYPE type_name IS REF CURSOR [RETURN return_type]

Example 6-24 Cursor Variable Declarations

```
DECLARE
  TYPE empcurtyp IS REF CURSOR RETURN employees%ROWTYPE;  -- strong type
  TYPE genericcurtyp IS REF CURSOR;                       -- weak type

  cursor1  empcurtyp;       -- strong cursor variable
  cursor2  genericcurtyp;   -- weak cursor variable
  my_cursor SYS_REFCURSOR;  -- weak cursor variable

  TYPE deptcurtyp IS REF CURSOR RETURN departments%ROWTYPE;  -- strong type
  dept_cv deptcurtyp;  -- strong cursor variable
BEGIN
  NULL;
END;
/
```

- If you specify return_type, then the REF CURSOR type and cursor variables of that type are strong; if not, they are weak. SYS_REFCURSOR and cursor variables of that type are weak.
- With a strong cursor variable, you can associate only queries that return the specified type.
- With a weak cursor variable, you can associate any query.

Using REF CURSOR to Process Cursor Output

```
create or replace procedure getpayments
(
  arg_cust_no in payment.cust_no%type,
  out_cursor out sys_refcursor
) as
begin
  open out_cursor for
     select pay_date, tour_no, pay_amount from payment
     where cust_no = arg_cust_no
     order by pay_amount desc;
end getpayments;
```

SQL Worksheet (note use of bind variable *myrows*)

```
var myrows refcursor;
exec getpayments(1,:myrows);
print myrows
```

SUMMARY

- Presented how to code and use various FOR loop structures
- Discussed how to declare, define and call a stored procedure and a stored function.
- Discussed how to define and use implicit and explicit cursors, and their associated attributes
- Discussed user objects in Oracle database and explained the cleanout procedure
- Discussed how to define and use a special PL/SQL data type: REF CURSOR which can be used to handle a multi row data from a cursor out of a procedure