Bases de données

Prof. Giovanna Di Marzo Serugendo
Assistant: Mohammad Parhizkar
mohammad.parhizkar@unige.ch

Semestre: printemps 2019

session 4- 18/03/2019

PL/SQL Table-based Records

(+page 25)

The **%ROWTYPE** attribute enables a programmer to create **table-based** and **cursor-based** records.

Example 1:

```
----Records----
---Example1: The following example illustrates the concept of table-based records:
SET SERVEROUTPUT ON;
DECLARE
   director_rec directors%rowtype;
BEGIN
   SELECT * into director_rec
   FROM directors
   WHERE director_id = 5;
   dbms_output.put_line('Director ID: ' || director_rec.director_id);
   dbms_output.put_line('Director First Name: ' || director_rec.director_first_name);
   dbms_output.put_line('Director Last Name: ' || director_rec.director_last_name);
   dbms_output.put_line('Director Salary: ' || director_rec.director_salary);
END;
/
```

Script output:

```
Director ID: 5
Director First Name: James
Director Last Name: Cameron
Director Salary: 101000


PL/SQL procedure successfully completed.
```

CURSOR

Procedure

example

Example:

```
-- database: financial
SET SERVEROUTPUT ON;

-- New procedure: bestunitssold
CREATE OR REPLACE PROCEDURE bestunitssold AS

max_unit financial.unitssold%TYPE;
best_unit financial.segments%TYPE;
-- best_unit has same datatype as financial.segments

BEGIN
    SELECT MAX(unitssold) INTO max_unit FROM financial;
    SELECT  segments INTO   best_unit FROM financial WHERE unitssold=(SELECT MAX(unitssold) FROM financial);

    dbms_output.put_line(best_unit || ' - max:  ' || max_unit );

END;
/


-- execute the procedure bestunitssold
BEGIN
    bestunitssold;
END;
```

Output:

```
Procedure BESTUNITSSOLD compiled

Government - max:  4492.5


PL/SQL procedure successfully completed.
```

The purpose of this procedure:

to find out the best seller (which is just one record) and store that value in ONE variable (best_unit).

What if we want to return multiple row from a procedure and process one row at a time?

we do not need a normal variable…! we need a cursor!

CURSOR

- Oracle creates a memory area, known as the context area, for processing an SQL statement, which contains all the information needed to process the statement. **A cursor is a pointer to this context area.**
- **Definition:** A cursor is a pointer to a private SQL area that stores information about the processing of a SELECT or data manipulation language (DML) statement (INSERT, UPDATE, DELETE, or MERGE) for access at a later time.
- PL/SQL controls the context area through a cursor.

FR

Pour chaque instruction SQL, Oracle crée des zones de travail pour exécuter les ordres SQL, stocker leurs résultats et les utiliser : les curseurs.

Il y a deux types de curseur:

1. implicite : créés automatiquement par Oracle pour ses propres traitements
2. explicite : créés par l'utilisateur pour pouvoir traiter le résultat de requêtes retournant plus d'un enregistrement/tuple.

CURSOR

We have two types of cursors:

- Explicit cursors: programmer-defined cursors
  (OPEN-FETCH-CLOSE)

- Implicit cursors: automatically created by Oracle

Q. **What happens when the query fetches multiple records?**

A. This is where cursor sets in. By using a cursor, you can use the same variables (or a record type) to hold the data, and do whatever manipulation you want to do, and move to the next record.

Explicit CURSOR

- **Definition (simple way)**: An explicit cursor is a **SELECT** statement that is defined within the *declaration* section of your PL/SQL code.

Explicit CURSOR steps:

1. Declaring the cursor for initializing the memory
2. Opening the cursor for allocating the memory
3. Fetching the cursor for retrieving the data
4. Closing the cursor to release the allocated memory

Explicit CURSOR: Attributes

Oracle provides four attributes to check the status of explicit cursor:

(number)

- **%ROWCOUNT:** It returns the number of rows fetched by the fetch statement.

(True / False)

- **%NOTFOUND:** It returns TRUE if fetch statement does not return a row. It returns FALSE, when fetch statement returns at least one row.

(True / False)

- **%FOUND:** It returns TRUE if fetch statement returns at least one row. It returns FALSE, when fetch statement does not return a row.

(True / False)

- **%ISOPEN:** TRUE, if the cursor is already open in the program. It returns FALSE when the cursor is not open in the program.

Explicit CURSOR

Different syntaxes to declare an explicit cursor:

1. Cursor without parameters
2. Cursor with parameters

Explicit CURSOR

Without parameters

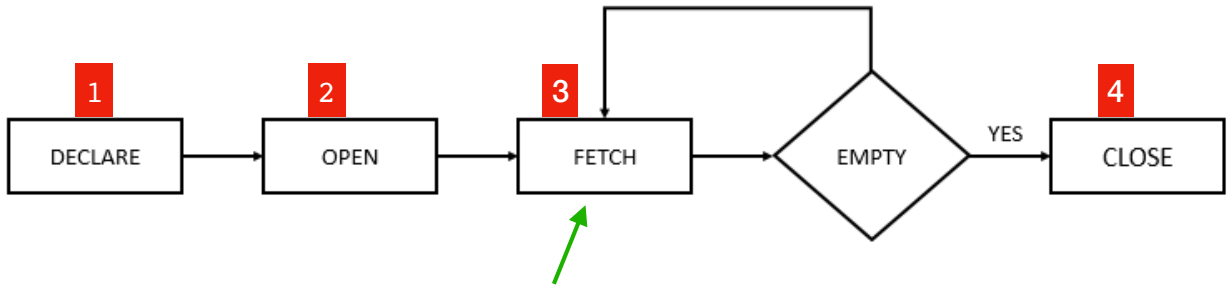
Example 1:

```
DECLARE
   c_id directors.director_id%type;
   c_name directors.director_last_name%type;
   c_salary directors.director_salary%type;
1  CURSOR c_directors is
      SELECT director_id, director_last_name, director_salary FROM directors;
BEGIN
2  OPEN c_directors;
   LOOP
3  FETCH c_directors into c_id, c_name, c_salary;
      EXIT WHEN c_directors%notfound;
      dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_salary);
   END LOOP;
4  CLOSE c_directors;
END;
/
```

Script output:

```
1 Reitman 22500
2 Mankiewicz 42500
3 Lasseter 62500
4 Doctor 82500
5 Cameron 102500
6 Haggis 122500
7 Bird 142500
8 Hobson 162500
9 Soderbergh 182500
10 Reitman 202500


PL/SQL procedure successfully completed.
```



After the cursor has been declared and opened, you can retrieve data from the cursor. The process of getting data from the cursor is called fetching the cursor. There are two ways to fetch a cursor:

```
FETCH cursor_name INTO PL/SQL variables;
```

or

```
FETCH cursor_name INTO PL/SQL record;
```

http://www.oracletutorial.com/plsql-tutorial/plsql-cursor/

Explicit CURSOR

Without parameters

Example 2:

```
1  DECLARE
     CURSOR crs IS SELECT * FROM directors;
     REC directors%ROWTYPE;
   BEGIN
     IF NOT crs%ISOPEN THEN
2      OPEN crs;
     END IF;
     LOOP
3      FETCH crs INTO REC;
       EXIT WHEN crs%NOTFOUND;
       DBMS_OUTPUT.PUT_LINE(REC.director_first_name || ' ' ||
                            REC.director_last_name);
     END LOOP;
     IF crs%ISOPEN THEN
4      CLOSE crs;
     END IF;
   END;
   /
```

- The FETCH statement places the contents of current row into variables.
- To retrieve all rows in a result set, you need to fetch each row till the last one.

Script output:

```
Ivan Reitman
Joseph Mankiewicz
John Lasseter
Pete Doctor
James Cameron
Paul Haggis
Brad Bird
Henry Hobson
Steven Soderbergh
Jason Reitman


PL/SQL procedure successfully completed.
```

Code explanation:

An *explicit* cursor is declared in this example. A corresponding variable that has a %ROWTYPE anchored declaration will hold each record returned. The cursor is opened and a loop is used to fetch each row into the record variable.

The loop is finished when the cursor attribute %NOTFOUND is equal to true. The cursor is closed when processing is complete. This saves database resources and prevents exceptions that can result from having too many cursors open. Note that an exception is thrown if there is an attempt to close a cursor that is not open, or if there is an attempt to open a cursor that is already open. The %ISOPEN boolean cursor attribute is useful for avoiding these exceptions.

Cursor FOR LOOP

You would use a CURSOR FOR LOOP when you want to fetch and process every record in a cursor. The CURSOR FOR LOOP will terminate when all of the records in the cursor have been fetched.

Syntax:

```
FOR record_index in cursor_name
LOOP
   {...statements...}
END LOOP;
```

- With a cursor FOR loop, the process of opening, fetching, and closing is handled implicitly.

Example:

```
SET SERVEROUTPUT ON;

DECLARE
   CURSOR cursor_film IS
      SELECT title
      FROM films
      WHERE director_id > 5
      ORDER BY title;
BEGIN
   FOR v_film IN cursor_film LOOP
      DBMS_OUTPUT.PUT_LINE ('Title = ' || v_film.title);
   END LOOP;
END;
/
```

Script output:

```
Title = Crash
Title = Maggie
Title = Ocean's twelve
Title = Ratatouille
Title = Solaris
Title = Traffic
Title = Up in the air


PL/SQL procedure successfully completed.
```

Cursor FOR LOOP

In PL/SQL whenever we are fetching data, we need to store the fetched data in a variable.

Example:

```
 SET SERVEROUTPUT ON
-- cursor_for_loop
DECLARE
    CURSOR c_director IS SELECT * FROM directors
    WHERE director_id <=7;
    tmp directors%rowtype;
BEGIN
--OPEN c_director;

    FOR tmp IN c_director
    LOOP
    --FETCH c_director into tmp;
    dbms_output.put_line('ID:    '||tmp.director_id);
    dbms_output.put_line('First Name:  '||tmp.director_first_name);
    dbms_output.put_line('Last Name:  '||tmp.director_last_name);
    dbms_output.put_line('Salary:'||tmp.director_salary);

    --EXIT WHEN c_director%NOTFOUND;

    END LOOP;

--CLOSE c_director;
END;
/
```

Script output:

```
ID:    1
First Name:  Ivan
Last Name:  Reitman
Salary:21000
ID:    2
First Name:  Joseph
Last Name:  Mankiewicz
Salary:41000
ID:    3
First Name:  John
Last Name:  Lasseter
Salary:61000
ID:    4
First Name:  Pete
Last Name:  Doctor
Salary:81000
ID:    5
First Name:  James
Last Name:  Cameron
Salary:101000
ID:    6
First Name:  Paul
Last Name:  Haggis
Salary:121000
ID:    7
First Name:  Brad
Last Name:  Bird
Salary:141000


PL/SQL procedure successfully completed.
```

Explicit CURSOR

With parameters

We can pass parameters into a cursor and use them in the query.

Syntax:

```
CURSOR cursor_name (parameter_list)
IS
  SELECT_statement;
```

Example:

```
SET Serveroutput ON;
DECLARE
    rec_director directors%ROWTYPE;
    CURSOR cur_director (low_salary NUMBER, high_salary NUMBER)
    IS
        SELECT *
        FROM directors
        WHERE director_salary BETWEEN low_salary AND high_salary;
BEGIN
    -- Rich directors
    DBMS_OUTPUT.PUT_LINE('Rich director: ');
    OPEN cur_director(80000,180000);
    LOOP
        FETCH cur_director INTO rec_director;
        EXIT WHEN cur_director%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(rec_director.director_last_name || ': ' ||rec_director.director_salary);
    END LOOP;
    CLOSE cur_director;

    -- Very rich directors
    DBMS_OUTPUT.PUT_LINE( '---- ' );

    DBMS_OUTPUT.PUT_LINE('Very rich directors: ');
    OPEN cur_director(180000,300000);
    LOOP
        FETCH cur_director INTO rec_director;
        EXIT WHEN cur_director%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(rec_director.director_last_name || ': ' ||rec_director.director_salary);
    END LOOP;
    CLOSE cur_director;

END;
```

Script output:

```
Rich director:
Doctor: 81000
Cameron: 101000
Haggis: 121000
Bird: 141000
Hobson: 161000
----
Very rich directors:
Soderbergh: 181000
Reitman: 201000


PL/SQL procedure successfully completed.
```

PL/SQL cursor with parameters to fetch data based on parameters.

Nested Explicit CURSOR

TP4-Q.4

Example 1:

```
--------------------Nested Cursors -------------------------
SET SERVEROUTPUT ON
/*This example has two cursors. The first is a cursor of the departmen id,
and the second is a list of employees. */
DECLARE
        v_dep departments.department_id%TYPE;
        v_emp_flag CHAR;
--The variable v_dep is initialized, to be the dept id of the current record of the c_dep cursor.
--The c_ employee cursor ties in the c_dep cursor by means of this variable.
        CURSOR c_dep IS
            SELECT department_id, department_name
                FROM departments
            WHERE manager_id= 100;
        CURSOR c_emp IS
            SELECT first_name, last_name
                FROM employees
            WHERE department_Id = v_dep;
BEGIN
/* It is retrieving employess who have the dept id of the current record for
the parent cursor. */
        FOR r_dep IN c_dep
        LOOP
            v_emp_flag := 'N';
            v_dep := r_dep.department_id;
            DBMS_OUTPUT.PUT_LINE(CHR(10));
            DBMS_OUTPUT.PUT_LINE('Employee working at '||
/*Each iteration of the parent
cursor will execute the DBMS_OUTPUT in lines 16 and 17 only once. */
            r_dep.department_name);
            FOR r_emp in c_emp
            LOOP
/*The DBMS_OUTPUT will be
executed once for each iteration of the child loop,
producing a line of output for each employee*/
                    DBMS_OUTPUT.PUT_LINE(
                    r_emp.first_name||
                    ' '||r_emp.last_name);
                    v_emp_flag := 'Y';
            END LOOP;
            IF v_emp_flag = 'N'
                    THEN
                    DBMS_OUTPUT.PUT_LINE
                    ('No Employee working this department');
/*The DBMS_OUTPUT.PUT_LINE statement  will only execute if
the inner loop did not execute.*/
            END IF;
        END LOOP;
END;
/*v_emp_flag: The variable is set to N in the beginning of
the parent loop. If the child loop executes at least once,
the variable is set to Y.*/
/*. After the child loop has closed, a check is made with an
IF statement to determine the value of the variable. If it is still N, it can be
safely concluded that the inner loop did not process.
This will then allow the last DBMS_OUTPUT.PUT_LINE statement to execute.*/
```

- Cursors can be nested inside each other.
- Although this may sound complex, it is really just a **loop inside a loop**, much like nested loops,

Script output 1:

```
Employee working at Executive
Steven King
Neena Kochhar
Lex De Haan


PL/SQL procedure successfully completed.
```

Next page –>

<— Previous page

Nested Explicit CURSOR

Example 1 – result explanation:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMEN...
100	Steven	King	SKING	515.123.4567	21.09.89	AD_PRES	24000	(null)	(null)	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21.09.89	AD_VP	17000	(null)	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	13.01.93	AD_VP	17000	(null)	100	90

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGE...	DEPARTMENT_ID
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21.09.89	AD_VP	17000	(null)	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	13.01.93	AD_VP	17000	(null)	100	90
114	Den	Raphaely	DRAPHEAL	515.127.4561	07.12.94	PU_MAN	11000	(null)	100	30
120	Matthew	Weiss	MWEISS	650.123.1234	18.07.96	ST_MAN	8000	(null)	100	50
121	Adam	Fripp	AFRIPP	650.123.2234	10.04.97	ST_MAN	8200	(null)	100	50
122	Payam	Kaufling	PKAUFLIN	650.123.3234	01.05.95	ST_MAN	7900	(null)	100	50
123	Shanta	Vollman	SVOLLMAN	650.123.4234	10.10.97	ST_MAN	6500	(null)	100	50
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16.11.99	ST_MAN	5800	(null)	100	50
145	John	Russell	JRUSSEL	011.44.1344.429268	01.10.96	SA_MAN	14000	0.4	100	80
146	Karen	Partners	KPARTNER	011.44.1344.467268	05.01.97	SA_MAN	13500	0.3	100	80
147	Alberto	Errazuriz	AERRAZUR	011.44.1344.429278	10.03.97	SA_MAN	12000	0.3	100	80
148	Gerald	Cambrault	GCAMBRAU	011.44.1344.619268	15.10.99	SA_MAN	11000	0.3	100	80
149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29.01.00	SA_MAN	10500	0.2	100	80
201	Michael	Hartstein	MHARTSTE	515.123.5555	17.02.96	MK_MAN	13000	(null)	100	20

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
120	Treasury	(null)	1700
130	Corporate Tax	(null)	1700
140	Control And Credit	(null)	1700
150	Shareholder Services	(null)	1700

Example 1
Script output:

```
Employee working at Executive
Steven King
Neena Kochhar
Lex De Haan


PL/SQL procedure successfully completed.
```

Implicit CURSOR

- Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement.
- Programmers can not control the implicit cursors and the information in it.
- Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement.
  - For INSERT operations, the cursor holds the data that needs to be inserted.
  - For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

A **SELECT-INTO** is also referred to as an **IMPLICIT** query, because Oracle Database opens a cursor for the SELECT statement, fetches the row, and then closes the cursor when it finishes doing that (or when an exception is raised).

You can, alternatively, explicitly declare a cursor and then perform the open, fetch, and close operations yourself.

Implicit CURSOR

You have a choice between using an implicit or explicit cursor only when you execute a single-row SELECT statement (a SELECT that returns only one row).

Implicit CURSOR: SELECT-INTO statement

SELECT-INTO offers the fastest and simplest way to fetch a **single row** from a SELECT statement.

Syntax:

```
SELECT select_list
INTO
variable_list FROM remainder_of_query;
```

- If the SELECT statement identifies more than one row to be fetched, Oracle Database will raise the TOO_MANY_ROWS exception. If the statement doesn't identify any rows to be fetched, Oracle Database will raise the NO_DATA_FOUND exception.

Example 1:

```
--Get the last name for a specific employee id (the primary key in the employees table):
SET SERVEROUTPUT ON;
DECLARE
  l_last_name  employees.last_name%TYPE;
BEGIN
  SELECT last_name
    INTO l_last_name
    FROM employees
   WHERE employee_id = 138;

  DBMS_OUTPUT.put_line (
     l_last_name);
END;
```

Script output:

```
Stiles


PL/SQL procedure successfully completed.
```

Example 2:

```
--Fetch an entire row from the employees table for a specific employee ID:

DECLARE
   l_employee   employees%ROWTYPE;
BEGIN
   SELECT *
     INTO l_employee
     FROM employees
    WHERE employee_id = 138;

   DBMS_OUTPUT.put_line (
      l_employee.last_name);
END;
```

Script output:

```
Stiles


PL/SQL procedure successfully completed.
```

Implicit CURSOR: SELECT-INTO statement

SELECT-INTO offers the fastest and simplest way to fetch a **single row** from a SELECT statement.

Example 3:

```
--Fetch columns from different tables:

DECLARE
  l_last_name
     employees.last_name%TYPE;
  l_department_name
     departments.department_name%TYPE;
BEGIN
  SELECT last_name, department_name
    INTO l_last_name, l_department_name
    FROM employees e, departments d
   WHERE e.department_id=d.department_id
         AND e.employee_id=138;

  DBMS_OUTPUT.put_line (
     l_last_name ||
     ' in ' ||
     l_department_name);
END;
```

Script output:

```
Stiles in Shipping


PL/SQL procedure successfully completed.
```

Implicit CURSOR: Attributes

Oracle provides four attributes called as implicit cursor attributes to check the status of DML operations.

(number)

- **%ROWCOUNT:** It returns the number of rows influenced by an UPDATE, DELETE or an INSERT statement.

(True / False)

- **%NOTFOUND:** It returns TRUE if an UPDATE, DELETE or an INSERT statement influenced zero rows or a SELECT INTO statement returned no rows.

(True / False)

- **%FOUND:** It returns TRUE if an UPDATE, DELETE or an INSERT statement influenced one or more rows or a SELECT INTO statement returned one or more rows.

(True / False)

- **%ISOPEN:** It always returns FALSE for implicit cursors.

Implicit CURSOR: Attributes

Example 1:

This program will update **directors** table *(from TP1)* and increase the salary of each director by 500.

```
---------- Directors table (TP1) --------
SET Serveroutput ON;
DECLARE
   total_rows number(2);
BEGIN
   UPDATE directors
   SET director_salary = director_salary + 500;
   IF sql%notfound THEN
      dbms_output.put_line('no directors selected');
   ELSIF sql%found THEN
      total_rows := sql%rowcount;
      dbms_output.put_line( total_rows || ' directors selected ');
   END IF;
END;
/
```

When you execute this block, our implicit cursor attributes help you to find out whether any row has been returned by the UPDATE statement.

The block use the **SQL%ROWCOUNT** attribute to determine the number of rows affected by UPDATE statement.

Script output:

```
10 directors selected


PL/SQL procedure successfully completed.
```

Result:

DIRECTOR_ID	DIRECTOR_FIRST_NAME	DIRECTOR_LAST_NAME	DIRECTOR_BD	COUNTRY	DIRECTOR_SALARY
1	Ivan	Reitman	01.03.46	Slovakia	22500
2	Joseph	Mankiewicz	01.03.09	USA	42500
3	John	Lasseter	01.03.57	USA	62500
4	Pete	Doctor	(null)	USA	82500
5	James	Cameron	01.03.54	Canada	102500
6	Paul	Haggis	01.03.53	Canada	122500
7	Brad	Bird	(null)	USA	142500
8	Henry	Hobson	01.03.85	USA	162500
9	Steven	Soderbergh	01.03.63	USA	182500
10	Jason	Reitman	(null)	Canada	202500

Implicit CURSOR: Attributes

Example 2:

```
set serveroutput on
--edit implicit_cursor
BEGIN
    UPDATE employees SET department_id= 50 --'Shipping'
        WHERE first_name='Ernst';

    IF SQL%FOUND THEN
        dbms_output.put_line('Updated - If Found');
    END IF;

    IF SQL%NOTFOUND THEN
        dbms_output.put_line('NOT Updated - If NOT Found');
    END IF;

    IF SQL%ROWCOUNT>0 THEN
        dbms_output.put_line(SQL%ROWCOUNT||' Rows Updated');
    ELSE
        dbms_output.put_line('NO Rows Updated Found');
    END IF;
END;
/
```

Script output 2:

```
NOT Updated - If NOT Found
NO Rows Updated Found


PL/SQL procedure successfully completed.
```

Example 3:

```
set serveroutput on
--edit implicit_cursor
BEGIN
    UPDATE employees SET department_id= 50 --'Shipping'
        WHERE last_name='Ernst';

    IF SQL%FOUND THEN
        dbms_output.put_line('Updated - If Found');
    END IF;

    IF SQL%NOTFOUND THEN
        dbms_output.put_line('NOT Updated - If NOT Found');
    END IF;

    IF SQL%ROWCOUNT>0 THEN
        dbms_output.put_line(SQL%ROWCOUNT||' Rows Updated');
    ELSE
        dbms_output.put_line('NO Rows Updated Found');
    END IF;
END;
/
```

Script output 3:

```
Updated - If Found
1 Rows Updated


PL/SQL procedure successfully completed.
```

Result 2: nothing has changed here …

102	Lex	De Haan	LDEHAAN	515.123.4569	13.01.93	AD_VP	17000	(null)	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	03.01.90	IT_PROG	9000	(null)	102	60
104	Bruce	Ernst	BERNST	590.423.4568	21.05.91	IT_PROG	6000	(null)	103	60
105	David	Austin	DAUSTIN	590.423.4569	25.06.97	IT_PROG	4800	(null)	103	60
106	Valli	Pataballa	VPATABAL	590.423.4560	05.02.98	IT_PROG	4800	(null)	103	60

Result 3:

102	Lex	De Haan	LDEHAAN	515.123.4569	13.01.93	AD_VP	17000	(null)	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	03.01.90	IT_PROG	9000	(null)	102	60
104	Bruce	Ernst	BERNST	590.423.4568	21.05.91	IT_PROG	6000	(null)	103	50
105	David	Austin	DAUSTIN	590.423.4569	25.06.97	IT_PROG	4800	(null)	103	60
106	Valli	Pataballa	VPATABAL	590.423.4560	05.02.98	IT_PROG	4800	(null)	103	60

Implicit CURSOR: Attributes

Change the first_name of the director, who has a last_name which starts with B

Example 4:

```
-----Implicit cursors-----ROWCOUNT
SET SERVEROUTPUT ON
BEGIN
    UPDATE directors
        SET director_first_name = 'Bradd'
    WHERE director_last_name LIKE 'B%';
    DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT);
END;
```

LIKE condition provides us pattern matching in WHERE clause

Script output:

```
1        <———————  The number of changes

PL/SQL procedure successfully completed.
```

For more information about **LIKE** condition: *https://docs.oracle.com/cd/B13789_01/server.101/b10759/conditions016.htm*

PL/SQL Table Variables

—-> Next page

PL/SQL Table Variables

Example:

```
SET Serveroutput ON;
--------Two new tables creation ---------
CREATE TABLE tab1 (x1 NUMBER, y1 VARCHAR2(20), z1 VARCHAR2(40)) TABLESPACE users;
CREATE TABLE tab2 (x2 NUMBER, y2 NUMBER, z2 NUMBER) TABLESPACE users;
-----INSERT values in new table tab1 ---------
DECLARE
  i   NUMBER := 0;
BEGIN
  FOR i IN 0 .. 13
  LOOP
     INSERT INTO tab1(x1, y1, z1)
          VALUES (i, dbms_random.string('l',20), dbms_random.string('l',40));
  END LOOP;
END;
/
-------INSERT values in new table tab2 ---------
DECLARE
  i   NUMBER := 0;
BEGIN
  FOR i IN 0 .. 6
  LOOP
     INSERT INTO tab2(x2, y2, z2)
          VALUES (i, TRUNC(dbms_random.value*1000), TRUNC(dbms_random.value*100));
                END LOOP;
END;
/
-------Queries---------
SELECT * FROM tab1;
SELECT * FROM tab2;
```

optional, specify the table space

block 1

to generate random text strings

putting some random strings in tab1

block 2

putting some random numbers in tab2

The TRUNC function returns a numeric value, e.g. TRUC(125.232) = 125

to generate random numbers between 0-1

TAB1:

	X1	Y1	Z1
0	tyjxqoohszouijebluyg	oeyuuxyzxysqevjskesatexvjtepsfdvvienviwo
1	mhxwuymsnxbfzwbyyckn	ccusgweadggwlqxocknwrgvscfttkobykrgsdbwy
2	espyozucgpvpigffhomr	lfynynppsumhddvffpujuojtvgdrgtkphumdgeng
3	tgkksqvovaocdvlglbfe	gzezpzkjhjmzwftnfeirgrtjovvtvrhmdsgjgutz
4	zhnsfhpiztxssznvuasl	qswkbsicannflwjmyjrootloqvhxszuwnpkbzwvf
5	ucnkpheqifjjtezhndiu	rcznxcxgksersmeyeapeyaqgmdtdatsvrzusxatvi
6	lvjukhnuudfouoguwdow	qbcszkpmyummlntkyuhxxjpohbmwxhtqvxalgyhi
7	cahfrzytinsumdkxdoow	rjdtrnxsmbtauodpjnhqpeqkkdnyloixilctnfye
8	asudcvjvrrntsfhcnrsd	bgpumwuhtbtjgbtszafbvbsdtpzvcvxplnzmppkw
9	xdfxtglsfmeiskhvvpqo	srwfpswvqeowdlzpiewzcdevgekgdpbhhhbmvbtk
10	kapiydybpdvxtujywhls	hxdylqjzntzteliahcwvhfdwklfdonedzwiwryhl
11	xxutodxbxgtenppsijba	docmqipwjqlfarpwjbxhrzhccuvifkwvmkxogjqy
12	tjgyxjawqmzaijcgiyuw	slrwajtgafvckefpjizthviotybuislyvivutafy
13	pwzbrfpsrbzptdllktrj	befwnqwdeknyeyuybyglzbkcvwkfbmduekvdmwev

TAB2:

X2	Y2	Z2
0	643	13
1	639	85
2	999	75
3	510	82
4	52	70
5	873	38
6	788	6

file: TAB1_TAB2_TabVariables.sql

<—- From previous page

PL/SQL Table Variables

```
DECLARE
  TYPE tab1_tab_type IS TABLE OF tab1%ROWTYPE
     INDEX BY BINARY_INTEGER;

  i              INTEGER     := 0;
  tab1_tab         tab1_tab_type;
  tab1_tab_empty   tab1_tab_type;

  CURSOR c_1
  IS
     SELECT x1, y1, z1
       FROM tab1;

  CURSOR c_2
  IS
     SELECT x2, TO_CHAR (y2), TO_CHAR (z2)
       FROM tab2;
BEGIN
  FOR c_c IN c_1
  LOOP
     tab1_tab (i) := (c_c);
     i := i + 1;
  END LOOP;

  FOR c_c IN c_2
  LOOP
     tab1_tab (i) := (c_c);
     i := i + 1;
  END LOOP;

  FOR my_row IN 0 .. i - 1
  LOOP
     DBMS_OUTPUT.put_line (   RPAD (TO_CHAR (tab1_tab (my_row).x1), 9)
                           || RPAD (tab1_tab (my_row).y1, 30)
                           || RPAD (tab1_tab (my_row).z1, 30)
                          );
  END LOOP;

  tab1_tab := tab1_tab_empty;  --erase tab1_tab of all values
END;
/
```

a PL/SQL data type used for storing signed integers, identical to PLS_INTEGER

Script output:

```
0       tyjxqoohszouijebluyg          oeyuuxyzxysqevjskesatexvjtepsf
1       mhxwuymsnxbfzwbyyckn          ccusgweadggwlqxocknwrgvscfttko
2       espyozucgpvpigffhomr          lfynynppsumhddvffpujuojtvgdrgt
3       tgkksqvovaocdvlglbfe          gzezpzkjhjmzwftnfeirgrtjovvtvr
4       zhnsfhpiztxssznvuasl          qswkbsicannflwjmyjrootloqvhxsz
5       ucnkpheqifjjtezhndiu          rcznсxgksersmeyeapeyaqgmdtdats
6       lvjukhnuudfouoguwdow          qbcszkpmyummlntkyuhxxjpohbmwxh
7       cahfrzytinsumdkxdoow          rjdtrnxsmbtauodpjnhqpeqkkdnylo
8       asudcvjvrrntsfhcnrsd          bgpumwuhtbtjgbtszafbvbsdtpzvcv
9       xdfxtglsfmeiskhvvpqo          srwfpswvqeowdlzpiewzcdevgekgdp
10      kapiydybpdvxtujywhls          hxdylqjzntzteliahcwvhfdwklfdon
11      xxutodxbxgtenppsijba          docmqipwjqlfarpwjbxhrzhccuvifk
12      tjgyxjawqmzaijcgiyuw          slrwajtgafvckefpjizthviotybuis
13      pwzbrfpsrbzptdllktrj          befwnqwdeknyeyuybyglzbkcvwkfbm
0       643                           13
1       639                           85
2       999                           75
3       510                           82
4       52                            70
5       873                           38
6       788                           6

PL/SQL procedure successfully completed.
```

PL/SQL Records

PL/SQL Records

- A **record** is another kind of datatypes that can hold data items of different kinds.
- Records consist of different fields, similar to a row of a database table.
- For example, you want to keep track of your books in a library. You might want to track the following attributes about each book, such as Title, Author, Subject, Book ID.

PL/SQL can handle the following types of records:

(page 2)

- Table-based
- Cursor-based records
- User-defined records

PL/SQL Cursor-based Records

The **%ROWTYPE** attribute enables a programmer to create **table-based** and **cursor-based** records.

Example:

```
-----Example 2: Cursor-based records
DECLARE
   CURSOR director_cur is
      SELECT director_id, director_first_name, director_last_name
      FROM directors;
   director_rec director_cur%rowtype;
BEGIN
   OPEN director_cur;
   LOOP
      FETCH director_cur into director_rec;
      EXIT WHEN director_cur%notfound;
      DBMS_OUTPUT.put_line(director_rec.director_id || ' ' || director_rec.director_last_name);
   END LOOP;
END;
/
```

Script output:

```
1 Reitman
2 Mankiewicz
3 Lasseter
4 Doctor
5 Cameron
6 Haggis
7 Bird
8 Hobson
9 Soderbergh
10 Reitman


PL/SQL procedure successfully completed.
```

PL/SQL User-defined Records

Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book:

- Title
- Author
- Subject
- Book ID

Syntax:

```
TYPE
type_name IS RECORD
  ( field_name1  datatype1  [NOT NULL]  [:= DEFAULT EXPRESSION],
   field_name2   datatype2   [NOT NULL]  [:= DEFAULT EXPRESSION],
   ...
   field_nameN  datatypeN  [NOT NULL]  [:= DEFAULT EXPRESSION);
record-name  type_name;
```

Example:

```
DECLARE
TYPE books IS RECORD
(title  varchar(50),
   author  varchar(50),
   subject varchar(100),
   book_id   number);
book1 books;
book2 books;
```

Thank you!