PL/SQL חלק א'

מבנה התוכנית

- PL/SQL מרחיבה את SQL ומחזק את כוחו על ידי הענקת כלים משפות פרוצדוראליות.

- בלוק (תוכנית) ב-PL/SQL הוא בעל המבנה הבא:

```
DECLARE
                    /*הכרזת משתנים, טיפוסים ופרוצדורות. */

BEGIN
                    /* חלק ביצועי: משפטים פרוצדוראליים ומשפטי SQL */
                    /* זהו החלק היחיד שנדרש בבלוק. (זהו החלק העיקרי שרץ). */

EXCEPTION
                    /* חלק הטיפול בחריגים. משפטי טיפול בטעויות יבואו כאן. */
END;
```

- משפטי ה-SQL שיכולים להשתלב בתוכנית PL/SQL הם אלו המתחילים ב-SELECT, UPDATE, DELETE.
- יש לדעת כי **תחביר ה-SELECT שונה מהרגיל** (נראה בהמשך).
- משפטי הגדרה כמו CREATE, DROP, ALTER **אינם** מותרים ב-PL/SQL.
- הכלים הפרוצדוראליים ש-PL/SQL תומך בהם הם:
  - השמות.
  - התניות.
  - לולאות.
  - קריאות לפרוצדורות.
  - הדקים (Triggers).

- PL/SQL בדומה ל-SQL אינו רגיש לאותיות קטנות/גדולות.
- הערות (בדומה לשפת C, יבואו בצורה /* …….. */

- כדי להריץ תוכנית PL/SQL עלינו להוסיף 2 שורות בסוף התוכנית:
  - הראשונה רק עם נקודה (A line with single dot).
  - לאחריה שורה עם run; (הסימן נקודה פסיק לאחר המלה run).
- או בשורה הנפרדת סימן : /

טיפוסי המשתנים

אינפורמציה בין תוכנית ה-PL/SQL לבין בסיס הנתונים מועברת דרך משתנים. טיפוסי המשתנים המותרים הם:

- ב- PL/SQL שם משתנה חייב להתחיל באות, ואורכו המקסימאלי יהיה 30 תווים.
- טיפוסי המשתנים המותרים הם:
  - טיפוסים שבהם אנו משתמשים ב-SQL לשדות.,וטיפוסים נוספים כגון BOOLEAN

Datatype	Max Size	Default Size	Description
NUMBER (width,scale)	38	38	Numeric values rounded to a whole number unless a scale is given, i.e. NUMBER(5,2) means a numeric values 5 digits in length allowing for 2 decimal places
VARCHAR2 (width)	32767 *		Variable length character data
CHAR (width)	32767 **		Fixed length character data
DATE			Valid date values
BOOLEAN			Boolean values TRUE and FALSE

לדוגמה :

```
DECLARE
        i       NUMBER;
        str     VARCHAR(15);
```

- במקרים רבים, משתנה משמש לצורך ביצוע מניפולציות על מידע שמאוחסן ב- Relation קיים.
  במקרה זה וכדי למנוע טעות בהגדרת המשתנה, מגדירים את המשתנה בדיוק כטיפוס השדה על ידי הסימון %TYPE.
  - לדוגמא:

```
DECLARE
   FirstNum Table1.Num1%TYPE;
```

מגדיר משתנה ששמו FirstNum שהוא כטיפוס השדה Num1 ב-Table1.

- משתנה יכול להיות מטיפוס רשומה עם כמה שדות. הדרך הפשוטה להגדרה היא על ידי %ROWTYPE:

```
DECLARE
   NumbersTuple Table1%ROWTYPE;
```

יוצר משתנה NumbersTuple להיות רשומה עם השדות Num1,Num2,Num3
(מתוך ידיעה שה-Relation ששמו Table1 הוא עם הסכימה
Table1(Num1,Num2,Num3)).

- ערכו ההתחלתי של כל משתנה, ללא תלות בטיפוסו, הוא NULL.
- ניתן לתת ערכים למשתנים על ידי האופרטור := (נקודותיים שווה).

```
DECLARE
        a NUMBER;
BEGIN
        a:=0;
        a := a + 1;
END;
/
```

SELECT INTO

- השינוי העיקרי ששונה מהצורה של SQL הוא בצורה של פקודת ה-SELECT ב-PL/SQL.
- אחרי ה- SELECT ושדותיו, חייבת לבוא המלה INTO ורשימת משתנים, אחד לכל attribute של ה- SELECT, לשם ייכנסו מרכיבי השורה המוחזרת מה-SELECT.
  - חשוב לציין, **ששורה מוחזרת ולא כמה שורות מוחזרות**, היות וב- PL/SQL ה- SELECT יעבוד רק אם תוצאת השאילתה היא **שורה בודדת**.
  - במקרים שתוצאת השאילתה מחזירה יותר משורה בודדת, יש לעבוד עם **Cursor** (סמן, מצביע), כפי שיתואר בהמשך.

לדוגמה:

```
CREATE TABLE T1(
   e INTEGER,
   f INTEGER
);

INSERT INTO T1 VALUES(1, 3);
INSERT INTO T1 VALUES(2, 4);

                    /* Above is plain SQL; below is the PL/SQL program. */

DECLARE
   a NUMBER;
   b NUMBER;
BEGIN
   SELECT e,f INTO a,b FROM T1 WHERE e>1;
   INSERT INTO T1 VALUES(b,a);
END;
/
```

צפייה בתוצאות

שיטה א' : שימוש בטבלה .

1. ניצור טבלה

```
CREATE TABLE my_debug
( date_created DATE, text VARCHAR2(500));
```

2. את מה שאנו רוצים לבדוק/פלט נכתוב לטבלה שיצרנו

```
DECLARE
      l_x NUMBER := 0;
BEGIN
      INSERT INTO my_debug
            VALUES (SYSDATE,'Before='||TO_CHAR(l_x));
      l_x := l_x + 10;
      INSERT INTO my_debug
            VALUES (SYSDATE,'After='||TO_CHAR(l_x));
END;
.
 Run;
```

3. נתשאל את הטבלה

```
SELECT text
FROM my_debug
ORDER BY date_created;
```

4. התוצאה

```
TEXT
--------------
Before=0
After=10
```

שיטה ב' : שימוש במשתנים מקשרים.

- נוכל להדפיס רק משתנה שהוגדר בצורה המיוחדת:

VARIABLE <name> <type>

- כאשר הטיפוסים הם מסוג: NUMBER, CHAR(n), CHAR.
- נוכל להשים ערכים למשתנה זה, כאשר לפני המשתנה נשים נקודותיים.
- כעת, מחוץ ל-PL/SQL נוכל לכתוב:

PRINT <name>;

1. נצהיר על המשתנה

```
VARIABLE result NUMBER
```

2. נכתוב קטע קוד

```
DECLARE
l_x NUMBER := 0;
BEGIN
 l_x := l_x + 10;
 :result := l_x;
END;
/
```

3. נדפיס את הערך

```
 print result
```

4. התוצאה

```
RESULT
--------------
 10
```

שיטה ג' : שימוש ב DBMS_OUTPUT

- DBMS_OUTPUT - הינה חבילה הכוללת מספר פרוצדורות ופונקציות אשר מכניסות מידע ל- buffer.

1. לאפשר את הפלט

```
 SET serverout ON;
```

2. להשתמש באחת הפונקציות בכדי להדפיס.

Procedure	Description
PUT	Append text to the current line of the output buffer
NEW_LINE	Put and end-of-line marker into the output buffer, this effectively flushes the buffer.
PUT_LINE	Append text and an end-of-line marker to the current line in the output buffer. This also flushes the buffer.

לדוגמה:

```
DBMS_OUTPUT.put_line('Hello World');
```

או

```
DBMS_OUTPUT.put('Hello');
DBMS_OUTPUT.put_line(' World');
```

או

```
DBMS_OUTPUT.put('Hello World');
DBMS_OUTPUT.new_line;
```

דוגמה מלאה:

```
SET SERVEROUT ON;
DECLARE
   X NUMBER;
BEGIN
   X:=10;
   DBMS_OUTPUT.put_line('Hello World');
   DBMS_OUTPUT.NEW_LINE;
   DBMS_OUTPUT.PUT_LINE('ERROR ' || x);
END;
.
RUN;
```

- ניתן לקבל מידע על הפעולה כגון כמה שורות השתנו...

Attribute	Description
SQL%ROWCOUNT	The number of rows processed by the SQL statement
SQL%FOUND	TRUE if at least one row was processed by the SQL statement, otherwise FALSE
SQL%NOTFOUND	TRUE is no rows were processed by the SQL statement, otherwise FALSE.

לדוגמה :

```
BEGIN
      DELETE  FROM t1
      WHERE e>50;
      DBMS_OUTPUT.put_line(TO_CHAR(SQL%ROWCOUNT)|| ' rows deleted');
END;
.
Run;
```

דוגמת קלט/פלט :

```
DECLARE
         i integer;
         j integer;
BEGIN
         i:=2;
         DBMS_OUTPUT.PUT_LINE('This is a message and the value of i is ' || i);
         DBMS_OUTPUT.PUT_LINE('If we want to input, we put ampersand before
the variable');
         j:=&k;
         DBMS_OUTPUT.PUT_LINE('The value of j is '|| j);
END;
```

- נשים לב, כי k אינו משתנה חוקי בתוכנית.
- אנו מציבים את ערך הקלט למשתנה j שאיתו נעבוד.

התניות

- משפט IF נראה כך:

```
IF <condition> THEN
        <statement_list>
ELSE
        <statement_list>
END IF;
```

חלק ה-ELSE אינו הכרחי.

- תנאי מקונן יראה כך:

```
IF <condition_1> THEN
        --Action
ELSIF <condition_2> THEN
        --Action
ELSIF <condition_n> THEN
        --Action
ELSE
        --Action
END IF;
```

דוגמה:

```
DECLARE
   a NUMBER;
   b NUMBER;
BEGIN
   SELECT e,f INTO a,b FROM T1 WHERE e>1;
   IF b=1 THEN
      INSERT INTO T1 VALUES(b,a);
   ELSE
      INSERT INTO T1 VALUES(b+10,a+10);
   END IF;
END;
.
run;
```

דוגמא נוספת:

```
BEGIN
      DELETE  FROM  t1
      WHERE  e>50;
      IF(SQL%ROWCOUNT>0) then
         DBMS_OUTPUT.put_line(TO_CHAR(SQL%ROWCOUNT)|| ' rows deleted');
      ELSE
            DBMS_OUTPUT.put_line('no data to change');
      END IF;
END;
.
run;
```

שימוש ב- SQL%FOUND

```
BEGIN
        DELETE  FROM  t1
        WHERE  e>50;
       IF (SQL%FOUND)  THEN
           DBMS_OUTPUT.put_line(TO_CHAR(SQL%ROWCOUNT)|| ' rows deleted');
       ELSE
            DBMS_OUTPUT.put_line('no data to change');
       END IF;
END;
```

לולאות

- לולאות נכתבות כך:

```
LOOP
   <loop_body> /* A list of statements. */
END LOOP;
```

- כאשר לפחות שורה אחת ב-<loop_body> תכלול משפט יציאה EXIT מהצורה:

```
EXIT WHEN <condition>;
```

הלולאה תצא כאשר התנאי מתקיים,
וכמובן במקום שנאמר לה לצאת.

לדוגמה:

```
DECLARE
   i NUMBER := 1;
BEGIN
   LOOP
      INSERT INTO T1 VALUES(i,i);
      i := i+1;
      EXIT WHEN i>100;
   END LOOP;
END;
.
run;
```

הלולאה תכניס לטבלה T1 את הזוגות (1,1) עד (100,100).

- לולאת WHILE היא מהצורה:

```
WHILE <condition> LOOP
      <loop_body>
END LOOP;
```

לדוגמה :

```
DECLARE
         l_x NUMBER := 10;
BEGIN
         While (l_x <100) LOOP
                  DBMS_OUTPUT.put_line(l_x );
                  l_x := l_x + 10;
         END LOOP;
END;
```

- לולאת FOR היא מהצורה:

```
FOR <var> IN <start>..<finish> LOOP
      <loop_body>
END LOOP;
```

לדוגמה :

```
BEGIN
   FOR counter IN 1 .. 10 LOOP
        DBMS_OUTPUT.put_line(counter);
   END LOOP;
END;
```

משתנה ה-<var> הוא לוקאלי ללולאה ולא צריך להיות מוגדר.
<start> ו-<finish> הינם קבועים.

Cursor (מצביע, סמן)

- Cursor הוא משתנה שרץ על שורות (tuples) ב-relation.
- ה-relation יכול להיות טבלה או אפילו פלט של תשאול מבט.
- אנו מעבירים אל ה-Cursor כל שורה ויכולים לעבד את הערכים בשורה זו.
- **Cursor משמש** בדרך כלל כדי **לאחסן נתונים שאילתות שהחזירו יותר מאשר ערך יחיד.**

- מאפייני הסמן:
  - **%ISOPEN** - מחזיר true אם הסמן פתוח, אחרת, מחזיר false.
  - **%FOUND** - מחזיר true אם הרשומה יובאה בהצלחה, false אם לא.
  - **%NOTFOUND** - מחזיר true אם הרשומה לא יובאה בהצלחה, false אם כן.
  - **%ROWCOUNT** - מחזיר את מספר הרשומות הנכללות בסמן.

- מבנה ה- Cursor:

```
DECLARE
   CURSOR <cursor_name> IS <SQL_Statment>;
   -- variables declaration
BEGIN
   OPEN <cursor_name>;
   LOOP
         FETCH <cursor_name> INTO <var1>,<var2>...
         EXIT WHEN  ……;
   END LOOP;
   CLOSE <cursor_name>;
END;
```

דוגמא:

– תוכנית המדפיסה את כל פרטי העובדים (מספר עובד, שם, משכורת).

```
DECLARE
   CURSOR c1 IS SELECT * FROM emp;
   e_rec emp%rowtype;
BEGIN
   OPEN c1;
   LOOP
      FETCH c1 INTO e_rec;
      EXIT WHEN c1%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('Number: ' || ' ' || e_rec.empno);
      DBMS_OUTPUT.PUT_LINE('Name : ' || ' ' || e_rec.ename);
      DBMS_OUTPUT.PUT_LINE('Salary: ' || ' ' || e_rec.sal);

   END LOOP;
```

```
   CLOSE c1;
END;
/
```

התוכנית הבאה תמחק מ-T1 הנתון את כל השורות בהם ערך השדה הראשון קטן מערך השדה השני, ותכניס את השורה ההפוכה ל-T1.

מספר הערות בקשר לתוכנית:

- בשורות (2) ו-(3) העדפנו להקל על עצמנו בהגדרת טיפוסי המשתנים ולהיות יותר זהירים.
- בשורות (4) עד (8) מופיעה הגדרת ה-Cursor ששמו T1Cursor. בשורה (8) הגדרת ה-Cursor היא FOR UPDATE היות ואנו מעדכנים את T1 בשורה (14) בעזרת ה-Cursor.
- בשורה (10) אנו פותחים את ה-Cursor – שלב הכרחי.
- בשורה (12) אנו **מביאים** דרך ה-Cursor למשתנים לוקאליים. ככלל, משפט FETCH צריך לספק משתנים לכל מרכיבי השורה שאותה אנו מקבלים. היות והשאילתה בשורות (5) עד (7) מייצרת זוגות, אנו סיפקנו שני משתנים, ולא פחות חשוב, בטיפוסם המתאים.
- בשורה (13) בדיקת סיום הלולאה. %NOTFOUND לאחר שם של Cursor נכון רק כאשר פעולת FETCH עם Cursor זה נכשלה למצוא עוד שורות.
- בשורה (14) המחיקה בעזרת התנאי המיוחד ב-WHERE שהוא CURRENT OF T1Cursor.
- בשורה (17) אנו סוגרים את ה-Cursor.

```
 1) DECLARE
 2)         a T1.e%TYPE;
 3)         b T1.f%TYPE;
 4)         CURSOR T1Cursor IS
 5)                 SELECT e, f
 6)                  FROM T1
 7)                 WHERE e < f
 8)                 FOR UPDATE;
 9) BEGIN
10)         OPEN T1Cursor;
11)         LOOP
12)                 FETCH T1Cursor INTO a, b;
13)                 EXIT WHEN T1Cursor%NOTFOUND;
14)                 DELETE FROM T1 WHERE CURRENT OF T1Cursor;
15)                 INSERT INTO T1 VALUES(b, a); /* Insert the reverse tuple:
*/
16)         END LOOP;
        /* Free cursor used by the query. */
17)     CLOSE T1Cursor;
18) END;
19) .
20) run;
```

דוגמא נוספת עם קלט:

```
DECLARE
        CURSOR employee_cur(p_job VARCHAR2)
        IS SELECT empno, ename, sal FROM emp
        WHERE job = p_job;
        r_emp_rec employee_cur%ROWTYPE;
        job varchar2(10);
BEGIN
        job:='&j';
        OPEN employee_cur(job);
        LOOP
                FETCH employee_cur INTO r_emp_rec;
                EXIT WHEN employee_cur%NOTFOUND;
                dbms_output.put_line('empno:  '||r_emp_rec.empno||'  sal:
                '||r_emp_rec.sal);
        END LOOP;
        CLOSE employee_cur;
END;
```

דוגמא נוספת לשימוש בסמן לעדכון:

```
DECLARE
        CURSOR employee_cur(p_job VARCHAR2)
        IS SELECT empno, ename, sal FROM emp
        WHERE job = p_job
        FOR UPDATE;
        r_emp_rec employee_cur%ROWTYPE;
BEGIN
        OPEN employee_cur('CLERK');
        LOOP
                FETCH employee_cur INTO r_emp_rec;
                EXIT WHEN employee_cur%NOTFOUND;
                dbms_output.put_line('empno:  '||r_emp_rec.empno||'  sal
                before:  '||r_emp_rec.sal);
                UPDATE emp SET sal = sal * 1.15
                WHERE CURRENT OF employee_cur;
        END LOOP;
        CLOSE employee_cur;
END;
```

תרגול בנושא בלוקים :

1. כתבו תוכנית (בלוק) שנותנת לכל העובדים של מחלקה 10 תוספת של 15 אחוז במשכורת. הציגו הודעה שמדפיסה כמה עובדים קיבלו העלאה.
2. כתבו בלוק שמקבל תיאור תפקיד חדש ותיאור תפקיד ישן , מוצא את כל בעלי התפקיד הישן ומחליף להם את התפקיד לתיאור החדש. בסיום הפעולה אם התבצעו החלפות – תצא הודעה על מספר ההחלפות שהתבצעו ואם לא תצא הודעה מתאימה.

תרגול בנושא cursor

1. צור תוכנית שמדפיסה (כפלט) את כל העמודות והרשומות של הטבלה DEPT. עשו שימוש CURSOR ובלולאת FOR .
2. צרו תוכנית שמעתיקה את כל הרשומות בטבלה DEPT לטבלה שנקראת בשפ OLD_DEPT ללא שימוש בלולאת FOR . הציגו כפלט כמה שורות הועתקו.

בהצלחה !