פרוצדורות

עמוד 124 .

נראה דוגמה לפרוצדורה addtuple1 שמקבלת את i ומכניסה את השורה ('i,'xxx) לטבלה הבאה:

```
CREATE TABLE T2 (
   a INTEGER,
   b CHAR(10)
);

CREATE PROCEDURE addtuple1(i IN NUMBER) AS
BEGIN
   INSERT INTO T2 VALUES(i, 'xxx');
END addtuple1;
.
run;
```

ניתן ליצור פרוצדורה על ידי CREATE OR REPLACE כדי לבצע מחיקה של הפרוצדורה הקודמת בשם זה, אם הייתה קיימת כזו, ולא לקבל הודעת שגיאה.

הפרוצדורה יכולה לקבל מס' כלשהו של פרמטרים, כל אחד מלווה במצב (mode) וסוג. המצבים האפשריים הם:

- IN (לקריאה בלבד) – ערך הפרמטר כן מתעדכן בערך הארגומנט בכניסה לפרוצדורה, אך הארגומנט אינו מקבל את ערך הפרמטר ביציאה ממנה.
- OUT (לכתיבה בלבד) **-** ערך הפרמטר לא מתעדכן בערך הארגומנט בכניסה לפרוצדורה, אך הארגומנט כן מקבל את ערך הפרמטר ביציאה ממנה.
- INOUT (לקריאה וכתיבה) - ערך הפרמטר מתעדכן בערך הארגומנט בכניסה לפרוצדורה, והארגומנט מקבל את ערך הפרמטר ביציאה ממנה.

הערה חשובה: אין לתת גודל לפרמטר מסוג CHAR או VARCHAR. הגודל ייקבע בהתאם לארגומנט שהועבר בקריאה לפרוצדורה.

אגב, אין חובה לציין את שם הפרוצדורה הנסגרת לאחר ה-END. די לכתוב ;END.

הגדרת משתנים לוקאליים של הפרוצדורה **לא** תתבצע בעזרת DECLARE, אלא בצורה הבאה:

```
CREATE PROCEDURE <procedure_name> (. . . . . .)  AS
<local_var_declarations>
BEGIN
   <procedure_body>
END;
.
run;
```

השורה בסוף ;run מסיימת את יצירת הפרוצדורה. היא אינה מריצה אותה. כדי להריץ את הפרוצדורה נשתמש בתוכנית PL/SQL :

```
BEGIN
   addtuple1(99);
END;
.
run;
```

או בלי PLSQL עובד רק ב- run sql command (מסך שחור)

```
EXEC addtuple1(99);
```

הפרוצדורה הבאה גם מוסיפה שורה ל-T2, אך מקבלת את שני השדות:

```
CREATE PROCEDURE addtuple2(
   x T2.a%TYPE,
   y T2.b%TYPE)
AS
BEGIN
   INSERT INTO T2(a, b)
   VALUES(x, y);
END addtuple2;
.
run;
```

נוסיף שורה (10, 'abc') ל-T2:

```
BEGIN
   addtuple2(10, 'abc');
END;
.
run;
```

נראה דוגמה לשימוש ב-OUT:

```
CREATE TABLE T3 (
   a INTEGER,
   b INTEGER
);

CREATE PROCEDURE addtuple3(a NUMBER, b OUT NUMBER)
AS
BEGIN
   b := 4;
   INSERT INTO T3 VALUES(a, b);
END;
.
run;

DECLARE
   v NUMBER;
BEGIN
   addtuple3(10, v);
   dbms_output.put_line('val of v is : '||v);
END;.
run;
```

נשים לב, כי השמה לפרמטר שהוגדר כ-INOUT/OUT תגרום לארגומנט המתאים להתעדכן בערכו ביציאה מהפרוצדורה. לכן, לעתים ארגומנט עבור פרמטר מסוג OUT/INOUT לא יאותחל, כפי שקורה עם v בתוכנית. ברור כי ארגומנט קבוע לא יכול להיות מועבר לפרמטר OUT/INOUT.

פונקציות

ניתן ליצור פונקציות במקום פרוצדורות. בהגדרת פונקציה, לאחר רשימת הפרמטרים, נכתוב את המלה RETURN ולאחריה הטיפוס המוחזר:

```
CREATE FUNCTION <func_name>(<param_list>) RETURN <return_type> AS ...
```

בגוף הפונקציה נכתוב RETURN <expression>; כדי לצאת מהפונקציה ולהחזיר את הערך של <expression>.

דוגמה לפונקציה:

```
CREATE or replace FUNCTION SalAction(Salary NUMBER,title VARCHAR)
RETURN BOOLEAN IS
Min_sal NUMBER;
Max_sal NUMBER;
BEGIN
   SELECT min(sal) losal, max(sal) hisal INTO Min_sal, Max_sal
   FROM emp WHERE job=title;
   RETURN (salary>= Min_sal) AND(salary<= Max_sal );
END SalAction;
```

קריאה לפונקציה :

```
DECLARE
  X BOOLEAN;
BEGIN
  X:=SalAction(1000,'CLERK');
 IF (x) THEN
      dbms_output.put_line('salary of 1000 is normal');
ELSE
      dbms_output.put_line('salary of 1000 is not normal');
END IF;

END;
```

אופציה אינטראקטיבית:

```
DECLARE
  X BOOLEAN;
  s number;
BEGIN
  s:=&salary;
  X:=SalAction(s,'CLERK');
IF (x) THEN
      dbms_output.put_line('salary of '|| s ||' is normal');
ELSE
      dbms_output.put_line('salary of '|| s ||' is not normal');
END IF;
END;
```

לבדוק איזה פרוצדורות או פונקציות קיימות בבסיס נתונים יש לתת פקודה הבאה:

```
SELECT object_type,object_name
FROM user_objects
WHERE object_type='PROCEDURE'
Or object_type='FUNCTION';
```

כדי למחוק פרוצדורה או פונקציה קיימים, נכתוב:

```
drop procedure <procedure_name>;
drop function <function_name>;
```

Triggers

הדקים הם מבנה ב-PL/SQL הדומה לפרוצדורות. אך פרוצדורה נקראת ישירות דרך קריאה פרוצדוראלית,בעוד שהדק מורץ באופן עקיף ברגע שאירוע ההדק קורה. אירוע ההדק יכול להיות פקודת INSERT, DELETE או UPDATE. התזמון יכול להיות BEFORE או AFTER הפקודה. ההדק יכול להיות ברמת שורה או ברמת משפט. הדק ברמת שורה פועל **פעם אחת מיוחדת** עבור כל שורה שהושפעה מאירוע ההדק. הדק ברמת משפט פועל פעם אחת עבור כל אירוע ההדק.

זהו התחביר ליצירת הדק:

```
CREATE [OR REPLACE] TRIGGER <trigger_name>
   {BEFORE|AFTER} {INSERT|DELETE|UPDATE} ON <table_name>
   [FOR EACH ROW [WHEN (<trigger_condition>)]]
   <trigger_body>
```

ניתן לציין עד שלושה אירועי הדק בעזרת המלה OR. לגבי UPDATE, ניתן לכתוב UPDATE OF ורשימת attribute(s) מ-<Table_name>. במקרה זה האירוע הוא רק בעדכון שדות אלה. דוגמאות:

```
... INSERT ON R ...
... INSERT OR DELETE OR UPDATE ON R ...
... UPDATE OF A, B OR INSERT ON R ...
```

אם אנו כותבים את אופציית FOR EACH ROW, אז ההדק הוא ברמת שורה. אחרת הוא ברמת המשפט. עבור הדק ברמת השורה ניתן להגביל את ההדק בעזרת תנאי SQL שנציין ב- WHEN (ללא SubQueries). התנאי צריך להתקיים כדי שההדק יופעל. אם אנו לא כותבים את חלק ה-WHEN, ההדק מופעל בכל אירוע הדק שקורה.

<trigger_body> הוא בלוק של PL/SQL ולא רצף משפטי SQL.

עלינו לדאוג לא להיכנס למצבים בעייתיים של הדקים התלויים זה בזה, או של אילוצים התלויים בהדקים.

נראה דוגמה להדק אשר לכשנכניס שורה ל-T4 בה ערך השדה הראשון קטן מ-10, ההדק יכניס את השורה ההפוכה ל-T5:

```
CREATE TABLE T4 (a INTEGER, b CHAR(10));
CREATE TABLE T5 (c CHAR(10), d INTEGER);

CREATE TRIGGER trig1
   AFTER INSERT ON T4
   FOR EACH ROW
   WHEN (NEW.a <= 10)
   BEGIN
      INSERT INTO T5 VALUES(:NEW.b, :NEW.a);
   END trig1;
.
run;
```

המשתנים המיוחדים NEW ו-OLD משמשים להתייחסות לשורה החדשה והישנה , בהתאמה. **שימו לב**: בגוף ההדק נשים נקודותיים לפני ה- NEW וה-OLD, בעוד שב-WHEN לא נשים נקודותיים!!!

נשים לב כי מה שכתבתנו, לא מריץ את ההדק, אלא רק יוצר אותו. רק אירוע הדק – הכנסת שורה ל-T4 – תפעיל את ההדק.

למחיקת הדק נכתוב:

```
drop trigger <trigger_name>;
```

כדי לאפשר או להקפיא הדק נכתוב:

```
alter trigger <trigger_name> {disable|enable};
```

לקבלת מידע על הדק:

```
SELECT trigger_name FROM  user_triggers;

SELECT trigger_type,table_name,triggering_event
FROM user_triggers
WHERE trigger_name='<trigger_name>';
```

גילוי שגיאות

PL/SQL לא תמיד אומר מהם טעויות הקומפילציה. כדי לראות אותן נכתוב:

```
show errors procedure <procedure_name>;
show errors trigger <trigger_name>;

SELECT * FROM user_errors;
```

כדי לראות את טעות הקומפילציה האחרונה נכתוב באופן סתמי SHO ERR;.

טיפול בחריגים

עמוד 88 . דוגמאות לחריגים הקיימים הם :

Exception	Oracle Error	Occurs when ...
DUP_VAL_ON_INDEX	-1	Attempt to store a duplicate key
NO_DATA_FOUND	-1403	SELECT statement returns no rows
VALUE_ERROR	-6502	Arithmetic, truncation, conversion error
INVALID_CURSOR	-1001	Invalid cursor operation such as closing an already closed cursor
TOO_MANY_ROWS	-1422	If a SELECT statement returns more than one row
INVALID_NUMBER	-1722	Attempt to convert a non-numeric value to a numeric

לדוגמה:

```
DECLARE
    a NUMBER;
    b NUMBER;
BEGIN
    SELECT e,f INTO a,b FROM T1 WHERE e>1;
    INSERT INTO T1 VALUES(b,a);
EXCEPTION
    WHEN TOO_MANY_ROWS THEN DBMS_OUTPUT.put_line('THERE IS MORE THEN ONE ROW TO
SELECT');

END;
```

במידת הצורך ניתן גם ליצור חריג חדש:

```
DECLARE
   a number ;
  INVALID_STATUS exception;
BEGIN
    a:= 89999;
    IF a>9999 THEN
          RAISE INVALID_STATUS;
     END IF;

EXCEPTION
     WHEN INVALID_STATUS THEN
        dbms_OUTPUT.put_line('not a valid status;('
END;
```

דוגמאות נוספות לפרוצדורות ב-PL/SQL

נראה דוגמה להגדרת פרוצדורה :

```
CREATE OR REPLACE PROCEDURE proc1(empID IN emp.empno%TYPE, new_job IN
emp.job%TYPE) IS
BEGIN
  UPDATE emp
   SET job = new_job
   WHERE empno=empID;
END;
.
run;
```

מה אמורה לעשות פרוצדורה זו?

נראה דוגמה נוספת לפרוצדורה. בדוגמה זו ה-Cursor מקבל פרמטר:

```
CREATE PROCEDURE proc2 (dno NUMBER, percentage NUMBER DEFAULT 0.5) IS
        CURSOR emp_cur (dept_no NUMBER) IS
                SELECT sal FROM EMP WHERE deptno=Dept_no
                FOR UPDATE;
        empsal NUMBER(8);
BEGIN
        OPEN emp_cur(dno);
        LOOP
                FETCH emp_cur into empsal;
                EXIT WHEN emp_cur%NOTFOUND;
                UPDATE Emp SET sal=empsal*((100+percentage)/100)
                WHERE CURRENT OF emp_cur;
        END LOOP;
        CLOSE emp_cur;
END raise_salary;
```

מה תעשה פרוצדורה זו?

דוגמה מעניינת ל-Trigger

Trigger זה מחלק למקרים, ובודק מהו האירוע שהתרחש:

```
CREATE OR REPLACE TRIGGER MYTRIG2
          AFTER DELETE OR INSERT OR UPDATE ON JD11.BOOK
          FOR EACH ROW
BEGIN
  IF DELETING THEN
    INSERT INTO JD11.XBOOK (PREVISBN, TITLE, DELDATE) VALUES (:OLD.ISBN, :OLD.TITLE,
SYSDATE);
  ELSIF INSERTING THEN
    INSERT INTO JD11.NBOOK (ISBN, TITLE, ADDDATE) VALUES (:NEW.ISBN, :NEW.TITLE,
SYSDATE);
  ELSIF UPDATING ('ISBN') THEN
    INSERT INTO JD11.CBOOK (OLDISBN, NEWISBN, TITLE, UP_DATE) VALUES (:OLD.ISBN
:NEW.ISBN, :NEW.TITLE, SYSDATE);
  ELSE /* UPDATE TO ANYTHING ELSE THAN ISBN */
    INSERT INTO JD11.UBOOK (ISBN, TITLE, UP_DATE) VALUES (:OLD.ISBN :NEW.TITLE, SYSDATE);
  END IF
END;
```

תרגילים בנושא פונקציות ופרוצדורות:

1. צרו פרוצדורה שמוחקת שורות מטבלה בשם `old_emp` (שהיא העתק של emp) הפרוצדורה מקבלת job ומוחקת את כל העובדים שיש להם את התפקיד הזה. בנוסף יש להציג כמה רשומות נמחקו. כתבו תוכנית שמפעילה את הפרוצדורה.
2. שנו את הפרוצדורה מהתרגיל הקודם, כך שהיא תחזיר את מספר העובדים שנמחקו ע"י הפרמטר OUT. כתבו תוכנית שתפעיל את הפרוצדורה ותדפיס כמה עובדים נמחקו.

תרגול בנושא TRIGGERS:

צרו טבלה בשם

ChangeEmployee (UserName,TableName,Action,ActionDate)

1) צור טריגר (ברמה כללית)אשר אחרי הוספה/מחיקה/שינוי של טבלה המקורית בשם Employees מוסיף לטבלה ChangeEmployee רשומה חדשה בהתאם.

כדי להכניס UserName יש להשתמש ב:

```
-- Find username of person performing INSERT into table
SELECT SYS_CONTEXT ('USERENV', 'CURRENT_USER') INTO v_username from dual;
```

וכדי להכניס תאריך עכשווי יש להשתמש ב- SYSDATE

2) צור טריגר, שלאחר עדכון של שכר, אם השכר עלה ביותר מ- 50% אז מכניס לטבלה החדשה EMP_NEW מספר עובד, משכורת ישנה , משכורת חדשה.