PL/SQL: Prozeduren und Funktionen

Stephan Karrer

Block-Typen in PL/SQL

Anonym

```
[DECLARE]


BEGIN
  --statements

[EXCEPTION]

END;
```

Prozedur

```
PROCEDURE name
IS

BEGIN
  --statements

[EXCEPTION]

END;
```

Funktion

```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;
[EXCEPTION]

END;
```

Prozeduren und Funktionen

- Sind benannte PL/SQL-Blöcke, die parametrisiert sein können
- Werden üblicherweise in kompilierter Form (Bytecode) in der Datenbank gespeichert, deshalb auch „gespeicherte“ Prozedur/Funktion
- Besitzen ebenfalls Blockstruktur wie die anonymen Blöcke:
  - Optional: Deklarativer Bereich (ohne Schlüsselwort `DECLARE`)
  - Obligatorisch: Ausführbarer Bereich
  - Optional: Exception-Bereich
- Ein Prozeduraufruf steht für eine Anweisung in SQL und PL/SQL
- Ein Funktionsaufruf steht für einen Ausdruck, der einen Wert zurückgibt (analog den vorhandenen SQL-Funktionen)

Funktionen und Prozeduren: Zentrale Speicherung auf dem Server

```
CREATE OR REPLACE FUNCTION half_of_square (param NUMBER)
                  RETURN NUMBER IS
BEGIN
RETURN (param * param)/2 + (param * 4);
END half_of_square;
```

Verwaltet via Data Dictionary

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

SELECT text
    FROM user_source
    WHERE name = 'HALF_OF_SQUARE';
```

Erzeugen einer Prozedur: Syntax

```
CREATE [OR REPLACE] PROCEDURE prozedurname
 [(parameter [mode] datentyp,
   ...)]
IS|AS
  [lokale_deklarationen; ...]
BEGIN
      -- Anweisungen;
END [prozedurname];
```

Beispiel

```
CREATE OR REPLACE PROCEDURE change_name (emp_id NUMBER,
                                         lname VARCHAR2)
IS      -- alternativ "AS"
BEGIN
    DBMS_OUTPUT.PUT_LINE('not yet implemented');
END;
```

Beispiel für eine Prozedur

```
CREATE OR REPLACE PROCEDURE change_name (emp_id NUMBER,
                                         lname VARCHAR2)
IS
BEGIN
    UPDATE employees SET last_name = lname
      WHERE employee_id = emp_id;
    COMMIT;
EXCEPTION
  WHEN OTHERS THEN ROLLBACK;
END;
```

```
-- Aufruf aus SQL:
EXECUTE change_name(100, 'Koenig');      -- SQL+
CALL    change_name(100, 'King');        -- ANSI-Syntax


-- Aufruf aus PL/SQL-Block:
BEGIN
    change_name(100, initcap('koenig');
END;
```

Erzeugen von Funktionen: Syntax

```
CREATE [OR REPLACE] FUNCTION funktionsname
       [(parameter [mode] datentyp, . . .)]
RETURN datentyp IS|AS
       [lokale_deklarationen; . . .]
BEGIN
  -- Anweisungen;
  RETURN ausdruck;
END [funktionsname];
```

- RETURN-Anweisung im Block ist Pflicht
  (eine Funktion liefert einen Wert zurück)

Beispiel für eine Funktion

```
CREATE OR REPLACE FUNCTION get_salary
 (empid employees.employee_id%TYPE) RETURN NUMBER
IS
  sal employees.salary%TYPE := 0;
BEGIN
  SELECT salary
  INTO   sal
  FROM   employees
  WHERE  employee_id = empid;
  RETURN sal;
END get_salary;
```

```
-- Aufruf aus PL/SQL-Block:
DECLARE
  result employees.salary%TYPE;
BEGIN
  result := get_salary(100);
  DBMS_OUTPUT.PUT_LINE(get_salary(110));
END;
```

Übergeben von Parametern

```
CREATE OR REPLACE PROCEDURE
         change_name ( emp_id employees.employee_id%TYPE,
                       lname VARCHAR2)
IS BEGIN
    -- ...
END;
/
EXECUTE change_name(50+50, initcap('koenig'));
```

- Übergabeparameter müssen typisiert angeben werden
- Der konkrete Typ beim Aufruf muss natürlich passen
- Beim Aufruf kann auch ein Ausdruck, ein Funktionsaufruf oder eine Variable verwendet werden

Funktionen und Prozeduren: Default-Werte verwenden

```
CREATE OR REPLACE PROCEDURE add_department(
    deptname departments.department_name%TYPE := 'Test',
    deptloc  departments.location_id%TYPE DEFAULT 2700)
IS
BEGIN
  INSERT INTO departments (department_id,
    department_name, location_id)
  VALUES (departments_seq.NEXTVAL, deptname, deptloc);
END ;
```

```
-- Mögliche Aufrufformen

EXECUTE add_department
EXECUTE add_department ('Muster', 1200)
EXECUTE add_department (deptloc => 1200)
```

Schreibweisen bei der Parameterübergabe

```
-- Parameterangabe via Position:
EXECUTE add_department ('TRAINING', 2500)

-- Benannte Parameterübergabe:
EXECUTE add_department (p_loc=>2400,
                        p_name=>'EDUCATION')
```

Beim Aufruf eines Unterprogramms (Funktion oder Prozedur) können die Übergabeparameter wie folgt angegeben werden:

- Positional: Gleiche Reihenfolge wie in der Definition des Unterprogramms
- Benannt: Listet die Parameter in beliebiger Reihenfolge und benutzt den Zuordnungs-Operator (=>) für den Bezug
- Gemischt: Die ersten Parameter positional, der Rest benannt

Modi für die Parameter

- Der Parametermodus wird in der Parameterleiste der Prozedur angegeben.
- Der IN-Modus ist der Standard.

```
CREATE PROCEDURE proc_name(param_name [mode] datatype)
...
```

Vergleich der Parameter-Modi

- IN-Modus:
  - muss nicht explizit angegeben werden (default)
  - kann im Sub-Programm nicht verändert werden
  - wird per Referenz übergeben
  - Default-Wert kann benutzt werden
- OUT-Modus:
  - muss explizit angeben werden
  - aktueller Parameter muss Variable sein, kein Default-Wert möglich
  - wird in der Regel im Sub-Programm gesetzt für Wert-Rückgabe
  - kann im Sub-Programm gelesen und geschrieben werden
  - Initialisierung durch den Aufrufer wird ignoriert
  - wird bei Rückkehr kopiert, sofern keine Exception auftritt
- IN OUT-Modus:
  - wie OUT-Modus, aber Initialisierung durch den Aufrufer ist nutzbar

Benutzung des OUT Parameter-Modus

```
CREATE OR REPLACE PROCEDURE query_emp
 (p_id      IN  employees.employee_id%TYPE,
  p_name    OUT employees.last_name%TYPE,
  p_salary OUT employees.salary%TYPE) IS
BEGIN
  SELECT  last_name, salary INTO p_name, p_salary
  FROM    employees
  WHERE   employee_id = p_id;
END query_emp;
```

```
DECLARE
  v_emp_name employees.last_name%TYPE;
  v_emp_sal  employees.salary%TYPE;
BEGIN
  query_emp(171, v_emp_name, v_emp_sal);
  DBMS_OUTPUT.PUT_LINE(v_emp_name||' earns '||
    to_char(v_emp_sal, '$999,999.00'));
END;
```

Benutzung des IN OUT Parameter-Modus

```
CREATE OR REPLACE PROCEDURE change_string
  (string1 IN VARCHAR2, string2 IN OUT VARCHAR2) IS
BEGIN
  string2 := string2 || ' ' || string1;
END change_string;
```

Aufruf aus PL/SQL

```
DECLARE
   mystring VARCHAR2(20) := 'Hallo';
BEGIN
 change_string('Otto', mystring);
 DBMS_OUTPUT.PUT_LINE(mystring);
END;
```

Aufruf via SQL*Plus
mit HOST-Variablen

```
VARIABLE string1 VARCHAR2(25)
VARIABLE string2 VARCHAR2(30)
EXECUTE :string1 := 'Otto'
EXECUTE :string2 := 'Hallo'
EXECUTE change_string(:string1, :string2)
PRINT string1 string2
```

Lokale anonyme Funktion

```
DECLARE
   FUNCTION square(original NUMBER)
      RETURN NUMBER IS original_squared NUMBER;
   BEGIN
     original_squared := original * original;
     RETURN original_squared;
   END;
BEGIN
DBMS_OUTPUT.PUT_LINE(square(100));
END;
```

- Zur Modularisierung sind neben Blöcken auch lokale Funktionen und Prozeduren möglich (mit üblichem Gültigkeitsbereich)
- Die Deklaration dieser Funktionen und Prozeduren muss am Ende des Deklarationsteils erfolgen!

Lokale (anonyme) Prozedur

```
DECLARE
  in_string VARCHAR2(100) := 'This is my test string.';
  out_string VARCHAR2(200);

  PROCEDURE double ( original IN VARCHAR2, new_string OUT
                     VARCHAR2 ) AS
    BEGIN
      new_string := original || ' + ' || original;
    EXCEPTION
      WHEN VALUE_ERROR THEN
       DBMS_OUTPUT.PUT_LINE('Output buffer not long enough.');
    END;

BEGIN
  double(in_string, out_string);
  DBMS_OUTPUT.PUT_LINE(in_string || ' - ' || out_string);
END;
```

Unterprogramme mit lokaler Ausnahmebehandlung

Aufrufer

```
PROCEDURE
 PROC1 ...
IS
 ...
BEGIN
 ...
  PROC2(arg1);
 ...
EXCEPTION
 ...
END PROC1;
```

Aufgerufene Prozedur

```
PROCEDURE
 PROC2 ...
IS
 ...
BEGIN
 ...
EXCEPTION
 ...
END PROC2;
```

Exception tritt auf

Exception lokal behandelt

Zurück zum Aufrufer ohne Exception (reguläre Rückkehr)

Unterprogramm mit Ausnahmebehandlung: Beispiel

```
CREATE PROCEDURE add_department(
    p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS
BEGIN
  INSERT INTO DEPARTMENTS (department_id,
    department_name, manager_id, location_id)
  VALUES (DEPARTMENTS_SEQ.NEXTVAL, p_name, p_mgr, p_loc);
  DBMS_OUTPUT.PUT_LINE('Added Dept: '|| p_name);
EXCEPTION
 WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE('Err: adding dept: '|| p_name);
END;
```

Aufrufer

```
CREATE PROCEDURE create_departments IS
BEGIN
  add_department('Media', 100, 1800);
  add_department('Editing', 99, 1800);
  add_department('Advertising', 101, 1800);
END;
```

Unterprogramme mit lokaler Ausnahmebehandlung

Aufrufer

```
PROCEDURE
 PROC1 ...
IS
 ...
BEGIN
 ...
  PROC2(arg1);
 ...
EXCEPTION
 ...
END PROC1;
```

Aufgerufene Prozedur

```
PROCEDURE
 PROC2 ...
IS
 ...
BEGIN
 ...

END PROC2;
```

Exception tritt auf und wird nicht behandelt

Kontrolle geht zurück an Ausnahmebeandlung des Aufrufers

Unterprogramm ohne Ausnahmebehandlung: Beispiel

```
SET SERVEROUTPUT ON
CREATE PROCEDURE add_department_noex(
    p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS
 BEGIN
  INSERT INTO DEPARTMENTS (department_id,
    department_name, manager_id, location_id)
  VALUES (DEPARTMENTS_SEQ.NEXTVAL, p_name, p_mgr, p_loc);
  DBMS_OUTPUT.PUT_LINE('Added Dept: '|| p_name);
 END;
```

Aufrufer

```
CREATE PROCEDURE create_departments_noex IS
BEGIN
  add_department_noex('Media', 100, 1800);
  add_department_noex('Editing', 99, 1800);
  add_department_noex('Advertising', 101, 1800);
END;
```

Funktionen direkt in SQL nutzen

```
CREATE OR REPLACE FUNCTION get_sal
 (p_id employees.employee_id%TYPE) RETURN NUMBER
IS
  v_sal employees.salary%TYPE := 0;
BEGIN
  SELECT salary
  INTO   v_sal
  FROM   employees
  WHERE  employee_id = p_id;
  RETURN v_sal;
END ;
```

```
-- Aufruf aus SQL:
SELECT get_sal(100) FROM DUAL;  -- wie SQL-Funktion
SELECT last_name, salary, get_sal(100) FROM employees
    WHERE employee_id = 110;

- Als Parameter für Prozedur (schachtelbar):
EXECUTE dbms_output.put_line(get_sal(100))
```

Benannte (und gemischte) Parameter-Notation mit SQL (ab 11g)

```
CREATE OR REPLACE FUNCTION f(
  p_parameter_1 IN NUMBER DEFAULT 1,
  p_parameter_5 IN NUMBER DEFAULT 5)
RETURN NUMBER
IS
 v_var number;
BEGIN
  v_var := p_parameter_1 + (p_parameter_5 * 2);
  RETURN v_var;
END f;
```

```
SELECT f(p_parameter_5 => 10) FROM DUAL;
```

Einschränkung bei der Verwendung von Funktionen in SQL

In SQL aufrufbare benutzerdefinierte Funktionen müssen:

- In der Datenbank gespeichert sein
- Nur Eingabe-Parameter verwenden, die SQL-Datentypen nutzen
- Gültige SQL-Datentypen zurückgeben

Beim Aufruf von benutzerdefinierten Funktionen in SQL:

- müssen die Parameter bis Oracle 10g gemäß Ihrer Position angegeben werden
- müssen Sie der Eigentümer der Funktion sein oder das EXECUTE-Privileg besitzen

Nicht erlaubte Seiteneffekte bei der Verwendung von Funktionen in SQL

- Funktionsaufrufe in SELECT-Anweisungen dürfen keine DML-Anweisungen erzeugen.
- Funktionsaufrufe in UPDATE- oder DELETE-Anweisungen dürfen keine Abfragen und keine DML-Anweisungen bzgl. derselben Tabellen erzeugen.
- Funktionsaufrufe in SQL-Anweisungen dürfen keine Transaktionsbeendigung verursachen, d.h. Kein COMMIT oder ROLLBACK ausführen.
  Somit dürfen auch keine DDL-Anweisungen ausgeführt werden, da diese ein implizites COMMIT verursachen.
- Funktionsaufrufe in SQL-Anweisungen dürfen keine Steueranweisungen für Sitzungen (z.B. SET ROLE) bzw. das System (z.B. ALTER SYSTEM) verursachen

Verwenden von Funktionen in SQL

Analog zu den integrierten SQL-Funktionen können benutzerdefinierte Funktionen verwendet werden in:

- SELECT-Liste von Abfragen
- WHERE- und HAVING-Klauseln bei Abfragen und DML-Anweisungen
- ORDER BY-, GROUP BY-, CONNECT BY-, START WITH-Klauseln bei Abfragen
- VALUES-Klausel bei INSERT-Anweisungen
- SET-Klausel bei UPDATE-Anweisungen

Ein Beispiel zu Einschränkung wegen Seiteneffekten

```
CREATE OR REPLACE FUNCTION dml_call_sql(p_sal NUMBER)
   RETURN NUMBER IS
BEGIN
  INSERT INTO employees(employee_id, last_name,
                  email, hire_date, job_id, salary)
  VALUES(1, 'Frost', 'jfrost@company.com',
         SYSDATE, 'SA_MAN', p_sal);
  RETURN (p_sal + 100);
END;
```

Folgende UPDATE-Anweisung verursacht einen Fehler:

```
UPDATE employees
  SET salary = dml_call_sql(2000)
WHERE employee_id = 170;
```

Verwaltung von Prozeduren und Funktionen

```
DROP PROCEDURE raise_salary;    -- Löschen

DROP FUNCTION my_func;    -- Löschen
```

Informationen zu Prozeduren aus dem Data-Dictionary (für Funktionen analog)

```
DESCRIBE user_source       -- SQL*PLUS-Befehl

SELECT text
FROM   user_source
WHERE  name = 'ADD_DEPT' AND type = 'PROCEDURE'
ORDER BY line;
```