Fortgeschrittenes zu Packages

Überladen von Unterprogrammen

- Der gleiche Name kann für mehrere Prozeduren oder Funktionen verwendet werden.
- Die unterschiedlichen Varianten müssen sich bei ihren formalen Parametern bzgl. Anzahl, Reihenfolge oder Typ unterscheiden.
- Dadurch kann Funktionalität unter einheitlicher Namensgebung bei Bedarf erweitert werden.
- Funktioniert nicht für Stand-Alone Unterprogramme.

Beispiel in der Spezifikation

```
CREATE OR REPLACE PACKAGE dept_pkg IS
  PROCEDURE add_department
    (p_deptno departments.department_id%TYPE,
     p_name departments.department_name%TYPE :='unknown',
     p_loc  departments.location_id%TYPE := 1700);

PROCEDURE add_department
  (p_name departments.department_name%TYPE := 'unknown',
   p_loc  departments.location_id%TYPE := 1700);
END dept_pkg;
/
```

Beispiel im Package Body

```
CREATE OR REPLACE PACKAGE BODY dept_pkg  IS
PROCEDURE add_department
  (p_deptno departments.department_id%TYPE,
   p_name   departments.department_name%TYPE := 'unknown',
   p_loc    departments.location_id%TYPE := 1700) IS
  BEGIN
    INSERT INTO departments(department_id,
      department_name, location_id)
    VALUES  (p_deptno, p_name, p_loc);
  END add_department;

PROCEDURE add_department
  (p_name   departments.department_name%TYPE := 'unknown',
   p_loc    departments.location_id%TYPE := 1700) IS
  BEGIN
    INSERT INTO departments (department_id,
      department_name, location_id)
    VALUES (departments_seq.NEXTVAL, p_name, p_loc);
  END add_department;
 END dept_pkg; /
```

Überladen und das Package STANDARD

- Das Standard Package definiert die PL/SQL-Umgebung und eingebaute Funktionalitäten.
- Die meisten Standard-Funktionen sind überladen. Ein Beispiel ist die `TO_CHAR` Funktion:

```
FUNCTION TO_CHAR (p1 DATE) RETURN VARCHAR2;
FUNCTION TO_CHAR (p2 NUMBER) RETURN VARCHAR2;
FUNCTION TO_CHAR (p1 DATE, P2 VARCHAR2) RETURN VARCHAR2;
FUNCTION TO_CHAR (p1 NUMBER, P2 VARCHAR2) RETURN
   VARCHAR2;
. . .
```

- Ein PL/SQL-Unterprogramm mit dem gleichen Namen wie ein Standard-Programm überlagert die Standardvariante im lokalen Kontext, ausser man qualifiziert über den Package-Namen.

Korrekte Reihenfolge der Elemente

```
CREATE OR REPLACE PACKAGE BODY forward_pkg IS
  PROCEDURE award_bonus(. . .) IS
  BEGIN
    calc_rating (. . .);    --illegal reference
  END;

PROCEDURE calc_rating (. . .) IS
  BEGIN
    ...
  END;
END forward_pkg;
/
```

Umgehen mittels Vorwärtsdeklaration

Im Body wird eine Vorwärtsdeklaration als eine reine Spezifikation, terminiert durch Semikolon, vereinbart.

```
CREATE OR REPLACE PACKAGE BODY forward_pkg IS
 PROCEDURE calc_rating (...);-- forward declaration

 -- Subprograms defined in alphabetical order

 PROCEDURE award_bonus(...) IS
 BEGIN
   calc_rating (...);
   . . .
 END;

 PROCEDURE calc_rating (...) IS
 BEGIN
   . . .
 END;
END forward_pkg;
```

Nutzen von Package-Funktionen in SQL

```
CREATE OR REPLACE PACKAGE taxes_pkg IS
  FUNCTION tax (p_value IN NUMBER) RETURN NUMBER;
END taxes_pkg;
/
CREATE OR REPLACE PACKAGE BODY taxes_pkg IS
  FUNCTION tax (p_value IN NUMBER) RETURN NUMBER IS
    v_rate NUMBER := 0.08;
  BEGIN
    RETURN (p_value * v_rate);
  END tax;
END taxes_pkg;
/
```

```
SELECT taxes_pkg.tax(salary), salary, last_name
FROM   employees;
```

Package-Initialisierung

Ein Block nach dem Deklarationsteil kann zur Initialisierung der Package-Komponenten genutzt werden.

```
CREATE OR REPLACE PACKAGE taxes IS
  v_tax   NUMBER;
  ...  -- declare all public procedures/functions
END taxes;
/
CREATE OR REPLACE PACKAGE BODY taxes IS
  ... -- declare all private variables
  ... -- define public/private procedures/functions
 BEGIN
   SELECT   rate_value INTO v_tax
   FROM     tax_rates
   WHERE    rate_name = 'TAX';
END taxes;
/
```

Persistenter Zustand des Package

Die Variablenwerte eines Pakets definieren seinen Zustand. Hier gilt:

- Initialisierung erfolgt beim ersten Laden
- Persistent (default) für die gesamte Session:
  - Gespeichert in der User Global Area (UGA)
  - Einmalig je Session
  - Verändert sich durch den Aufruf von Unterprogrammen oder Variablenmodifikationen
- Nicht persistent für die gesamte Session, sondern nur je Unterprogrammaufruf bei Verwendung von:
  `PRAGMA SERIALLY_REUSABLE`
  in der Spezifikation und, falls vorhanden, im Body

Persistenter Zustand der Package Variablen: Beispiel

		State for Scott		*State for Jones*	
Time	Events	v_std_commission [variable]	MAX (comm_pc) [column]	v_std_commission [variable]	MAX (comm_pc) [Column]
9:00	Scott> EXECUTE comm_pkg.reset_comm(0.25)	0.10 0.25	0.4	-	0.4
9:30	*Jones> INSERT INTO employees(last_name, commission_pct) VALUES('Madonna', 0.8);*	0.25	0.4		0.8
9:35	*Jones> EXECUTE comm_pkg.reset_comm (0.5)*	0.25	0.4	0.1 0.5	0.8
10:00	*Scott> EXECUTE comm_pkg.reset_comm(0.6)* *Err -20210 'Bad Commission'*	0.25	0.4	0.5	0.8
11:00 11:01 12:00	*Jones> ROLLBACK;* *EXIT ...* *EXEC comm_pkg.reset_comm(0.2)*	0.25 0.25 0.25	0.4 0.4 0.4	0.5 - 0.2	0.4 0.4 0.4

Persistenter Zustand eines Package Cursor: Beispiel

```
CREATE OR REPLACE PACKAGE curs_pkg IS -- Package spec
  PROCEDURE open;
  FUNCTION next(p_n NUMBER := 1) RETURN BOOLEAN;
  PROCEDURE close;
END curs_pkg;

CREATE OR REPLACE PACKAGE BODY curs_pkg IS
-- Package body
  CURSOR cur_c IS
    SELECT employee_id FROM employees;
  PROCEDURE open IS
  BEGIN
    IF NOT cur_c%ISOPEN THEN
      OPEN cur_c;
    END IF;
  END open;
. . .  -- code continued on next slide
```

Beispiel (Fortsetzung)

```
. . .
  FUNCTION next(p_n NUMBER := 1) RETURN BOOLEAN IS
    v_emp_id employees.employee_id%TYPE;
  BEGIN
    FOR count IN 1 .. p_n LOOP
      FETCH cur_c INTO v_emp_id;
      EXIT WHEN cur_c%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('Id: ' ||(v_emp_id));
    END LOOP;
    RETURN cur_c%FOUND;
  END next;
  PROCEDURE close IS
    BEGIN
      IF cur_c%ISOPEN THEN
        CLOSE cur_c;
      END IF;
    END close;
END curs_pkg;
```

Ausführung des CURS_PKG Package

Enter SQL Statement:

```
1  -- Make sure to enable SET SERVEROUTPUT ON using SQL*Plus or SQL Developer.
2
3  EXECUTE curs_pkg.open
4  DECLARE
5    v_more BOOLEAN := curs_pkg.next(3);
6  BEGIN
7    IF NOT v_more THEN
8      curs_pkg.close;
9    END IF;
10 END;
11 /
12
13
14
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

```
anonymous block completed
anonymous block completed
Id: 100
Id: 101
Id: 102

anonymous block completed
anonymous block completed
Id: 103
Id: 104
Id: 105
```