Programmieren mit PL/SQL

RETURNING-Klausel
Rekursive Funktionen
Exceptions

RETURNING - Klausel

```
SQL> desc oe.employees
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 EMPLOYEE_ID                               NOT NULL NUMBER(6)
 FIRST_NAME                                         VARCHAR2(20)
 LAST_NAME                                 NOT NULL VARCHAR2(25)
 EMAIL                                     NOT NULL VARCHAR2(25)
 PHONE_NUMBER                                       VARCHAR2(20)
 HIRE_DATE                                 NOT NULL DATE
 JOB_ID                                    NOT NULL VARCHAR2(10)
 SALARY                                             NUMBER(8,2)
 COMMISSION_PCT                                     NUMBER(2,2)
 MANAGER_ID                                         NUMBER(6)
 DEPARTMENT_ID                                      NUMBER(4)

SQL> SELECT COUNT(*) FROM oe.employees;

  COUNT(*)
----------
       107
```

RETURNING - Klausel

```
-- vorher Tabelle EMPLOYEES in eigenes Schema kopieren
-- CREATE TABLE employees AS SELECT * FROM oe.employees;
DECLARE
  TYPE EmpRec IS RECORD (last_name employees.last_name%TYPE,
                         salary employees.salary%TYPE);
  emp_info EmpRec;
  emp_id NUMBER := 100;
BEGIN
  UPDATE employees SET salary = salary * 1.1
   WHERE employee_id = emp_id
   RETURNING last_name, salary
   INTO emp_info;

  dbms_output.put_line('Just gave a raise to ' ||
    emp_info.last_name ||
    ', who now makes ' || emp_info.salary);
  ROLLBACK;
END;
```

RETURNING - Klausel

```
SQL> set serveroutput on
SQL> /
Just gave a raise to King, who now makes 26400

PL/SQL procedure successfully completed.
```

RETURNING-Klausel und Sequenzen

```
DROP SEQUENCE emp_seq;
CREATE SEQUENCE emp_seq START WITH 7935;
DECLARE
  v_ename VARCHAR2(10);
  v_empno NUMBER;
BEGIN
  INSERT INTO emp (empno,ename,job,mgr,hiredate,sal,comm,deptno)
  VALUES (emp_seq.nextval,'LEOBERT','CLERK',
          7782,SYSDATE,1000,0,10)
  RETURNING empno,ename INTO v_empno,v_ename;

  dbms_output.put_line('Die EMPNO von ' || v_ename
                       ||' ist ' || v_empno);
  ROLLBACK;
END;
/
```

RETURNING-Klausel und Sequenzen

```
SQL> DROP SEQUENCE emp_seq;

Sequence dropped.

SQL> CREATE SEQUENCE emp_seq START WITH 7935;

Sequence created.

SQL> DECLARE
  2    v_ename VARCHAR2(10);
  3    v_empno NUMBER;
  4  BEGIN
  5    INSERT INTO emp (empno,ename,job,mgr,hiredate,sal,comm,deptno)
  6    VALUES (emp_seq.nextval,'LEOBERT','CLERK',7782,SYSDATE,1000,0,10)
  7    RETURNING empno,ename INTO v_empno,v_ename;
  8
  9    dbms_output.put_line('Die EMPNO von ' || v_ename ||' ist ' || v_empno);
 10    ROLLBACK;
 11  END;
 12  /
Die EMPNO von LEOBERT ist 7935

PL/SQL procedure successfully completed.
```

Rekursive Module

```
DECLARE
  PROCEDURE Show_Emps(p_eno IN NUMBER, p_dashes IN VARCHAR2)
  IS
    CURSOR empc IS SELECT empno,ename FROM emp WHERE mgr = p_eno;
    next_p_dashes VARCHAR2(10);
  BEGIN
    -- Parameter p_dashes gibt Einrueckung an ,z.B. '----':
    FOR empcrec IN empc LOOP
      DBMS_Output.Put_Line (p_dashes||to_char(empcrec.empno)||
                            ' '||empcrec.ename);
      next_p_dashes := p_dashes||'--';
      Show_Emps(empcrec.empno,next_p_dashes);
    END LOOP;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN NULL;
  END;
BEGIN
  Show_Emps(7839,'');
END;
/
```

Rekursive Module

```
SQL> /
7566 JONES
--7788 SCOTT
----7876 ADAMS
--7902 FORD
----7369 SMITH
7698 BLAKE
--7499 ALLEN
--7521 WARD
--7654 MARTIN
--7844 TURNER
--7900 JAMES
7782 CLARK
--7934 MILLER

PL/SQL procedure successfully completed.
```

Alle Angestellten mit der EMPNO von KING im Feld MGR und deren Abhängige werden angezeigt

Exceptions

Exceptions (Ausnahmebedingungen) sind von Java her bekannt.

Ausnahmebedingungen werden durch Fehler (oder Warnungen) verursacht. Gründe hierfür sind:

1. Fehlermeldungen hervorgerufen durch das System
2. Fehler, die durch den User verursacht werden
3. Fehler hervorgerufen durch die Programmlogik

Es gibt vier Arten von PL/SQL-Exceptions

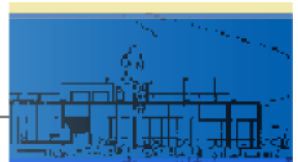
- **Named system exceptions** (Vordefinierte Ausnahmebedingungen, Named Predefined Exceptions): Fehlermeldungen die in PL/SQL benannt sind und durch den PL/SQL-Code oder das DBMS aufgerufen werden
- **Unnamed system exceptions** (Nicht vordefinierte Ausnahmebedingungen, Non-predefined Exceptions): Nur die häufigsten Fehlermeldungen besitzen eine PL/SQL-Bezeichnung
- **Named programmer defined exceptions** (Benannte benutzerdefinierte Ausnahmebedingungen, Named User-Defined Exceptions): werden im Deklarationsteil des PL/SQL-Codes angegeben und werden explizit durch den Programmierer im PL/SQL-Code aufgerufen
- **Unnamed programmer defined exceptions** (Unbenannte benutzerdefinierte Ausnahmebedingungen, Unnamed User-Defined Exceptions): Diese Ausnahmen werden direkt am Server im PL/SQL-Code deklariert und aufgerufen (durch die RAISE_APPLICATION_ERROR-Prozedur).

Named system exceptions

Exception	Erklärung	Oracle Error
at *str* line num	This is usually the last of a message stack and indicates where a problem occurred in the PL/SQL code.	ORA-06512
CURSOR_ALREADY_OPEN	ein Cursor ist bereits offen	ORA-06511
DUP_VAL_ON_INDEX	wenn versucht wird in eine Tabellenspalte, auf die ein unique Index gelegt worden ist, einen Wert einzufügen, der bereits existiert.	ORA-00001
INVALID_CURSOR	wenn eine ungültige Cursoroperation ausgeführt wird. Z.B. wenn versucht wird, einen nicht geöffneten Cursor zu schließen.	ORA-01001
INVALID_NUMBER	wenn in einem SQL Statement versucht wird eine Zeichenkette in eine Zahl zu konvertieren und die Kette ungültige Zeichen enthält. In Non- SQL -Statements wird in diesem Fall die Exception VALUE_ERROR gesetzt.	ORA-01722
NO_DATA_FOUND	wenn bei einem SELECT INTO Statement keine Zeilen zurückgegeben werden. Beim FETCH Statement wird diese Bedingung nicht gesetzt. AVG und SUM geben immer einen Wert oder NULL zurück, daher wird die Bedingung NO_DATA_FOUND in diesem Fall nicht gesetzt.	ORA-01403
PROGRAM_ERROR	wenn ein PL/SQL internes Problem entstanden ist.	ORA-06501
TOO_MANY_ROWS	wenn bei einem SELECT INTO mehr als eine Zeile zurückgegeben wird.	ORA-01422

VALUE_ERROR	wenn ein arithmetischer, ein conversion-, truncation- oder constraint-Error auftritt. Z.B. wenn ein Spaltenwert in eine Charactervariable gelesen wird und diese Variable kürzer deklariert worden ist als die Spalte. Das folgende Beispiel liefert einen VALUE_ERROR:
	`DECLARE`
	`  my_empno NUMBER(4);`
	`  my_ename CHAR(10);`
	`BEGIN`
	`  my_empno := 'HALL'; --raises VALUE_ERROR`
	
	`  --In SQL statements, INVALID_NUMBER is`
	`  --raised instead.`
		ORA-06502

ZERO_DIVIDE	Division durch 0	ORA-01476



Benutzerdefinierte Fehlermeldung

- Benutzerdefinierte Fehlermeldungen, die an die Client Application zurückgegeben werden, können mittels der Built-In Procedure Raise_Application_Error generiert werden.

```
Raise_Application_Error (Error_Number,
                         Error_Text,
                         [Keep_Error_Stack])
```

Benutzerdefinierte Fehlermeldung

```
CREATE PROCEDURE raise_salary (emp_id NUMBER, increase NUMBER)
AS
   current_salary   emp.sal%TYPE;
BEGIN
   SELECT sal INTO current_salary FROM emp
    WHERE empno = emp_id;
   IF current_salary IS NULL THEN
      /* Issue user-defined error message. */
      raise_application_error(-20101, 'Salary is missing');
   ELSE
      UPDATE emp SET sal = current_salary + increase
       WHERE empno = emp_id;
   END IF;
END raise_salary;
```

```
ERROR at line 1:
ORA-20101: Salary is missing
ORA-06512: at "RAISE_SALARY", line 8
ORA-06512: at line 1
```

Benutzerdefinierte Ausnahmebedingungen

```
DECLARE
   past_due    EXCEPTION;          -- declare the exception

...
BEGIN
   ...
   RAISE past_due;                 -- fire the exception
   ...
EXCEPTION
   WHEN past_due THEN              -- handler sequence_of_statements1
      ...
    WHEN exception_name2 THEN      -- another handler
                                      sequence_of_statements2
    ...
    WHEN OTHERS THEN               -- optional handler
    ...                               sequence_of_statements3
END;
```

Benutzerdefinierte Ausnahmebedingungen

```
CREATE PROCEDURE raise_salary (emp_id NUMBER, increase NUMBER) AS
   current_salary    emp.sal%TYPE;
   unknown_sal       EXCEPTION;

BEGIN
   SELECT sal INTO current_salary FROM emp
    WHERE empno = emp_id;
   IF current_salary IS NULL THEN
      /* Issue user-defined error message. */
      raise unknown_sal;
   ELSE
      UPDATE emp SET sal = current_salary + increase
       WHERE empno =emp_id;
   END IF;

EXCEPTION
      WHEN unknown_sal THEN
         UPDATE EMP SET SAL = 0
          WHERE EMPNO = EMP_ID;

END raise_salary;
```

Verschachtelung von Exceptions

```
begin
   ...
   declare
     a number(1);
   begin
     a := 55;
   exception when VALUE_ERROR then
     raise VALUE_ERROR;
   end;
   ...
exception when VALUE_ERROR then
   ...
end;
```

SQLCODE und SQLERRM

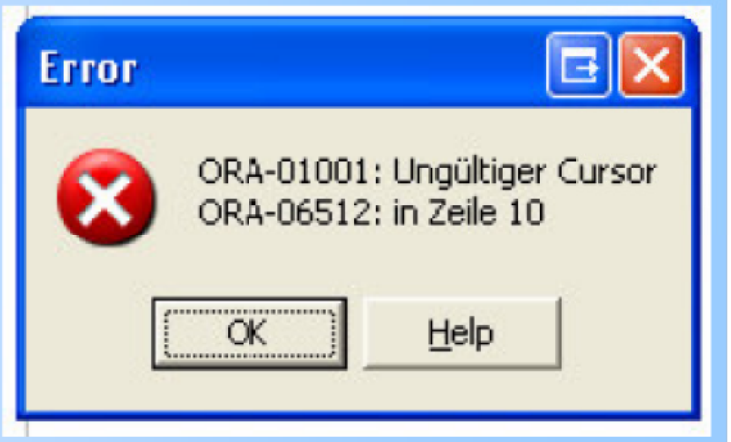
```
CREATE TABLE err_test
    (widget_name    VARCHAR2(100)
    ,widget_count   NUMBER
    ,CONSTRAINT no_small_numbers CHECK
        (widget_count > 1000));
BEGIN
    INSERT INTO err_test (widget_name, widget_count)
    VALUES ('Athena',2);
EXCEPTION
    WHEN OTHERS THEN
    IF SQLCODE = -2290
        AND SQLERRM LIKE '%NO_SMALL_NUMBERS%'
    THEN
        DBMS_OUTPUT.PUT_LINE('widget_count is too
            small');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Exception not hooked,'
            ||'SQLcode='||SQLCODE);
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
    END IF;
END;
```

Built-in Funktionen, die den SQL-Fehlercode sowie den Text der aktuellen Exception zur Verfügung stellen. Wird in der WHEN OTHERS Klausel verwendet

Exception-Handling

Wozu Exception Handling?

- Definiertes Behandeln von Fehlern
- Kontrolliertes Beenden von Prozessen im Fehlerfall
- Protokollierung
- Vermeidung kryptischer Fehlermeldungen



Kein Exception Handling ist besser als schlechtes Exception Handling!

Exception-Handling: Bad Practises

- **Ignoranz:**

```
EXCEPTION
   WHEN OTHERS THEN
      NULL;
```

- **Default-Annahme:**

```
EXCEPTION
   WHEN OTHERS THEN
      o_auftrag_mit_verlust := FALSE;
```

Exception-Handling: Bad Practises

- **Falsche Annahme:**

```
DECLARE
   str VARCHAR2(25);
BEGIN

   SELECT v
     INTO str
     FROM test
    WHERE v = 'Heute ist ein schöner Tag.';

    dbms_output.put_line('String in Tabelle vorhanden');

EXCEPTION
   WHEN OTHERS THEN
     dbms_output.put_line('String nicht in Tabelle vorhanden');
END;
```

Auswirkungen der Bad Practises

- Fehlermeldungen gehen verloren
- Falsche Fehlermeldungen
- Fehlerposition im Code nicht lokalisierbar
- Falsche Programmlogik
- Aufrufstack nicht nachvollziehbar

- *Hoher Aufwand beim Debugging*
- *Schlechter Support für den Kunden*

Ziele von gutem Exception Handling

- Genaue Identifizierung des Fehlers
- Frühzeitiges Erkennen von Fehlern
- Lokalisierung innerhalb weniger Code-Zeilen
- Nachvollziehbarkeit
- Reproduzierbarkeit
- Fehlererkennung oft ohne Debugging möglich

- *Schnelles „PACK AN“*
- *Den Anwender besser verstehen*

Bewertung von Exceptions 1

- **Exceptions als Programmiermittel**
  - Bsp: SELECT INTO abfangen mit NO_DATA_FOUND und Ersatzwert
  - GOTO-Funktionalität
  - Wird nicht weiter betrachtet
- **Ora-Fehler**
  - Fehlermeldung des DBMS
  - Eingriff von Entwickler oder Administrator notwendig
  - Wird immer protokolliert