PL/SQL: Exceptions

Stephan Karrer

Ausnahmebehandlung

```
DECLARE
 x NUMBER := 1;
 y NUMBER := 2;
 z NUMBER := 3;
BEGIN
 p(x);
 y := 1 / z;
 z := x + y;
EXCEPTION
 WHEN OTHERS THEN
 /* Handler to execute for all errors */

END;
```

- Jeder Laufzeitfehler wird durch eine Exception propagiert
- Jede Exception hat einen Fehlercode, z.B. ORA-1403 und eine Nachricht, z.B. No Data Found

Exception Propagation

- Falls der aktuelle Block einen Handler für die Exception hat, wird diese ausgeführt und anschließend an den aufrufenden Block zurückgekehrt
- Falls kein Handler für die Exception im aktuellen Block vorhanden ist, wird die Exception an den aufrufenden Block propagiert
- Falls kein aufrufender Block vorhanden ist, wird die Exception an die Laufzeitumgebung propagiert und führt zum Abbruch des Programms
- Falls die Exception während der Ausführung der deklarativen Sektion oder eines Exception-Handlers auftritt, wird abgebrochen und direkt an den aufrufenden Block propagiert

Beispiel: Exception im Deklarationsteil

```
BEGIN  -- Outer Block

  DECLARE  -- Inner Block
    -- Raises an error:
    credit_limit CONSTANT NUMBER(3) := 5000;
  BEGIN
    NULL;
  EXCEPTION WHEN OTHERS THEN
    -- Cannot catch exception. This handler is never invoked.
    DBMS_OUTPUT.PUT_LINE('Exception in a declaration');
  END;

EXCEPTION
  WHEN OTHERS THEN   -- Catch exception from inner block
    DBMS_OUTPUT.PUT_LINE('Error!');
END;
```

Ausnahmebehandlung: Benannte Exceptions

```
DECLARE
    sal_calc NUMBER(8,2);
BEGIN
    INSERT INTO employees_temp VALUES (301, 2500, 0);
    SELECT salary / commission_pct INTO sal_calc
        FROM employees_temp
        WHERE employee_id = 301;
    INSERT INTO employees_temp VALUES (302,
                                       sal_calc/100, .1);
EXCEPTION
    WHEN ZERO_DIVIDE THEN    NULL;
END;
```

- Vordefinierte und damit benannte Exceptions können explizit adressiert und geworfen werden
- Exceptions lediglich fangen ohne Behandlung ist schlechter Stil !

Vordefinierte (benannte) Exceptions

Exception Name	ORA Error	SQLCODE	Raised When ...
DUP_VAL_ON_INDEX	00001	-1	A program attempts to store duplicate values in a column that is constrained by a unique index.
INVALID_CURSOR	01001	-1001	A program attempts a cursor operation that is not allowed, such as closing an unopened cursor.
INVALID_NUMBER	01722	-1722	n a SQL statement, the conversion of a character string into a number fails because the string does not represent a valid number. (In procedural statements, VALUE_ERROR is raised.) This exception is also raised when the LIMIT-clause expression in a bulk FETCH statement does not evaluate to a positive number.
LOGIN_DENIED	01017	-1017	A program attempts to log on to the database with an invalid username or password.
NO_DATA_FOUND	01403	+100	A SELECT INTO statement returns no rows, or your program references a deleted element in a nested table or an uninitialized element in an index-by table. Because this exception is used internally by some SQL functions to signal completion, you must not rely on this exception being propagated if you raise it within a function that is invoked as part of a query.

und etliche weitere: siehe PL/SQL-Referenz

Assoziation einer Exception mit Oracle Laufzeitfehler

```
DECLARE
    deadlock_detected EXCEPTION;
    PRAGMA EXCEPTION_INIT(deadlock_detected, -60);
BEGIN
    /* Some operation that causes an ORA-00060 error */
EXCEPTION
    WHEN deadlock_detected THEN
        /* handle the error */
END;
```

- Nicht vordefinierte Exceptions können selbst benannt werden (mit Fehler assoziiert)
- Die Exception muss zuerst deklariert werden, bevor via PRAGMA-Anweisung die Assoziierung erfolgt
- Die sog. Vordefinierten sind auf diese Weise im Standard-Package benannt

Werfen von vordefinierten Exceptions

```
DECLARE
    acct_type INTEGER := 7;
BEGIN
    IF acct_type NOT IN (1, 2, 3) THEN
        RAISE INVALID_NUMBER; -- raise predefined exception
    END IF;
EXCEPTION
    WHEN INVALID_NUMBER THEN
        DBMS_OUTPUT.PUT_LINE
                ('HANDLING INVALID INPUT BY ROLLING BACK.');
        ROLLBACK;
END;
```

- Nur benannte Exceptions können explizit mit RAISE geworfen werden (keine Möglichkeit die Fehlernummer zu verwenden!)
- Somit für sonstige Exceptions:
  - Benennen der Exception
  - Oder Verwenden von RAISE_APPLICATION_ERROR

Eigene Exceptions

```
DECLARE
    out_of_stock EXCEPTION;
    number_on_hand NUMBER := 0;
BEGIN
    IF number_on_hand < 1 THEN
        RAISE out_of_stock; -- raise an defined exception
    END IF;
EXCEPTION
    WHEN out_of_stock THEN
    -- handle the error
        DBMS_OUTPUT.PUT_LINE('Encountered out-of-stock error.');
END;
```

- Eigene, applikationsspezifische, Exceptions müssen deklariert und explizit geworfen werden
- Standardmäßig ist der Fehlercode 1 und die Fehlernachricht „User-Defined Exception“ bei eigenen Exceptions

Gültigkeitsbereich eigener Exceptions

```
DECLARE
  past_due EXCEPTION;  -- notwendig, ansonsten Syntax-Fehler im Handler
BEGIN
    DECLARE ---------- sub-block begins
        past_due EXCEPTION; -- this declaration prevails
        due_date DATE := SYSDATE - 1;
        todays_date DATE := SYSDATE;
    BEGIN
        IF due_date < todays_date THEN
            RAISE past_due; -- this is not handled
        END IF;
    END; ------------ sub-block ends
EXCEPTION
  -- Does not handle raised exception
  WHEN past_due THEN DBMS_OUTPUT.PUT_LINE
                          ('Handling PAST_DUE exception.');
  WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE
        ('Could not recognize PAST_DUE_EXCEPTION');
END;
```

- Die Gültigkeit eigener Exceptions (User Defined) ist auf den Block und untergeordnete Blöcke begrenzt.

Gültigkeitsbereich gilt auch für selbst benannte interne Exceptions

```
DECLARE
  deadlock_detected EXCEPTION;
BEGIN
    DECLARE ---------- sub-block begins
        deadlock_detected EXCEPTION;
        PRAGMA EXCEPTION_INIT(deadlock_detected, -60);
    BEGIN
        RAISE deadlock_detected; -- this is not handled
    END; ------------- sub-block ends
EXCEPTION
  -- Does not handle raised exception
  WHEN deadlock_detected  THEN
      DBMS_OUTPUT.PUT_LINE('Handling deadlock_detected exception.');
  WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Could not recognize deadlock_detected_EXCEPTION');
END;
```

- Allerdings können wir mittels Packages den Gültigkeitsbereich globalisieren (siehe Package-Konzept)

Verwendung von RAISE_APPLICATION_ERROR

```
DECLARE
    num_tables NUMBER;
BEGIN
    SELECT COUNT(*) INTO num_tables FROM USER_TABLES;
    IF num_tables < 1000 THEN
    -- Issue your own error code (ORA-20101) + error message
RAISE_APPLICATION_ERROR
             (-20101, 'Expecting at least 1000 tables');
    ELSE
    -- Do rest of processing (for nonerror case)
      NULL;
    END IF;
END;
```

- Damit können Fehlernummern im Bereich [-20.999 , -20.000] und eigene Meldungstexte verwendet werden
- Ist im Paket DBMS_STANDARD definiert

Erneutes Werfen einer Exception

```
/* … */
BEGIN ---------- sub-block begins
    IF current_salary > max_salary THEN
        RAISE salary_too_high; -- raise the exception
    END IF;
EXCEPTION
    WHEN salary_too_high THEN
    -- first step in handling the error
    DBMS_OUTPUT.PUT_LINE('Salary ' || current_salary ||
                              ' is out of range.');
    DBMS_OUTPUT.PUT_LINE
                ('Maximum salary is ' || max_salary || '.');
    RAISE; -- reraise the current exception
END; ------------ sub-block ends
/* … */
```

- Für erneutes Werfen derselben aus dem Exception-Handler genügt ein einfaches RAISE

Kombination von Handlern

```
BEGIN
  /* … */
EXCEPTION
    WHEN exception1 THEN -- handler for exception1
    /* sequence_of_statements1 */
    WHEN exception2 THEN -- another handler for exception2
    /* sequence_of_statements2 */
    ...
    WHEN OTHERS THEN -- optional handler for all other errors
    /* sequence_of_statements3 */
END;
```

- Mehrere Handler können definiert werden
- Für jede Exception darf max. 1 Handler definiert sein
- Der OTHERS-Handler muss am Ende stehen, falls vorhanden

Kombination von Handlern

```
BEGIN
    /* … */
EXCEPTION
    WHEN over_limit OR under_limit OR VALUE_ERROR THEN
        DBMS_OUTPUT.PUT_LINE('wrong value');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('other error');
END;
```

- Ein Handler kann verschiedene Exceptions behandeln, um Code-Duplizierung zu vermeiden

Auswertung

- SQLCODE liefert die Fehlernummer:
  - 0 kein Fehler,
  - 1 benutzerdef. Exception,
  - 100 NO-DATA-FOUND,
  - -xxx Sonstiger Oracle-Fehler

- SQLERRM liefert die zugehörige Fehlermeldung

```
DECLARE
    name EMPLOYEES.LAST_NAME%TYPE;
    v_code NUMBER;
    v_errm VARCHAR2(64);
BEGIN
    SELECT last_name INTO name
        FROM EMPLOYEES   WHERE EMPLOYEE_ID = -1;
EXCEPTION
    WHEN OTHERS THEN
    v_code := SQLCODE;
    v_errm := SUBSTR(SQLERRM, 1, 64);
    DBMS_OUTPUT.PUT_LINE('Error code '||v_code|| ': '||v_errm);

END;
```

Auswertung

- Beide Funktionen können nicht direkt in SQL-Anweisungen verwendet werden

```
DECLARE
    name EMPLOYEES.LAST_NAME%TYPE;
    v_code NUMBER;
    v_errm VARCHAR2(512);
BEGIN
    SELECT last_name INTO name
        FROM EMPLOYEES   WHERE EMPLOYEE_ID = -1;
EXCEPTION
    WHEN OTHERS THEN
    -- Verwendung in SQL-Anweisungen mit Hilfe von Variablen
    v_code := SQLCODE;
    v_errm := SQLERRM;
    INSERT INTO errors (e_usr, e_date, e_code, e_errm)
        VALUES (USER, SYSDATE, v_code, v_errm);
END;
```

Auswertung mittels DBMS_Utility-Package

```
DECLARE
    name EMPLOYEES.LAST_NAME%TYPE;
BEGIN
    SELECT last_name INTO name
        FROM EMPLOYEES   WHERE EMPLOYEE_ID = -1;
EXCEPTION
    WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error code '|| SQLERRM);
    DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
    DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_BACKTRACE);
END;
```

- DBMS_UTILITY.FORMAT_ERROR_STACK ungekürzte Meldung (kein Stack!!)
  DBMS_UTILITY.FORMAT_ERROR_BACKTRACE Stack mit Zeilennummern
- Beide Funktionen aus dem Package können direkt in SQL-Anweisungen verwendet werden