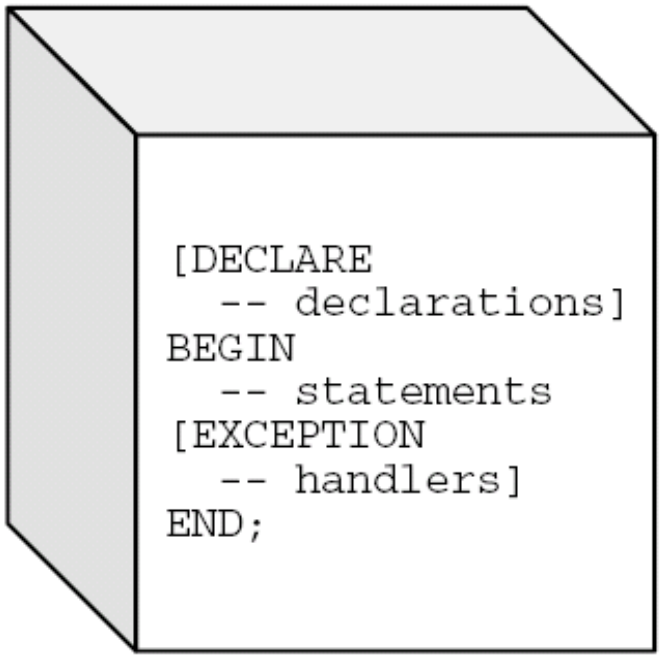
PL/SQL – Datentypen

Stephan Karrer

PL/SQL Blockstruktur:



DECLARE (optional):

- Deklaration und Initialisierung von Variablen und Konstanten
- Typdefinitionen, Ausnahme-Definitionen
- Cursor-Definitionen

EXCEPTION (optional):

Abfangen und Behandlung von

- internen Fehlern (Oracle oder PL/SQL)
- benutzerdefinierten Ausnahmen, können auch zur Signalisierung genutzt werden

Ein erstes Beispiel: Ausgabe von PL/SQL-Blöcken testen

```
/* Ermöglichen der Ausgabe in SQL*Plus */

SET SERVEROUTPUT ON -- SQL*Plus Kommando

DECLARE
f_name VARCHAR2(20);

BEGIN
     SELECT first_name INTO f_name
            FROM employees
            WHERE employee_id=100;

     DBMS_OUTPUT.PUT_LINE ('Firstname is ' || f_name);

EXCEPTION    WHEN OTHERS THEN

     DBMS_OUTPUT.PUT_LINE ('Exception ' || sqlerrm);
END;
/
-- Block-Trenner (SQL*Plus Kommando)
```

Austauschvariablen

```
DECLARE
   sal NUMBER;
BEGIN
  SELECT salary INTO sal FROM employees
    WHERE employee_id = &empid;
    SYS.DBMS_OUTPUT.PUT_LINE(sal);
END;
```

- Dienen der Übergabe von Benutzereingaben an SQL- und PL/SQL-Anweisungen
- Werden in SQL und PL/SQL via &-Zeichen referenziert
- Bei der Ausführung wird der Benutzer zur Eingabe aufgefordert und der Wert als Zeichenkette ersetzt (Makro-Mechanismus)

Austauschvariablen

```
SET VERIFY OFF
DECLARE
   sal NUMBER;
BEGIN
  SELECT salary INTO sal FROM employees
    WHERE employee_id = &&empid;
    DBMS_OUTPUT.PUT_LINE(sal);
    DBMS_OUTPUT.PUT_LINE(&empid);
END;
UNDEFINE empid
```

- VERIFY OFF unterdrückt die Anzeige des alten und neuen Anweisungstexts
- Durch &&-Zeichen wird nur ein Eingabe-Dialog erforderlich und die Variable hat Gültigkeit für die Session
- Mit UNDEFINE kann die Variable gelöscht werden

Syntaxregeln

Innerhalb des Blocks werden die einzelnen Anweisungen mittels Semikolon abgeschlossen (wie bei SQL)

Als Trenner (Whitespace) dienen Leerzeichen, Tabulatoren und Zeilenumbrüche

Schlüsselwörter dürfen nicht getrennt werden

Groß/Kleinschreibung von Bezeichnern und Schlüsselworten ist irrelevant (wie in SQL)

Als Zeichensatz für das Programm steht der Zeichensatz des DBMS zur Verfügung (Zeichenketten als Daten können sehr wohl nationalen Konventionen folgen)

Es existieren Zeilen- und Block-Kommentare

Variablen: Deklaration und Wertzuweisung

```
DECLARE
      n INTEGER; k INTEGER;
      birthday DATE;
      location VARCHAR2(15) := 'Munich';
      emp_count SMALLINT := 0;
      hours_worked INTEGER DEFAULT 40;
      emp_id INTEGER(4) NOT NULL := 9999;
      credit_limit CONSTANT REAL := 5000.00;
      summ_hours INTEGER := emp_count * hours_worked;

BEGIN
      /* ..... */
END;
```

Gültige Bezeichner

Bezeichner, z.B. Variablennamen, beginnen mit einem Buchstaben, gefolgt von Buchstaben, Ziffern, *$*, _ , #

Die maximale Länge von Bezeichnern ist 30 Zeichen

```
-- gültige Bezeichner
     t2
     phone#
     credit_limit
     oracle$number
     money$$$tree

-- ungültige Bezeichner
     mine&yours -- ampersand (&) is not allowed
     debit-amount -- hyphen (-) is not allowed
     on/off -- slash (/) is not allowed
     user id -- space is not allowed
```

Maskierung von Bezeichnern (quoting)

Sollen Bezeichner andere Zeichen enthalten bzw. zwischen Groß/Kleinschreibung unterschieden werden, so können diese in doppelte Anführungszeichen eingeschlossen werden

- Dies dient vor allem der Verwendung von PL/SQL-Schlüsselwörtern in SQL-Anweisungen

```
-- gültige Bezeichner
     "X+Y"
     "last name"
     "on/off switch"
     "employee(s)"
     "*** header info ***"
```

Es gelten die üblichen Verschattungsregeln (Geltungsbereich und Sichtbarkeit)

```
DECLARE
  a CHAR;
  b REAL;
 BEGIN
  -- identifiers available here: a (CHAR), b
  DECLARE
      a INTEGER;
      c REAL;
  BEGIN
      NULL; -- identifiers available here: a (INTEGER), b, c
  END;

  DECLARE
      d REAL;
  BEGIN
      NULL; -- identifiers available here: a (CHAR), b, d
  END;
  -- identifiers available here: a (CHAR), b
END;
```

Durch Block-Label ist der Zugriff auf äußere Variablen möglich

```
<<outer>>
DECLARE
    birthdate DATE := '09-AUG-70';
BEGIN
    DECLARE
        birthdate DATE;
    BEGIN
        birthdate := '29-SEP-70';
        IF birthdate = outer.birthdate THEN
            DBMS_OUTPUT.PUT_LINE ('Same Birthday');
        ELSE
            DBMS_OUTPUT.PUT_LINE ('Different Birthday');
        END IF;
    END;
END;
```

Datentypen

Skalare (PLS_INTEGER, NUMBER, VARCHAR2, DATE, ...)

Zusammengesetzte (TABLE, RECORD, NESTED TABLE, VARRAY)

Große Objekte (BFILE, BLOB, CLOB, NCLOB)

Referenztypen (REF CURSOR, REF)

Zeichenketten als Datentyp

Typ	Speicherplatz
CHAR [(max. length [CHAR|BYTE])] NCHAR [(max. length [CHAR|BYTE])] VARCHAR2 [(max. length [CHAR|BYTE])] NVARCHAR2 [(max. length [CHAR|BYTE])]	bis max. 32.767 Byte (default 1)

Beispiele:

```
text VARCHAR2(15 BYTE) := 'Max Muster'
national_text CHAR(20 CHAR) := 'Jürgen Claß';
letter CHAR := 'A';
event1 VARCHAR2(15) := 'Mother''s Day';
event2 VARCHAR2(15) := q'!Mother's Day!';
```

Operationen auf Zeichenketten: Konkatenation

Operator	Beschreibung	Beispiel
`||`	Konkatenation von Zeichenketten und CLOB-Daten	`SELECT 'Name is ' || last_name FROM employees;`

```
DECLARE
  x VARCHAR2(4) := 'suit';
  y VARCHAR2(4) := 'case';

BEGIN
  DBMS_OUTPUT.PUT_LINE (x || y);
END;
```

SQL-Funktionen: Zeichenketten (Auszug)

Funktion	Beschreibung
LOWER(column)	Konvertiert Zeichenkette in Kleinbuchstaben
UPPER(column)	Konvertiert Zeichenkette in Großbuchstaben
INITCAP(column)	Konvertiert den ersten Buchstaben einer Zeichenkette in einen Großbuchstaben
CONCAT(STR1, STR2)	Verbindet zwei Strings
||	Verbindet n Strings
SUBSTR(column,start,length)	Extrahiert eine Zeichenkette der angegebenen Länge
LENGTH(column)	Gibt die Länge einer Zeichenkette wieder
INSTR(column,string,n))	Gibt die Position eines Zeichens in einem String an
LPAD(column,length)	Füllt den Zeichenwert so mit Leerzeichen auf, dass ein rechtsbündiger Blocksatz entsteht

Numerische Datentypen: Ganzzahlen

Typ	Wertebereich	Subtypen
BINARY_INTEGER *bzw.* PLS_INTEGER	-2.147.483.648 . ... +2.147.483.647	NATURAL NATURALN POSITIVE POSITIVEN SIGNTYPE SIMPLE_INTEGER

Beispiele:

```
k PLS_INTEGER;

....

k := 030;      k := 6;     k := -14;     k := +32767;
```

Numerische Datentypen: Gleitpunktzahl im Oracle-Format

Typ	Wertebereich	Subtypen
NUMBER[(precision,scale)]	+/- 1E-130 . ... +/- 1.0E126 precision: 1 ... 38 scale: -84 ... 127	DEC, DECIMAL, NUMERIC INTEGER, INT, SMALLINT REAL FLOAT, DOUBLE PRECISION

Beispiele:

```
k NUMBER; l NUMBER(6,2);

....

k := 030; k := +32767.78; k := 1.3E-7;  l :=455.12;
```

Arithmetische Operatoren:

```
DECLARE
  i1 PLS_INTEGER;
  i2 PLS_INTEGER;
  i3 PLS_INTEGER;
BEGIN
   i1 := 2;   i2 := 4;
   DBMS_OUTPUT.PUT_LINE( i1 + i2); -- Addition
   DBMS_OUTPUT.PUT_LINE( i2 - i1); -- Subtraktion
   DBMS_OUTPUT.PUT_LINE( i1 / i2 ); -- Division
   DBMS_OUTPUT.PUT_LINE( i2 * i3);  -- Multiplikation
   DBMS_OUTPUT.PUT_LINE( i1 ** i2 ); -- Potenz
END;
```

Numerische Datentypen: Gleitpunktzahlen nach IEEE 754

Typ	Wertebereich	Subtypen
BINARY_FLOAT *bzw.* BINARY_DOUBLE	1.17549E-38F ... 3.40282E+38F 2.22507485850720E-308 … 1.79769313486231E+308	SIMPLE_FLOAT SIMPLE_DOUBLE

Beispiele:

```
k BINARY_FLOAT;  l BINARY_DOUBLE;

....

k := 2.07f;  l := 3.00094d;  k := 1.3E-7;  l := -9.999e125;
```

SQL-Funktionen: Numerik (Auszug)

Funktion	Beschreibung
ABS(zahl)	Absolutbetrag einer Zahl
CEIL(zahl)	Nächstgrößere Ganzzahl
ROUND(zahl, n)	Rundung auf n Stellen
TRUNC(zahl, n)	Abschneiden von Stellen
REMAINDER(zahl1, zahl2) MOD(zahl1, zahl2)	Rest der ganzzahligen Division
NANVL(zahl1, zahl2)	Liefert Zahl2, wenn Zahl1 keine gültige Zahl
POWER(zahl1, zahl2)	Potenz
EXP(zahl)	Natürliche Exponentialfunktion
LOG(zahl1, zahl2)	Logarithmus zur Basis Zahl1
SQRT(zahl)	Quadratwurzel
SIN(zahl), COS(zahl), …	Trigonometrische Funktionen

Datentypen: Datum und verschiedene Zeitstempel

DATE	Speichert Jahrhundert, Jahr, Monat, Tag, Stunde, Minute und Sekunde.
TIMESTAMP [(fractional_seconds_precision)]	Der Datentyp TIMESTAMP speichert zusätzlich Sekundenbruchteile, ***`fractional_seconds_precision`*** spezifiziert optional die Genauigkeit (0 bis 9 Stellen). Der Standardwert ist 6.

TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE

TIMESTAMP [(fractional_seconds_precision)] WITH LOCAL TIME ZONE

INTERVAL YEAR [(year_precision)] TO MONTH

INTERVAL DAY [(day_precision)] TO SECOND [(fractional_seconds_precision)]

SQL-Funktionen: Zeitstempel (Auszug)

Funktion	Beschreibung
SYSDATE	Liefert aktuelles Systemdatum
MONTHS_BETWEEN(datecolumn1, datecolumn2)	Zahl der Monate zwischen zwei Datumsangaben
ADD_MONTHS(datecolumn,n)	Kalendermonate zu einem Datum hinzufügen
NEXT_DAY(datecolumn, next day) NEXT_DAY('15-MAR-98','TUESDAY')	Der Tag, der auf den angegebenen folgt
LAST_DAY(datecolumn)	Letzter Tag des Monats
ROUND(date)	Gerundetes Datum
TRUNC(date)	Abgeschnittenes Datum
EXTRACT(feld)	Extraktion des Zeit- bzw. Datumfelds

SQL-Funktionen: Konvertierung von Datentypen

Von	In	Mit
VARCHAR/CHAR	NUMBER	TO_NUMBER
VARCHAR/CHAR	DATE	TO_DATE
NUMBER	VARCHAR2	TO_CHAR
DATE	VARCHAR2	TO_CHAR

Beispiele:

```
SELECT TO_CHAR(hiredate, 'DD.MM.YYYY') FROM emp;
```

Ergibt z.B. "12.01.1983"

Wahrheitswerte

Typ	Bemerkungen
BOOLEAN	ist eigener Datentyp nur in PL/SQL SQL kennt zwar logische Ausdrücke, aber keine Wahrheitswerte als Spalteninhalt, d.h. Variablen können nicht in SQL-Anweisungen verwendet werden

Beispiele:

```
flag BOOLEAN NOT NULL := TRUE;

not_defined BOOLEAN := NULL;

valid BOOLEAN;  valid := (empno IS NOT NULL);
```

Logik-Operatoren

Logische Verküpfungsoperatoren können auf logische Ausdrücke und Werte angewendet werden.

Operator	Kommentar
AND, OR, NOT	Basisoperatoren

```
DECLARE
  b1 BOOLEAN;   b2 BOOLEAN;  b3 BOOLEAN;
BEGIN
  b1:=TRUE;  b2:=FALSE;
  b3:= b1 AND b2;
  b3:= b1 OR b2;
  b3:= NOT b1;
END;
```

Vergleichsoperatoren

=	gleich
<>, !=, ~=, ^=	ungleich
>	größer
>=	größer oder gleich
<	kleiner
<=	kleiner oder gleich

Spezielle Vergleichsoperatoren

Operator	Kommentar
BETWEEN	Prüft, ob der Operand im Intervall liegt
IN	Prüft, ob der Operand in der Aufzählung enthalten ist
LIKE	Prüft, ob der Operand einem Muster gleicht

```
DECLARE
  b BOOLEAN;
  letter VARCHAR2(1) := 'm';
  pattern VARCHAR2(20) := 'J%s_n';
BEGIN
  b := letter IN ('a', 'b', 'c');
  b := letter IN ('z', 'm', 'y', 'p');
  b := 2 BETWEEN 1 AND 3;
  b := 2 NOT BETWEEN 3 AND 4;
  b := 'Johnson' LIKE pattern;
END;
```

Nullwerte (Null Values)

- Nullwerte stehen für nicht verfügbare bzw. unbekannte Werte
- Werte können explizit auf NULL gesetzt bzw. daraufhin überprüft werden
- Ist ein Operand in arithmetischen Ausdrücken ein Nullwert, so ergibt die Auswertung stets NULL.
- Vergleiche mit Nullwerten liefern stets NULL (außer die speziellen Tests auf Nullwerte)
- Bei der Konkatenation von Zeichenketten wird ein Nullwert ignoriert (d.h. wie eine leere Zeichenkette behandelt)
- Bei der Auswertung logischer Ausdrücke wird durch Nullwerte die Prädikatenlogik erweitert.

Logik der Wahrheitswerte

x	y	x AND y	x OR y	NOT x
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
TRUE	NULL	NULL	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	NULL	FALSE	NULL	TRUE
NULL	TRUE	NULL	TRUE	NULL
NULL	FALSE	FALSE	NULL	NULL
NULL	NULL	NULL	NULL	NULL

Prüfung auf NULL-Wert

Operator	Kommentar
IS NULL	Prüft, ob der Operand ein NULL-Wert ist
IS NOT NULL	Prüft, ob der Operand kein NULL-Wert ist

```
DECLARE
  b BOOLEAN;
BEGIN
  if (b IS NOT NULL )
    THEN DBMS_OUTPUT.PUT_LINE( 'b ist not null' );
    ELSE DBMS_OUTPUT.PUT_LINE( 'b ist null' );
    END IF;
END;
```

SQL Funktionen: Nullwerte

NVL (expr1, expr2)

NVL2 (expr1, expr2, expr3)

NULLIF (expr1, expr2)

COALESCE (expr1, expr2, ... , exprN)

Verwendung von NVL und NVL2

```
SELECT last_name, NVL(TO_CHAR(commission_pct),'Not Applicable')
                  "COMMISSION" FROM employees
  WHERE last_name LIKE 'B%'
  ORDER BY last_name;
```

```
SELECT last_name, salary,
       NVL2(commission_pct, salary + (salary * commission_pct),
            salary) income
FROM employees WHERE last_name like 'B%'
ORDER BY last_name;
```

Verwendung von NVL und NVL2

```
SELECT last_name, NVL(TO_CHAR(commission_pct),'Not Applicable')
                  "COMMISSION" FROM employees
  WHERE last_name LIKE 'B%'
  ORDER BY last_name;
```

```
SELECT last_name, salary,
       NVL2(commission_pct, salary + (salary * commission_pct),
            salary) income
FROM employees WHERE last_name like 'B%'
ORDER BY last_name;
```

Verwendung von NULLIF und COALESCE

```
SELECT e.last_name, NULLIF(e.job_id, j.job_id) "Old Job ID"
   FROM employees e, job_history j
   WHERE e.employee_id = j.employee_id
   ORDER BY last_name;
```

```
SELECT product_id, list_price, min_price,
       COALESCE(0.9*list_price, min_price, 5) "Sale"
   FROM product_information
   WHERE supplier_id = 102050;
```