

## 2 Grundlagen von PL/SQL

Grundelemente von PL/SQL.

### 2.1 Übersicht

Themen des Kapitels – *Grundlagen von PL/SQL*

## Themen des Kapitels

- PL/SQL Blöcke
- Kommentare
- Bezeichner
- Variablen
- Operatoren

Im Kapitel *Grundlagen von PL/SQL* werden die grundlegenden Syntax-Elemente der Sprache PL/SQL und ihre Anwendung erläutert.

## 2.2 PL/SQL Blöcke

Struktur eines PL/SQL Blocks.

### PL/SQL Blöcke

```
DECLARE  
/* Deklaration der Variablen */  
  
BEGIN  
/* Start des Programmcodes */  
  
EXCEPTION  
/* Behandlung von Unterbrechungen */  
  
END;
```

## 2.3 Kommentare

Kommentare unter PL/SQL.

# Kommentare

*/\* Mehrzeiliger  
Kommentar \*/*

*-- Einzeiliger Kommentar*

## 2.4 Bezeichner

Bezeichner und ihre Verwendung in PL/SQL.

# Bezeichner

- *Bezeichner1*
- *“Bezeichner2“*

Wird ein Bezeichner in Anführungszeichen gesetzt, erfolgt eine Unterscheidung zwischen Groß-/Kleinschreibung und Leerzeichen werden berücksichtigt!

## 2.5 Variablen

Deklaration und Verwendung von Variablen.

### 2.5.1 Datentypen

## Datentypen

- *CHAR*
- *NCHAR*
- *VARCHAR2*
- *NVARCHAR2*
- *NUMBER(p,s)*
- *DECIMAL*
- *FLOAT*
- *REAL*
- *INTEGER*
- *DATE*

- *LONG*
- *BLOB*
- *CLOB*
- *NCLOB*
- *BFILE*
- *RAW*
- *LONGRAW*
- *ROWID*
- *UROWID*

Datentyp	Code	Beschreibung
<i>CHAR (Größe)</i>	96	Zeichendaten fester Länge (maximal 2000 Bytes)
<i>NCHAR (Größe)</i>	96	Zeichendaten fester Länge abhängig vom nationalen Zeichensatz (maximal 2000 Bytes)
<i>VARCHAR2 (Größe)</i>	1	Zeichendaten variabler Länge (maximal 4000 Bytes)

Datentyp	Code	Beschreibung
<i>NVARCHAR2</i> (Größe)	1	Zeichendaten variabler Länge abhängig vom nationalen Zeichensatz (maximal 4000 Bytes)
<i>BLOB</i>	113	Binary Large Object (maximal 4 GByte)
<i>CLOB</i>	112	Character Large Object (maximal 4 GByte)
<i>NCLOB</i>	112	Character Large Object abhängig vom nationalen Zeichensatz (maximal 4 GByte)
<i>BFILE</i>	114	Zeiger auf ein Large Binary File das außerhalb der Datenbank gespeichert wird (maximal 4 GByte)
<i>NUMBER</i> (p, s)	2	Precision $1 \leq p \leq 38$ Scale $-84 \leq s \leq 127$ (Bei negativen Werten für Scale wird bis zur entsprechenden Vorkommastelle gerundet)
<i>DATE</i>	12	Datum und Uhrzeit (01. Januar 4712 v. Chr. bis 31. Dezember 9999)
<i>RAW</i> (Größe)	23	Binäre Daten (maximal 2000 Byte)
<i>LONG RAW</i>	24	Binäre Daten variabler Länge (maximal 2 GByte)
<i>ROWID</i>	69	Eindeutige Adresse einer Zeile in der Datenbank.
<i>UROWID</i> (Größe)	208	Logische Adresse einer Zeile in einer Indexed Organized Table (maximal 4000 Byte)

## 2.5.2 Variablendeklaration

# Variablendeklaration

```
DECLARE  
    v_Var01  VARCHAR2(20);  
    v_Var02  VARCHAR2(20) := 'Vorbelegung';  
    v_Var03  NUMBER NOT NULL := 0;  
    v_Var04  CONSTANT NUMBER := 1000;  
BEGIN  
    ...  
END;
```

Ein deklarierte Variable ohne Default-Wert wird standardmäßig mit einem *NULL*-Wert vorbelegt.

## Zuweisung eines Wertes mit *SELECT*

```
SELECT ename INTO v_var01 FROM emp  
      WHERE ename = 'WARD';
```

Die Zuweisung des Wertes einer Spalte beim selektieren erfolgt mit dem Schlüsselwort *INTO*. Nach *INTO* muss eine Variable angegeben werden die vom Datentyp mit der Spalte in der Tabelle übereinstimmt (oder entsprechend Größer deklariert ist).

### 2.5.3 Spezielle Variablen

## Spezielle Variablen

```
DECLARE  
    v_Var01 tabelle.spalte%TYPE;
```

Der Datentyp einer Variablen kann vom Datentyp einer Spalte in einer Tabelle übernommen werden. Die Übernahme geschieht z.B. durch *EMP.ENAME%TYPE* d.h. es wird der Datentyp der Spalte *ENAME* aus der Tabelle *EMP* übernommen.

## 2.5.4 Subtypen

# Subtypen

```
DECLARE  
    v_Var01  NUMBER(4);  
    SUBTYPE zaehler IS v_Var01%TYPE;  
    v_zaehler zaehler;
```

Mit der Anweisung *SUBTYPE* kann ein benutzerdefinierter Datentyp erstellt werden der zur Deklaration anderer Variablen verwendet werden kann.

## 2.5.5 Umwandlungsfunktionen

# Umwandlungsfunktionen

- *TO\_CHAR*
- *TO\_DATE*
- *TO\_NUMBER*
- *RAWTOHEX*
- *TORAW*
- *CHARTOROWID*
- *ROWIDTOCHAR*

PL/SQL führt soweit mögliche eine implizite Umwandlung durch.

Mittels spezieller Funktionen, so genannten Umwandlungsfunktionen kann auch eine explizite Umwandlung erfolgen.

Funktion	Beschreibung	Konvertierbare Familie
<i>TO_CHAR</i>	Wandelt das Argument in eine Variable vom Datentyp <i>VARCHAR2</i> , abhängig von der optionalen Formatspezifikation um.	Numerisch, Datum
<i>TO_DATE</i>	Wandelt das Argument in eine Variable vom Datentyp <i>DATE</i> , abhängig von der optionalen Formatspezifikation um.	Zeichen

Funktion	Beschreibung	Konvertierbare Familie
<i>TO_NUMBER</i>	Wandelt das Argument in eine Variable vom Datentyp <i>NUMBER</i> , abhängig von der optionalen Formatspezifikation um.	Zeichen
<i>RAWTOHEX</i>	Wandelt einen <i>RAW</i> Wert in die Hexadezimaldarstellung einer binären Größe um.	Binärdaten
<i>HEXTORAW</i>	Wandelt einen Wert in Hexadezimaler Darstellung in die entsprechende binäre Größe um.	Zeichen (in hexadezimaler Darstellung)
<i>CHARTOROWID</i>	Wandelt ein Zeichen einer <i>ROWID</i> in das interne binäre Format um.	Zeichen (im <i>ROWID</i> Format)
<i>ROWIDTOCHAR</i>	Wandelt eine interne <i>ROWID</i> in das externe Format um.	<i>ROWID</i>

## 2.6 Operatoren

Einsatz und Funktion von Operatoren in PL/SQL.

### 2.6.1 Übersicht der Operatoren

## Übersicht - Operatoren

- Arithmetische Operatoren
- Operatoren zur Verbindung von Zeichenketten
- Vergleichs Operatoren
- Logische Operatoren

## 2.7 Arithmetische Operatoren

Typen von arithmetischen Operatoren.

# Arithmetische Operatoren

- + (Addition/Positiv)
- - (Subtraktion/Negativ)
- \* (Multiplikation)
- / (Division)

Operator	Funktion	Beispiel
+ -	Positiv oder Negativ  Addieren oder Subtrahieren	<i>SELECT * FROM orders WHERE qtysold = -1;</i>  <i>SELECT sal + comm FROM emp WHERE SYSDATE - hiredate &gt; 365;</i>
* /	Multiplikation oder Division	<i>UPDATE emp SET sal = sal * 1.1;</i>

## 2.8 Operatoren zur Verbindung von Zeichenketten

Verbinden von Zeichenketten mit speziellen Operatoren.

# Verbindung von Zeichenketten

- // (Concatenation)

Operator	Funktion	Beispiel
//	Verbindung von Zeichenketten	<i>SELECT 'Name ist '    ename FROM emp;</i>

## 2.9 Vergleichs Operatoren

Vergleichen von Werten mit speziellen Operatoren.

### Vergleichs Operatoren

<ul style="list-style-type: none"> <li>■ =</li> <li>■ &lt;&gt; != ^=</li> <li>■ &lt; &gt; &lt;= &gt;=</li> <li>■ <i>IN / NOT IN</i></li> <li>■ <i>ANY / SOME</i></li> <li>■ <i>ALL</i></li> <li>■ <i>[NOT] BETWEEN x AND y</i></li> <li>■ <i>EXISTS</i></li> <li>■ <i>[NOT] LIKE</i></li> </ul>	<ul style="list-style-type: none"> <li>■ <i>IS [NOT] NULL</i></li> </ul>
---	--

Vergleichs Operatoren vergleichen Werte und liefern als Ergebnis entweder *TRUE*, *FALSE* oder *UNKNOWN*.

Operator	Funktion	Beispiel
=	Test auf Gleichheit	<i>SELECT * FROM emp WHERE sal = 1500;</i>
!= ^= < >	Test auf Ungleichheit (plattformabhängig)	<i>SELECT * FROM emp WHERE sal != 1500;</i>

Operator	Funktion	Beispiel
>	Größer	
<	Kleiner	
>=	Größer gleich	
<=	Kleiner gleich	
<i>IN</i>	In einer Menge	<i>SELECT * FROM emp WHERE job IN ( 'CLERK', 'ANALYST' );</i>
<i>NOT IN</i>	Nicht in einer Menge	
<i>ANY</i> <i>SOME</i>	Vergleicht einen Wert mit allen Werten, die von einer Abfrage zurückgegeben werden.	<i>SELECT * FROM emp WHERE sal = ANY ( SELECT sal FROM emp WHERE deptno = 30 );</i>
<i>ALL</i>	Vergleicht einen Wert mit allen Werten die in einer Liste angegeben werden.	<i>SELECT * FROM emp WHERE sal &gt;= ALL ( 1400, 3000 );</i>
<i>[NOT]</i> <i>BETWEEN x</i> <i>AND y</i>	[Nicht] größer oder gleich x und kleiner oder gleich y.	<i>SELECT * FROM emp WHERE sal BETWEEN 2000 AND 3000;</i>

Operator	Funktion	Beispiel
<i>EXISTS</i>	TRUE wenn die Unterabfrage mindestens eine Zeile liefert.	<i>SELECT ename, deptno FROM dept WHERE EXISTS ( SELECT * FROM emp WHERE dept.deptno = emp.deptno);</i>
<i>x [NOT] LIKE y  [ESCAPE 'z']</i>	TRUE wenn x [nicht] dem Muster y entspricht.	<i>SELECT * FROM tab01 WHERE col01 LIKE 'A_C/%E%' ESCAPE '/';</i>
<i>IS [NOT] NULL</i>	Test auf NULL Wert.	<i>SELECT ename, deptno FROM emp WHERE comm. IS NULL;</i>

## 2.10 Logische Operatoren

Kombinieren von Bedingungen mit logischen Operatoren.

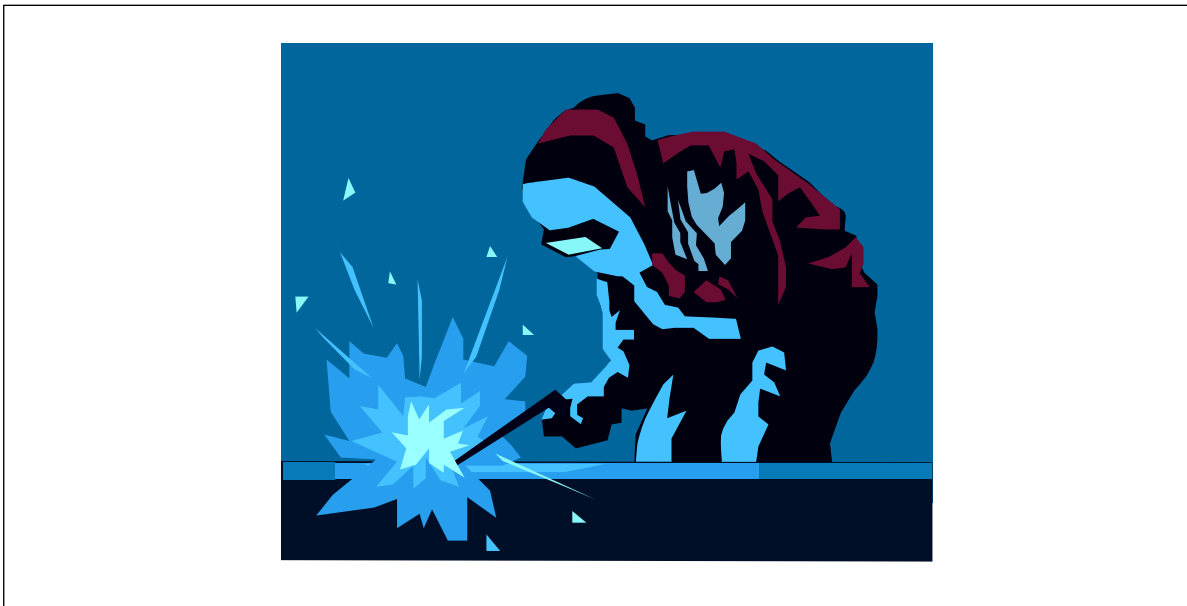
### Logische Operatoren

- *NOT*
- *AND*
- *OR*

Operator	Funktion	Beispiel
<i>NOT</i>	Liefert TRUE wenn die Bedingung nicht erfüllt ist.	<i>SELECT * FROM emp WHERE NOT (job IS NULL);</i>
<i>AND</i>	Liefert TRUE wenn beide Bedingungen erfüllt sind.	<i>SELECT * FROM emp WHERE job = 'CLERK' AND deptno = 10;</i>
<i>OR</i>	Liefert TRUE wenn eine oder beide Bedingungen erfüllt sind.	<i>SELECT * FROM emp WHERE job = 'CLERK' OR deptno = 10;</i>

## 2.11 Übung – PL/SQL Grundlagen

# Übung – PL/SQL Grundlagen



Vor der Ausführung der Übungen müssen die Beispieltabellen angelegt werden. Die Datenbankobjekte werden über das Script *store\_schema.sql* angelegt.

1. In der Übung soll eine PL/SQL Prozedur erstellt werden in der zwei Strings verbunden werden. Die Werte sollen in zwei Variablen abgelegt werden und mittels einer dritten Variable verbunden werden. Diese dritte Variable soll anschließen mit dem *DBMS\_OUTPUT* Paket ausgegeben werden.

Unter PL/SQL muss dazu vor der Ausführung der Wert:

```
set serveroutput on size 1000000
```

gesetzt sein.

Die Ausgabe erfolgt dann mit:

```
DBMS_OUTPUT.PUT_LINE (...)
```

2. Als nächstes soll eine PL/SQL Prozedur erstellt werden, die mittels eines *SELECT* Befehls den Namen des Angestellten mit dem höchsten *SALARY* ermittelt (aus der Tabelle *MORE\_EMPLOYEES*).

Der Wert soll anschließend mittels eines *DBMS\_OUTPUT* Aufrufs ausgegeben werden.

3. In dieser Übung muss eine PL/SQL Prozedur erstellt werden, die einen Datensatz in die Tabelle *MORE\_EMPLOYEES* einfügt.

Dieser Datensatz (*LAST\_NAME* und *SALARY*) soll anschließend mittels eines *DBMS\_OUTPUT* Aufrufs ausgegeben werden.

4. In der nächsten Übung soll der Kunde (Tabelle *CUSTOMERS*) mit dem höchsten Umsatz ermittelt werden. Die Ausgabe mit *DBMS\_OUTPUT* soll den Nachnamen, den Vornamen und die Telefonnummer umfassen. Zur Lösung werden die Tabellen *CUSTOMERS*, *PURCHASES* und *PRODUCTS* benötigt.

**Notizen:**