# 1

# DECLARE PL/SQL VARIABLES

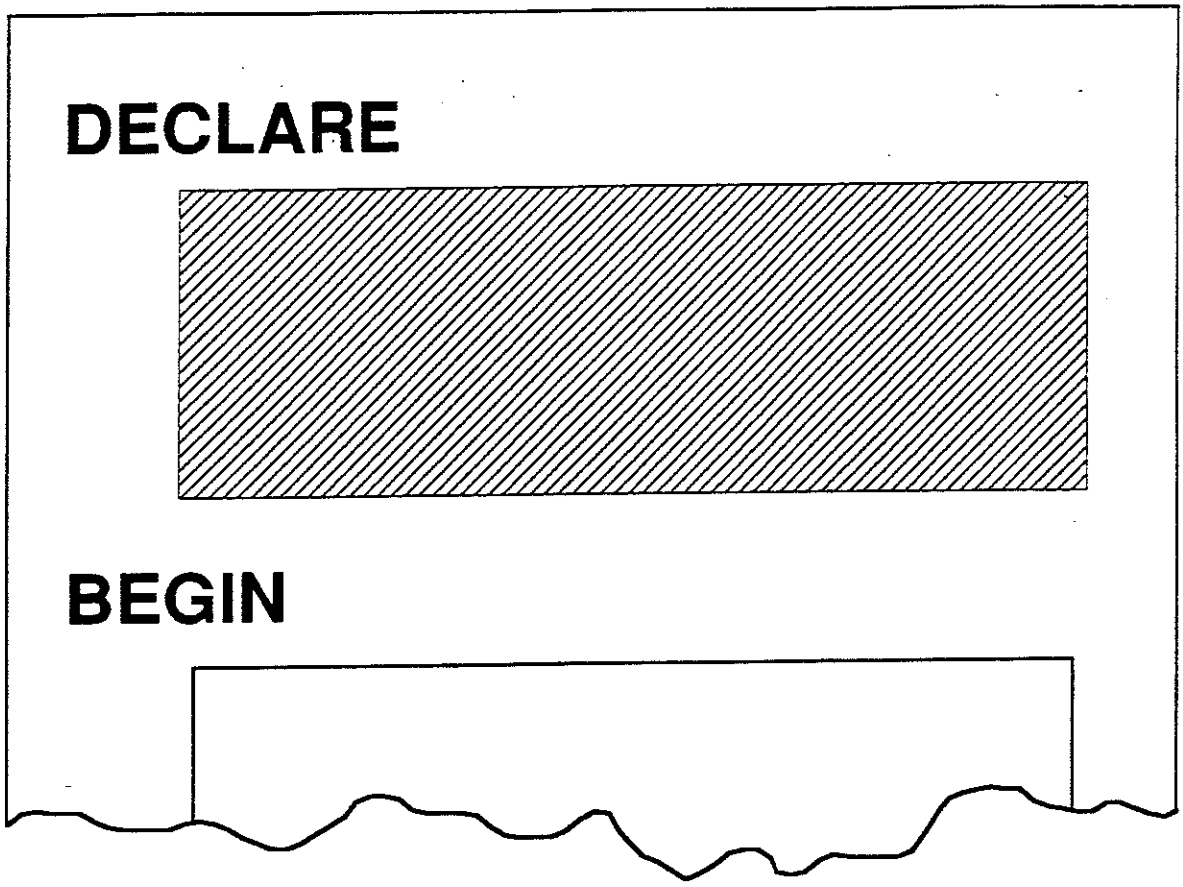
# SECTION OBJECTIVES

**At the end of this section, you should be able to:**

1. Identify legal and illegal PL/SQL declarations.
2. Describe PL/SQL datatypes and conversion rules.
3. Apply an understanding of variable and constant scoping concepts to derive values from a sample PL/SQL block.

# DECLARE PL/SQL VARIABLES: OVERVIEW

**Declare constants and variables in the declaration section.**



Binary Integers  -2GB → +2GB
POSITIVE  1 → +2GB
NATURAL  0 → +2GB

CHAR  fast lengde inntil 32kB
ROWID (fysisk adresse till en rad)  6 BYTE
block.rad.fil.

NUMBER  inntil 40 tegn m/ 38 tegn presisjon.
LONG  Tekst inntil 32kB
VARCHAR eller VARCHAR2 - Dynamisk lengde tekst inntil 32kB.
BOOLEAN - true / false / null
DATE - alltid 7 byte, inneh. dato/tidsstempel
RAW - Binær data 32kB lyd/bilde
LONG RAW - — " — oversettes til
internt lagringsformat.

---

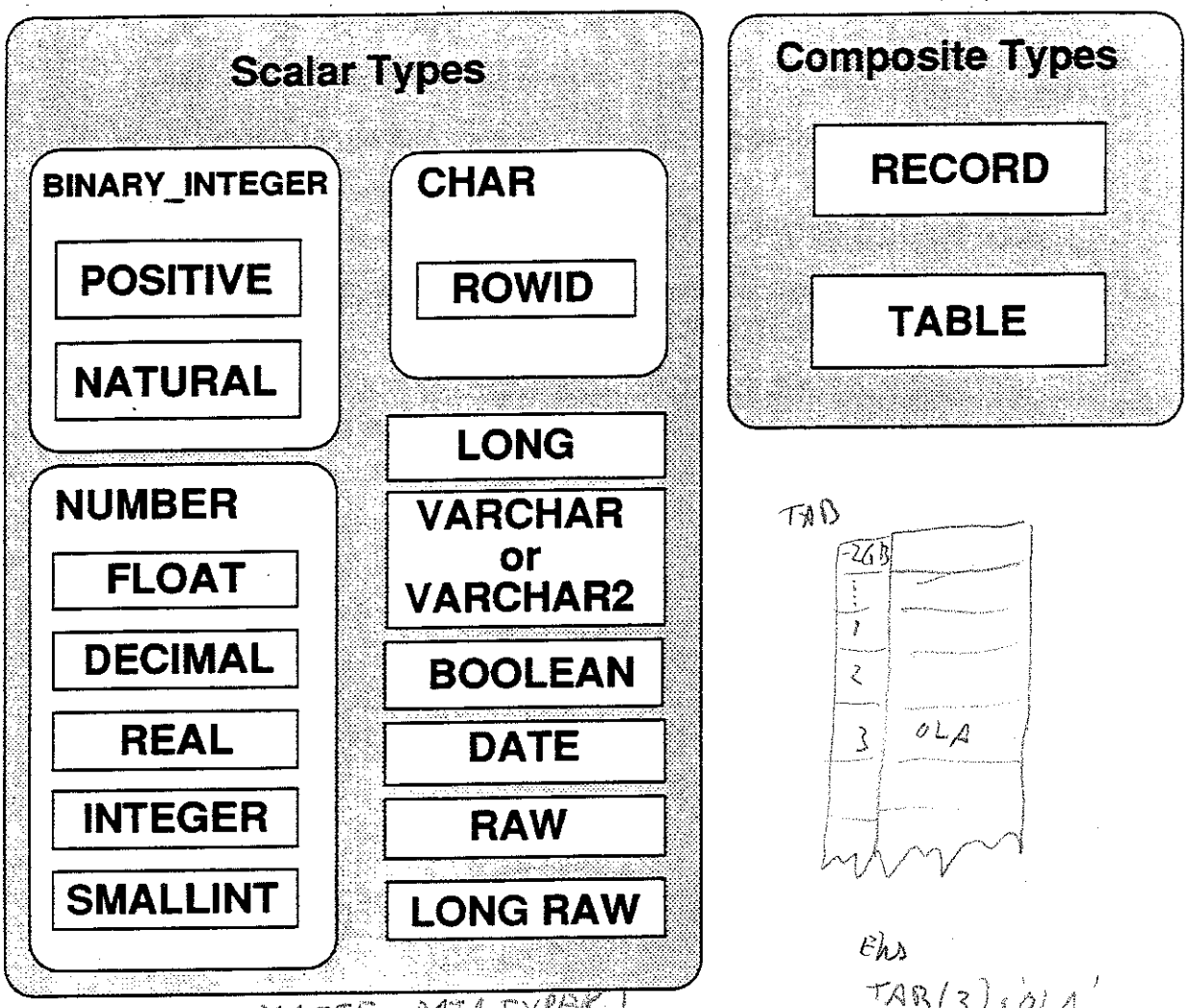
eksempel

```
declare v_rowid rowid;
        v_ename varchar2(20);
begin select rowid, ename into
        v_rowid, v_ename
      from emp
        where empno = 7369;
end;
/
```

# PL/SQL DATATYPES

**Declare variables as scalar or composite datatypes.**

*PL/SQL Datatypes*




TAB
-2GB
1
2
3 OLA

Eks
TAB(3) = 'OLA'
↑
index komponent

TABLE (SAMMENSATTE DATA TYPER)
- Benyttes for å unngå nøttverdstomplikk
- Et endimensjonelt array

RECORD
- Samler data i minne om en forekomst
- Endimensjonal array, kompleks.
- Inneholder entydige felt, kan ha ulike datatyper

DEPT

| DEPT NO | ENAME | LOCATION |
|---|---|---|
| NUMBER | VARCHAR2(20) | CHAR(Y) |

## PL/SQL Datatypes—cont'd

**Declare variables that have no internal components with scalar datatypes.**

### *Scalar Datatypes*

- Use the CHAR datatype to store fixed-length character data.
- Use the VARCHAR2 datatype to store variable length character data.
- Use the NUMBER datatype for fixed or floating point numbers of virtually any size.
- Use the BINARY_INTEGER datatype for integers.
- The BOOLEAN datatype has no corresponding datatype in the database. It can take values of TRUE, FALSE, and NULL.
- The NUMBER, CHAR, VARCHAR2, and DATE datatypes behave in the same manner as their corresponding datatypes in the database.

## PL/SQL Datatypes—cont'd

**Declare variables that have internal components with composite datatypes. The internal components of a composite datatype can be manipulated individually.**

### *Composite Datatypes*

- Table
- Record

### *Table Datatype*

- A PL/SQL datatype that is similar to but not the same as a database table
- Must contain only one column
- Column datatype of any scalar type
- Simulates a one-dimensional array of unlimited size
- Table elements indexed with a BINARY_INTEGER column
- Index component is called the **primary key** of the table

### *Record Datatype*

- Contains uniquely defined fields which can belong to different datatypes
- Allows for treating dissimilar fields that are logically related as a single unit

# DECLARE SCALAR VARIABLES

## Syntax

```
identifier [CONSTANT]  datatype [NOT NULL]  [:= plsql_expression];
```

## Examples

```
DECLARE
firstname, lastname CHAR(20); -- illegal
```

```
DECLARE
firstname      CHAR(20); -- legal
lastname       CHAR(20); -- legal
```

## Quick Notes

- The rules for identifier names are the same as for SQL objects.
- NOT NULL may be used to constrain a variable so that it cannot have the null value. Variables declared as NOT NULL must be initialized.
- Only one identifier per line is allowed.
- Forward referencing is not allowed.
- By default, variables are initialized to NULL.

For further information on the subject see:

*PL/SQL User's Guide and Reference Version 2.0*, 800-20

Ex

```
Declare
  v_t constant number := 5;
  v_t2 constant number := v_t + 7;
```

ok

```
Declare
  v_t2 constant number := v_t + 7;
  v_t constant number := 5;
```

forwarding fail.

# Declare Scalar Variables—cont'd

## Examples

### NUMBER/BINARY_INTEGER

```
cnt              BINARY_INTEGER;
revenue          NUMBER(9,2);
seconds_per_day  CONSTANT NUMBER := 60 * 60 * 24;
running_total    BINARY_INTEGER(10) := 0;
```

### CHAR/VARCHAR2

```
mid_initial      CHAR;
last_name        VARCHAR2(10) NOT NULL := 'PEEBLES';
company_name     CONSTANT VARCHAR2(12) := 'ORACLE';
```

### DATE

```
anniversary            DATE := '05-JUL-89';
project_completion     DATE;
next_checkup           DATE NOT NULL := '28-DEC-92';
```

### BOOLEAN

```
over_budget      BOOLEAN  NOT NULL := FALSE;
available        BOOLEAN  := NULL;
```

# DECLARE COMPOSITE VARIABLES

## *Steps to Declare TABLEs*

1 Declare a TABLE type.

2 Declare a PL/SQL TABLE of that type.

## Syntax

```
TYPE  type_name  IS TABLE OF
{ column_type  |  table.column%TYPE }  [ NOT NULL]
INDEX BY BINARY_INTEGER;
identifier type_name;
```

## Example

```
DECLARE
TYPE ename_tab_type IS TABLE OF VARCHAR2(10)
     INDEX BY BINARY_INTEGER;
ename_tab  ename_tab_type;
```

## Quick Note

- *ename_tab* represents an entire PL/SQL TABLE.

# Tables of records

In release 2.3 of PL/SQL (release 7.3 of the Oracle7 Server), we allow for PL/SQL "tables of records". Up to release 2.3, it was possible to create record types, but we only allowed for PL/SQL tables of scalars. In fact, in release 2.3 of PL/SQL we have three new features in the PL/SQL "table" area:

- The already mentioned PL/SQL tables of records
- New operations on PL/SQL tables for retrieving information on the "population".
- Better Memory management.

Here is an example of the usage of PL/SQL table of records:

```
CREATE OR REPLACE PACKAGE emp_package IS
  TYPE emp_table_type IS TABLE OF EMP%ROWTYPE INDEX BY BINARY_INTEGER;
  PROCEDURE read_emp_table(emp_table OUT emp_table_type);
END emp_package;

CREATE OR REPLACE PACKAGE BODY emp_package IS
  PROCEDURE read_emp_table(emp_table OUT emp_table_type) IS
    i       BINARY_INTEGER:=0;
  BEGIN
    FOR emp_record IN (SELECT * FROM EMP) LOOP
        emp_table(i):=emp_record;
     i:=i+1;
    END LOOP;
  END;
END emp_package;

DECLARE
  emp_table emp_package.emp_table_type;
BEGIN
  emp_package.read_emp_table(emp_table);
  dbms_output.put_line('An example:'||emp_table(4).ename);
END;
/
```

Note that in this example, the anchored datatype EMP%ROWTYPE is used to define the element types of the PL/SQL table type. We also could have defined a record type first, and then use this record type in the PL/SQL table type.

So something like:

```
        TYPE emp_record IS RECORD (ename emp.ename%TYPE,
                                   job   emp.job%TYPE);
    TYPE emp_table IS TABLE OF emp_record INDEX BY BINARY_INTEGER;
```

Besides this, you can now also use operations on PL/SQL tables to get information about the PL/SQL table. The Operations are:

count - to get the number of entries in the PL/SQL table
last - to get the last index from the PL/SQL table
first - to get the first index from the PL/SQL table
next(i) - to get the index after i from the PL/SQL table
prior(i) - to get the index before i from the PL/SQL table
delete - to delete entries from the PL/SQL table
exists(i) - to check whether entry i exists.

These operations must be used in a method-like syntax. Here is an example:

```
DECLARE
 emp_table emp_package.emp_table_type;
BEGIN
 emp_package.read_emp_table(emp_table);
 LOOP
    dbms_output.put_line(ename_table(i).ename);
    EXIT WHEN i=ename_table.last;
       i:=ename_table.next(i);
 END LOOP;
END;
/
```

# Declare Composite Variables—cont'd

## *Steps to Declare PL/SQL TABLEs*

1 Declare a RECORD type.

2 Declare a user-defined record of that type.

## Syntax

```
TYPE  type_name  IS RECORD
(field_name1   { field_type | table.column%TYPE }   [NOT NULL],
field_name2    { field_type | table.column%TYPE }   [NOT NULL],
...);
identifier type_name;
```

## Example

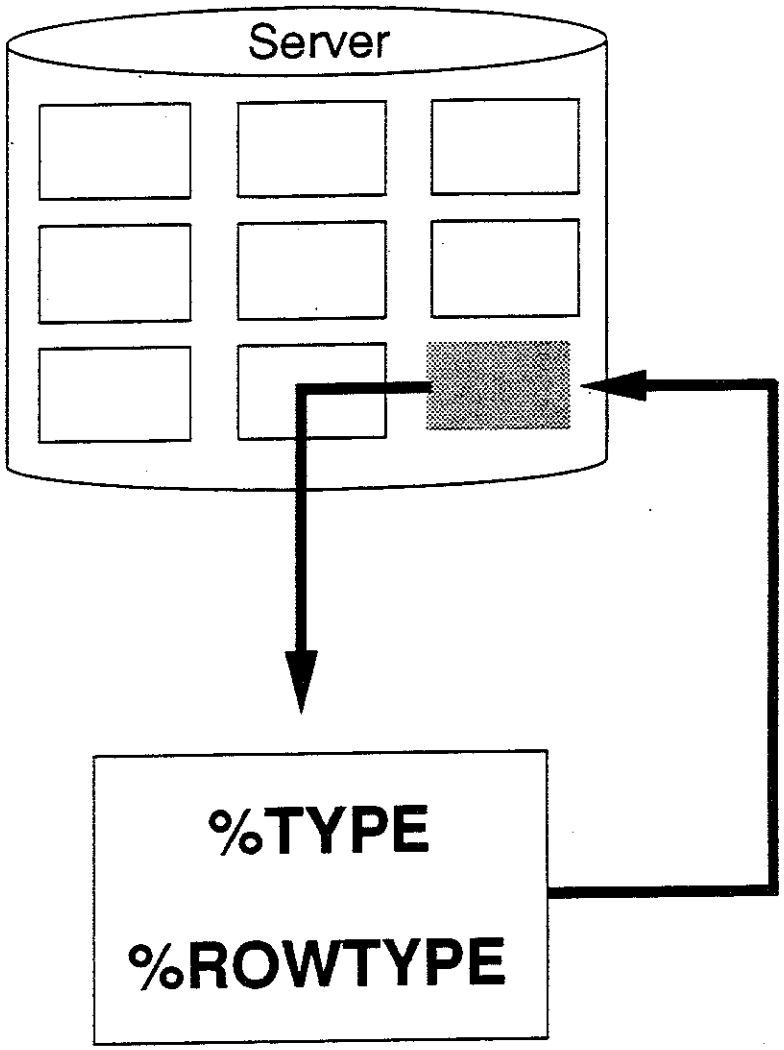
```
DECLARE
TYPE  dept_rec_type IS RECORD
      (deptno  NUMBER(2) NOT NULL,
      dname     dept.dname%TYPE,
      loc       dept.loc%TYPE);
dept_rec   dept_rec_type;
```

## Quick Note

- *dept_rec* represents an entire record.

Declare
dept_rec dept%rowtype; -- saves hole recordin
deptrec like rec.

# DECLARE ATTRIBUTES



## Declare Attributes—cont'd

**Associate PL/SQL objects (such as variables and constants) and database objects (such as columns and tables) with certain attributes.**

### Example - %TYPE attribute

```
DECLARE
books_printed NUMBER(6);
books_sold    books_printed%TYPE;
maiden_name   emp.ename%TYPE;
```

### Example - %ROWTYPE attribute

```
DECLARE
dept_row   dept%ROWTYPE;
```

# ASSIGN VARIABLES

## Assign values to scalar and composite variables.

### Syntax

### Scalar Assignment

```
plsql_variable := plsql_expression;
```

### Table Assignment

```
plsql_table_name (primary_key_value)  := plsql_expression;
```

### Record Assignment

```
record_name.field_name   := plsql_expression;
```

### Quick Notes

- := (ASSIGNMENT)= (VALUE EQUALITY)
- Aggregate assignment between entire records is allowed if the records belong to the same datatype.
- Column or database table references are not allowed on either side of the assignment operator (:=). The following references are illegal:

```
SCOTT.EMP.EMPNO := 1234;
location := dept.loc;
```

## Assign Variables—cont'd

**Create PL/SQL expressions consisting of variables, constants, literals, and function calls.**

### Example

```
DECLARE
      TYPE dept_rec_type  IS RECORD
         (deptno  NUMBER(2) NOT NULL,
         dname dept.dname%TYPE);
      TYPE ename_tab_type IS TABLE OF CHAR(10)
         INDEX BY BINARY_INTEGER;
      dept_rec1   dept_rec_type;
      dept_rec2   dept_rec_type;
      ename_tab   ename_tab_type;
      cnt         BINARY_INTEGER := 0;
      over_budget BOOLEAN;
      last_name   VARCHAR2(10);
BEGIN
      cnt := cnt + 1;
      over_budget := TRUE:
      last_name := 'JONES'
      ename_tab(3)  := last_name;
      dept_rec1 := dept_rec2;
END;
```

dept_rec2.deptno := 10;
dept_rec2.dname := 'edb';

dept_rec1 := dept_rec2; -- gir verdiene fra 2 til dept_rec 1

# CONVERT DATATYPES

**PL/SQL automatically tries to convert datatypes if the expected datatype is not used.**

## *Implicit Conversion Rules*

1 Expression evaluation

2 Variable assignment

## *Rule One: Expression Evaluation*

- PL/SQL can convert the datatypes of operands automatically.

## PL/SQL can automatically convert expressions from:

- CHAR/VARCHAR2 to NUMBER (but **not** NUMBER to CHAR/VARCHAR2).
- CHAR/VARCHAR2 to DATE (but **not** DATE to VARCHAR2).

## Quick Notes

- All arithmetic operators require numbers or dates.
- The concatenation operator requires character.

## Convert Datatypes—cont'd

### *Rule Two: Variable Assignment*

- The datatype on the left and right side of an assignment statement must be the same or implicitly convertible.

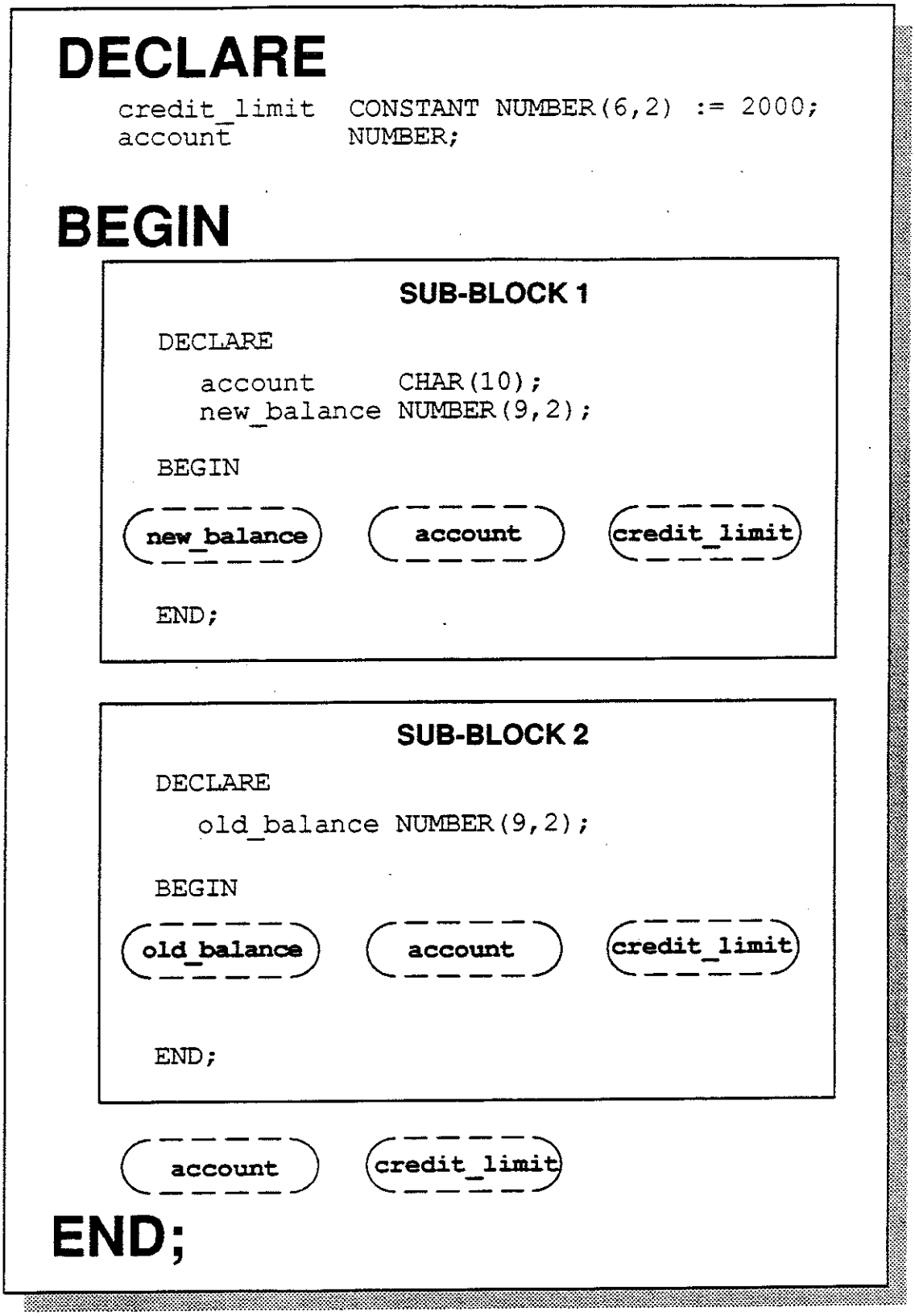
### PL/SQL can automatically convert assignments from:

- CHAR/VARCHAR2 to NUMBER.
- NUMBER to CHAR/VARCHAR2.
- CHAR/VARCHAR2 to DATE.
- DATE to CHAR/VARCHAR2.

### Quick Note

- Assignment also occurs through INSERT, UPDATE, or SELECT ... INTO statements.

# SCOPE VARIABLES AND CONSTANTS



## Scope Variables and Constants—cont'd

**Scope refers to the visibility of identifiers at different points in the PL/SQL block.**

### *Scoping Rules*

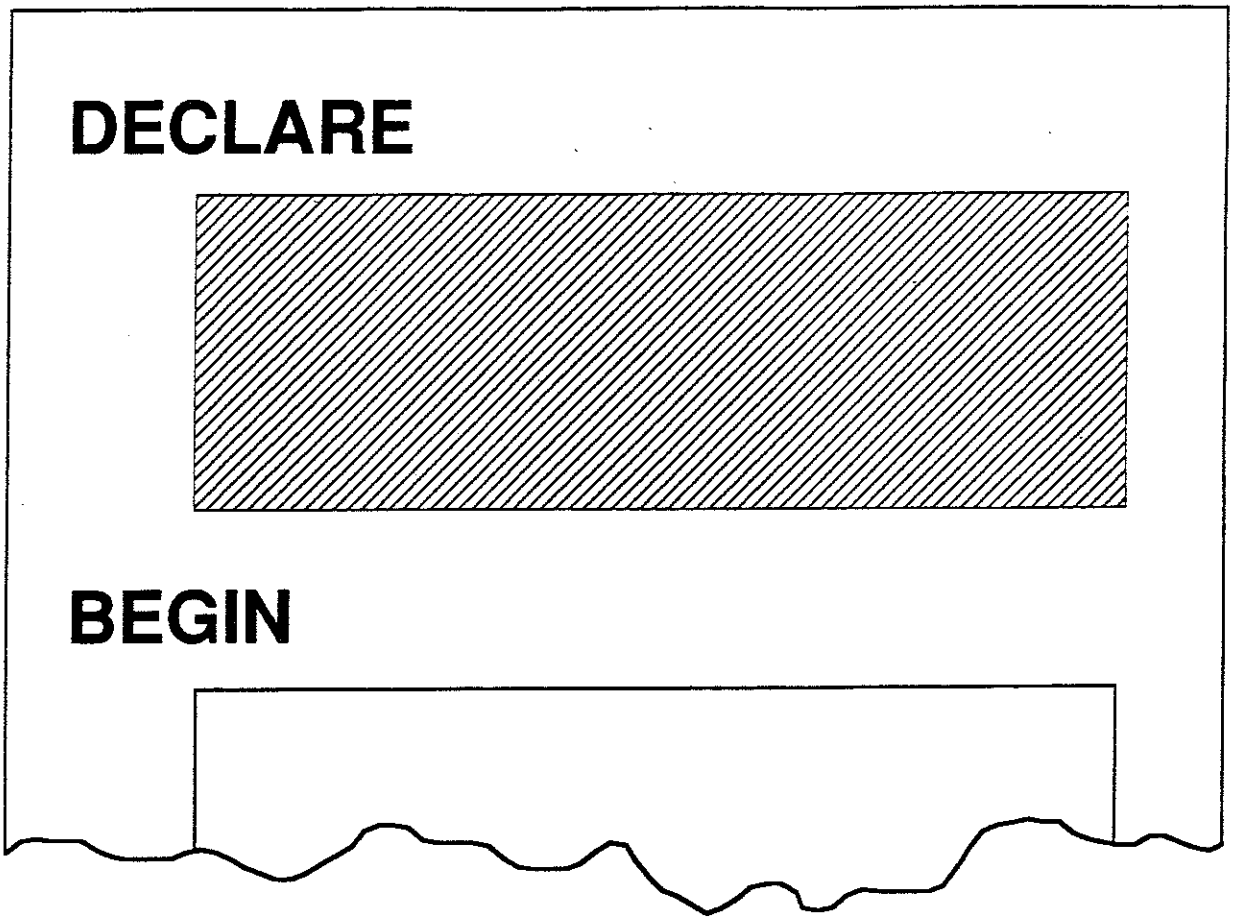
1 An identifier is visible in the block in which it is declared and all nested sub-blocks, procedures, and functions unless rule 2 applies.

2 If an identifier declared in an enclosing block is re-declared in a sub-block, the original identifier declared in the enclosing block is no longer visible in the sub-block. However, the newly declared identifier has the rules of scope defined in rule 1.

For further information on the subject see:

*PL/SQL User's Guide and Reference Version 2.0*, 800-20

# DECLARE VARIABLES: SUMMARY




## *Declare Constants and Variables*

- PL/SQL supports scalar and composite datatypes.
- Scalar datatypes have no internal components. VAR_CHAR2, BINARY_INTEGER, and BOOLEAN are examples.
- Composite datatypes have internal components that can be manipulated individually. The two composite datatypes are TABLE (similar to an array) and RECORD.
- Associate variables and constants with database objects (columns and rows) by using the %TYPE and %ROWTYPE attributes.
- PL/SQL automatically tries to convert datatypes if the expected datatype is not used.
- Declared objects exist within a specific scope.

# EXERCISE 1-1

**Evaluate each of the following declarations. Determine which of the following are *not* legal and explain why.**

1
```
DECLARE
my_var   NUMBER(7,2);
```

2
```
DECLARE
x, y, z  CHAR(10);
```

3
```
DECLARE
birthdate DATE NOT NULL;
```

4
```
DECLARE
in_stock  BOOLEAN := 1;
```

5
```
DECLARE
emp_rec  emp_rec_type;
```

6
```
DECLARE
TYPE  dname_tab_type IS TABLE OF CHAR(20)
     INDEX BY BINARY_INTEGER;
dname_tab  dname_tab_type;
```

## Exercise 1-1—cont'd

```
DECLARE
   weight      NUMBER := 600;
   location    CHAR(10) :='Europe';
BEGIN
   SUB-BLOCK 1
   DECLARE
      weight      NUMBER := 1;
      new_loc     CHAR(20);
   BEGIN
      weight := weight + 1;          (1)
      new_loc:= 'Western ' || location;   (2)
   END;
   SUB-BLOCK 2
   DECLARE
      new_wt      NUMBER;
      continent   CHAR(10);
   BEGIN
      continent := location;         (3)
      new_wt := weight + 10;         (4)
   END;
   weight := weight - 50;            (5)
   location := new_loc || ' and Asia';   (6)
END;
```

## Exercise 1-1—cont'd

**Evaluate the PL / SQL block on the opposite page. Apply the scoping guidelines presented in this section to determine the values indicated.**

1. 2
2. Western Europe 00000
3. Europe0000
4. ~~NULL~~ 610
5. 550
6. Western Error, new book 17300 Inn ; SWS World.