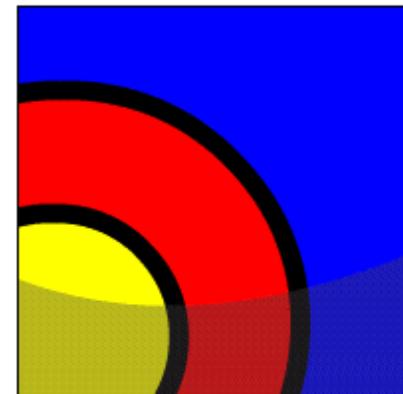


Writing PL/SQL Executable Statements

What Will I Learn?

In this lesson, you will learn to:

- Construct accurate variable assignment statements in PL/SQL
- Construct accurate statements using built-in SQL functions in PL/SQL
- Differentiate between implicit and explicit conversions of data types
- Describe when implicit conversions of data types take place
- List the drawbacks of implicit data type conversions
- Construct accurate statements using functions to explicitly convert data types
- Construct statements using operators in PL/SQL

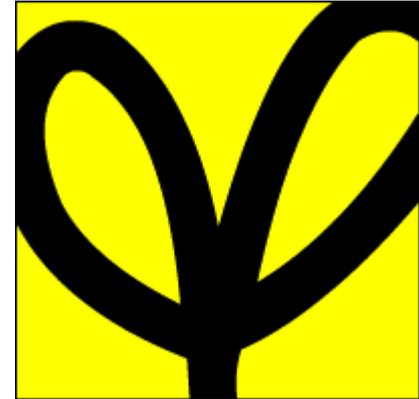


Why Learn It?

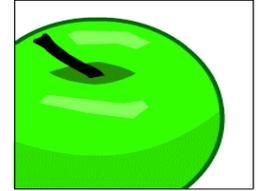
We've introduced variables and identifiers.

Now, you build your knowledge of the PL/SQL programming language by writing code to assign variable values. These values can be literals.

They can also be functions. SQL provides a number of predefined functions that you can use in SQL statements. Most of these functions are also valid in PL/SQL expressions.



Tell Me/Show Me



Assigning New Values to Variables

- Character and date literals must be enclosed in single quotation marks.

```
v_name      := 'Henderson';  
v_start_date := '12-DEC-2005';
```

- Statements can continue over several lines.

```
v_quote := 'The only thing that we can  
know is that we know nothing and that  
is the highest flight of human  
reason.';
```

- Numbers can be simple values or scientific notation.

```
v_my_integer := 100;  
v_my_sci_not := 2E5;
```

(2E5 meaning 2×10 to the power of 5 = 200,000)



Tell Me/Show Me

SQL Functions in PL/SQL

You are already familiar with functions in SQL statements. For example:

```
SELECT country_name, LAST_DAY(date_of_independence)
FROM wf_countries
WHERE date_of_independence IS NOT NULL;
```

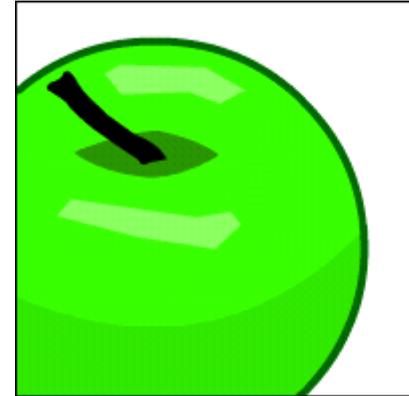
You can also use these functions in PL/SQL procedural statements. For example:

```
DECLARE
    v_last_day DATE;
BEGIN
    v_last_day := LAST_DAY(SYSDATE);
    DBMS_OUTPUT.PUT_LINE(v_last_day);
END;
```

Tell Me/Show Me

SQL Functions in PL/SQL

- Available in procedural statements:
 - Single-row character
 - Single-row number
 - Date
 - Data-type conversion
 - Miscellaneous functions
- Not available in procedural statements:
 - DECODE
 - Group functions



 **Tell Me/Show Me****Character Functions**

Valid character functions in PL/SQL include:

ASCII	LENGTH	RPAD
CHR	LOWER	RTRIM
CONCAT	LPAD	SUBSTR
INITCAP	LTRIM	TRIM
INSTR	REPLACE	UPPER

This is not an exhaustive list. Refer to the Oracle documentation for the complete list.

Tell Me/Show Me

Examples of Character Functions

- Get the length of a string:

```
v_desc_size          INTEGER(5);  
v_prod_description  VARCHAR2(70):='You can use this  
product with your radios for higher frequency';  
  
-- get the length of the string in prod_description  
v_desc_size:= LENGTH(v_prod_description);
```

- Convert the name of the country capitol to upper case:

```
v_capitol_name:= UPPER(v_capitol_name);
```

- Concatenate the first and last names:

```
v_emp_name:= v_first_name || ' ' || v_last_name;
```

 **Tell Me/Show Me****Number Functions**

Valid number functions in PL/SQL include:

ABS	EXP	ROUND
ACOS	LN	SIGN
ASIN	LOG	SIN
ATAN	MOD	TAN
COS	POWER	TRUNC

This is not an exhaustive list. Refer to the Oracle documentation for the complete list.



Tell Me/Show Me

Examples of Number Functions

- Get the sign of a number:

```
DECLARE
  v_my_num BINARY_INTEGER := -56664;
BEGIN
  DBMS_OUTPUT.PUT_LINE(SIGN(v_my_num));
END;
```

- Round a number to 0 decimal places:

```
DECLARE
  v_median_age NUMBER(6,2);
BEGIN
  SELECT median_age INTO v_median_age
  FROM wf_countries WHERE country_id=27;
  DBMS_OUTPUT.PUT_LINE(ROUND(v_median_age,0));
END;
```



Tell Me/Show Me

Date Functions

Valid date functions in PL/SQL include:

ADD_MONTHS	MONTHS_BETWEEN
CURRENT_DATE	ROUND
CURRENT_TIMESTAMP	SYSDATE
LAST_DAY	TRUNC

This is not an exhaustive list. Refer to the Oracle documentation for the complete list.



Tell Me/Show Me

Examples of Date Functions

- Add months to a date:

```
DECLARE
  v_new_date    DATE;
  v_num_months  NUMBER := 6;
BEGIN
  v_new_date := ADD_MONTHS(SYSDATE,v_num_months);
  DBMS_OUTPUT.PUT_LINE(v_new_date);
END;
```

- Calculate the number of months between two dates:

```
DECLARE
  v_no_months  PLS_INTEGER:=0;
BEGIN
  v_no_months := MONTHS_BETWEEN('31-JAN-06','31-MAY-05');
  DBMS_OUTPUT.PUT_LINE(v_no_months);
END;
```



Tell Me/Show Me

Data-Type Conversion

In any programming language, converting one data type to another is a common requirement. PL/SQL can handle such conversions with scalar data types. Data-type conversions can be of two types:

- Implicit conversions
- Explicit conversions

Tell Me/Show Me

Implicit Conversions

In implicit conversions, PL/SQL attempts to convert data types dynamically if they are mixed in a statement. Implicit conversions can happen between many types in PL/SQL, as illustrated by the following chart.

	DATE	LONG	NUMBER	PLS_INTEGER	VARCHAR2
DATE	N/A	X			X
LONG		N/A			X
NUMBER		X	N/A	X	X
PLS_INTEGER		X	X	N/A	X
VARCHAR2	X	X	X	X	N/A



Tell Me/Show Me

Example of Implicit Conversion

Consider the following example:

```
DECLARE
  v_salary          NUMBER(6) := 6000;
  v_sal_increase    VARCHAR2(5) := '1000';
  v_total_salary    v_salary%TYPE;
BEGIN
  v_total_salary := v_salary + v_sal_increase;
  DBMS_OUTPUT.PUT_LINE(v_total_salary);
END;
```

In this example, the variable `v_sal_increase` is of type `VARCHAR2`. While calculating the total salary, PL/SQL first converts `v_sal_increase` to `NUMBER` and then performs the operation. The result of the operation is the `NUMBER` type.

Tell Me/Show Me

Drawbacks of Implicit Conversions

At first glance, implicit conversions might seem useful; however, there are several drawbacks:

- Implicit conversions can be slower.
- When you use implicit conversions, you lose control over your program because you are making an assumption about how Oracle handles the data. If Oracle changes the conversion rules, then your code can be affected.
- Implicit conversion rules depend upon the environment in which you are running. For example, the date format varies depending on the language setting and installation type. Code that uses implicit conversion might not run on a different server or in a different language.
- Code that uses implicit conversion is harder to read and understand.



Tell Me/Show Me

Drawbacks of Implicit Conversions

It is the programmer's responsibility to ensure that values can be converted. For instance, PL/SQL can convert the CHAR value '02-JUN-92' to a DATE value, but cannot convert the CHAR value 'Yesterday' to a DATE value. Similarly, PL/SQL cannot convert a VARCHAR2 value containing alphabetic characters to a NUMBER value.

Valid?	Statement
Yes	<code>v_new_date DATE := '02-JUN-1992';</code>
No	<code>v_new_date DATE := 'Yesterday';</code>
Yes	<code>v_my_number NUMBER := '123';</code>
No	<code>v_my_number NUMBER := 'abc';</code>



Tell Me/Show Me

Explicit Conversions

Explicit conversions convert values from one data type to another by using built-in functions. Examples of conversion functions include:

<code>TO_NUMBER ()</code>	<code>ROWIDTONCHAR ()</code>
<code>TO_CHAR ()</code>	<code>HEXTORAW ()</code>
<code>TO_CLOB ()</code>	<code>RAWTOHEX ()</code>
<code>CHARTOROWID ()</code>	<code>RAWTONHEX ()</code>
<code>ROWIDTOCHAR ()</code>	<code>TO_DATE ()</code>



Tell Me/Show Me

Examples of Explicit Conversions

TO_CHAR

```
BEGIN
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(SYSDATE, 'Month YYYY'));
END;
```

TO_DATE

```
BEGIN
  DBMS_OUTPUT.PUT_LINE(TO_DATE('April-1999', 'Month-YYYY'));
END;
```



Tell Me/Show Me

Examples of Explicit Conversions (continued)

TO_NUMBER

```
DECLARE
  v_a VARCHAR2(10) := '-123456';
  v_b VARCHAR2(10) := '+987654';
  v_c PLS_INTEGER;
BEGIN
  v_c := TO_NUMBER(v_a) + TO_NUMBER(v_b);
  DBMS_OUTPUT.PUT_LINE(v_c);
END;
```



Tell Me/Show Me

Data Type Conversion Example

1

```
v_date_of_joining DATE:= '02-Feb-2000';
```

2

```
v_date_of_joining DATE:= 'February 02,2000';
```

3

```
v_date_of_joining DATE:= TO_DATE('February  
02,2000','Month DD,YYYY');
```

Tell Me/Show Me

Operators in PL/SQL

- Logical
- Arithmetic
- Concatenation
- Parentheses to control the order of operations

- Exponential operator (**)

} **Same as in SQL**

The operations within an expression are performed in a particular order depending on their precedence (priority).



Tell Me/Show Me

Operators in PL/SQL

The following table shows the default order of operations from high priority to low priority:

Operator	Operation
**	Exponentiation
+, -	Identity, negation
*, /	Multiplication, division
+, -,	Addition, subtraction, concatenation
=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN	Comparison
NOT	Logical negation
AND	Conjunction
OR	Inclusion



Tell Me/Show Me

Operators in PL/SQL

Examples:

- Increment the counter for a loop.

```
v_loop_count := v_loop_count + 1;
```

- Set the value of a Boolean flag.

```
v_good_sal := v_sal BETWEEN 50000 AND 150000;
```

- Validate whether an employee number contains a value.

```
v_valid      := (v_empno IS NOT NULL);
```

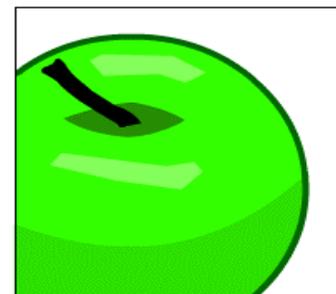
Tell Me / Show Me

Terminology

Key terms used in this lesson include:

Implicit conversion

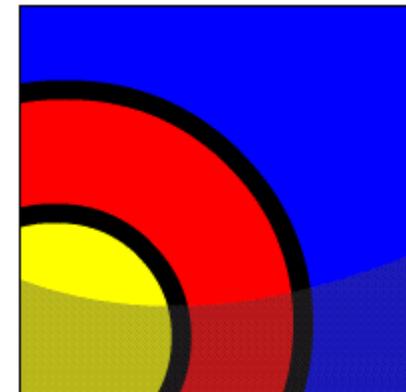
Explicit conversion



Summary

In this lesson, you have learned how to:

- Construct accurate variable assignment statements in PL/SQL
- Construct accurate statements using built-in SQL functions in PL/SQL
- Differentiate between implicit and explicit conversions of data types
- Describe when implicit conversions of data types take place
- List the drawbacks of implicit data-type conversions
- Construct accurate statements using functions to explicitly convert data types
- Construct statements using operators in PL/SQL





Try It/Solve It

The exercises for this lesson cover the following topics:

- Constructing accurate variable assignment statements in PL/SQL
- Constructing accurate statements using built-in SQL functions in PL/SQL
- Differentiating between implicit and explicit data-type conversions
- Describing when implicit data type conversions take place
- Listing the drawbacks of implicit data type conversions
- Constructing accurate statements using functions to explicitly convert data types
- Constructing statements using operators in PL/SQL

