Database Programming with PL/SQL

2-5
Writing PL/SQL Executable Statements
Objectives

This lesson covers the following objectives:

• 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
Purpose

• We’ve introduced variables and identifiers.
• In this lesson, you build your knowledge of the PL/SQL programming language by writing code to assign variable values.
• These values can be literals or values returned by a function.
• 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.
Assigning New Values to Variables

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

```plaintext
v_name       := 'Henderson';
v_start_date := '12-Dec-2005';
```

• Statements can continue over several lines.

```plaintext
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 (2E5 meaning 2x10 to the power of 5 = 200,000).

```plaintext
v_my_integer := 100;
v_my_sci_not := 2E5;
```
SQL Functions in PL/SQL

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

```sql
SELECT LAST_DAY(SYSDATE)
FROM DUAL;
```

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

```sql
DECLARE
  v_last_day DATE;
BEGIN
  v_last_day := LAST_DAY(SYSDATE);
  DBMS_OUTPUT.PUT_LINE(v_last_day);
END;
```
SQL Functions in PL/SQL

• Functions available in procedural statements:
  – Single-row character
  – Single-row number
  – Date
  – Data-type conversion
  – Miscellaneous functions

• Not available in procedural statements:
  – DECODE
  – Group functions
Character Functions

• Valid character functions in PL/SQL include:

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>LENGTH</td>
<td>RPAD</td>
</tr>
<tr>
<td>CHR</td>
<td>LOWER</td>
<td>RTRIM</td>
</tr>
<tr>
<td>CONCAT</td>
<td>LPAD</td>
<td>SUBSTR</td>
</tr>
<tr>
<td>INITCAP</td>
<td>LTRIM</td>
<td>TRIM</td>
</tr>
<tr>
<td>INSTR</td>
<td>REPLACE</td>
<td>UPPER</td>
</tr>
</tbody>
</table>

• This is not an exhaustive list.

• Refer to the Oracle documentation for the complete list.
Examples of Character Functions

• Get the length of a string:

```sql
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:

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

• Concatenate the first and last names:

```sql
v_emp_name:= v_first_name||' '||v_last_name;
```
Number Functions

- Valid number functions in PL/SQL include:

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>EXP</td>
<td>ROUND</td>
</tr>
<tr>
<td>ACOS</td>
<td>LN</td>
<td>SIGN</td>
</tr>
<tr>
<td>ASIN</td>
<td>LOG</td>
<td>SIN</td>
</tr>
<tr>
<td>ATAN</td>
<td>MOD</td>
<td>TAN</td>
</tr>
<tr>
<td>COS</td>
<td>POWER</td>
<td>TRUNC</td>
</tr>
</tbody>
</table>

- This is not an exhaustive list.
- Refer to the Oracle documentation for the complete list.
Examples of Number Functions

• Get the sign of a number:

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

• Round a number to 0 decimal places:

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

• Valid date functions in PL/SQL include:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_MONTHS</td>
<td>MONTHS_BETWEEN</td>
</tr>
<tr>
<td>CURRENT_DATE</td>
<td>ROUND</td>
</tr>
<tr>
<td>CURRENT_TIMESTAMP</td>
<td>SYSDATE</td>
</tr>
<tr>
<td>LAST_DAY</td>
<td>TRUNC</td>
</tr>
</tbody>
</table>

• This is not an exhaustive list.

• Refer to the Oracle documentation for the complete list.
Examples of Date Functions

• Add months to a date:

```sql
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:

```sql
DECLARE
  v_no_months  PLS_INTEGER := 0;
BEGIN
  v_no_months := MONTHS_BETWEEN('31-Jan-2006','31-May-2005');
  DBMS_OUTPUT.PUT_LINE(v_no_months);
END;
```
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
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.

<table>
<thead>
<tr>
<th></th>
<th>DATE</th>
<th>LONG</th>
<th>NUMBER</th>
<th>PLS_INTEGER</th>
<th>VARCHAR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>N/A</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>LONG</td>
<td></td>
<td>N/A</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>NUMBER</td>
<td>X</td>
<td>N/A</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PLS_INTEGER</td>
<td>X</td>
<td>X</td>
<td></td>
<td>N/A</td>
<td>X</td>
</tr>
<tr>
<td>VARCHAR2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>N/A</td>
</tr>
</tbody>
</table>
Example of Implicit Conversion

• 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.

```sql
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;
```
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.

• Code that uses implicit conversion is harder to read and understand.
Drawbacks of Implicit Conversions

Additional drawbacks:

• 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.

• It is strongly recommended that you **AVOID** allowing SQL or PL/SQL to perform implicit conversions on your behalf.

• You should use conversion functions to guarantee that the right kinds of conversions take place.
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-1992' 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.

<table>
<thead>
<tr>
<th>Valid?</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><code>v_new_date DATE := '02-Jun-1992';</code></td>
</tr>
<tr>
<td>No</td>
<td><code>v_new_date DATE := 'Yesterday';</code></td>
</tr>
<tr>
<td>Yes</td>
<td><code>v_my_number NUMBER := '123';</code></td>
</tr>
<tr>
<td>No</td>
<td><code>v_my_number NUMBER := 'abc';</code></td>
</tr>
</tbody>
</table>
Explicit Conversions

• Explicit conversions convert values from one data type to another by using built-in functions.

• Examples of conversion functions include:

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TO_NUMBER()</td>
<td>ROWIDTONCHAR()</td>
</tr>
<tr>
<td>TO_CHAR()</td>
<td>HEXTORAW()</td>
</tr>
<tr>
<td>TO_CLOB()</td>
<td>RAWTOHEX()</td>
</tr>
<tr>
<td>CHARTOROWID()</td>
<td>RAWTONHEX()</td>
</tr>
<tr>
<td>ROWIDTOCHAR()</td>
<td>TO_DATE()</td>
</tr>
</tbody>
</table>
Examples of Explicit Conversions

- **TO_CHAR**

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

- **TO_DATE**

```sql
BEGIN
    DBMS_OUTPUT.PUT_LINE(TO_DATE('April-1999', 'Month-YYYY'));
END;
```
Examples of Explicit Conversions

• **TO_NUMBER**

```sql
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;
```
Data Type Conversion Examples

• Example #1

```plsql
v_date_of_joining DATE := '02-Feb-2014';
```

• Example #2

```plsql
v_date_of_joining DATE := 'February 02, 2014';
```

• Example #3

```plsql
v_date_of_joining DATE := TO_DATE('February 02, 2014', 'Month DD, YYYY');
```
Operators in PL/SQL

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

- Logical
- Arithmetic
- Concatenation
- Parentheses to control the order of operations
- Exponential operator (**)

Same as in SQL
Operators in PL/SQL

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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>+, -</td>
<td>Identity, negation</td>
</tr>
<tr>
<td>*, /</td>
<td>Multiplication, division</td>
</tr>
<tr>
<td>+, -,</td>
<td></td>
</tr>
<tr>
<td>=, &lt;, &gt;, &lt;=, &gt;=, &lt;&gt;, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN</td>
<td>Comparison</td>
</tr>
<tr>
<td>NOT</td>
<td>Logical negation</td>
</tr>
<tr>
<td>AND</td>
<td>Conjunction</td>
</tr>
<tr>
<td>OR</td>
<td>Inclusion</td>
</tr>
</tbody>
</table>
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.
  \[ v\_good\_sal := v\_sal \text{ BETWEEN } 50000 \text{ AND } 150000; \]

• Validate whether an employee number contains a value.
  \[ v\_valid := (v\_empno \text{ IS NOT NULL}); \]
Terminology

Key terms used in this lesson included:

• Explicit conversion
• Implicit conversion
Summary

In this lesson, you should 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