PL/SQL QUERIES

Introduction

Overview

- PL/SQL Block Structure
- Executing PL/SQL
- Generating Output
- Variables
- Data types
- Decision control structures

PL/SQL

- PL/SQL is Oracle's *procedural* language extension to SQL, the non-procedural relational database language.
- With PL/SQL, you can use SQL statements to manipulate ORACLE data and the *flow* of control statements to process the data.
- Moreover, you can declare constants and variables, define subprograms (procedures and functions), and trap runtime errors.
- PL/SQL combines the data manipulating power of SQL with the data processing power of procedural languages.

PL/SQL

- Many Oracle applications are built using client/server architecture. The Oracle database resides on the server.
- The program that makes requests against this database resides on the client machine.
- This program can be written in C, Java, or PL/SQL.
- While PL/SQL is just like any other programming language, it has syntax and rules that determine how programming statements work together. It is important for you to realize that PL/SQL is not a stand alone programming language.
- PL/SQL is a part of the Oracle RDBMS, and it can reside in two environments, the client and the server.

PL/SQL

- As a result, it is very easy to move PL/SQL modules between server side and client side applications.
- When the PL/SQL engine is located on the server, the whole PL/SQL block is passed to the PL/SQL engine on the Oracle server.
- The PL/SQL engine processes the block.

PL/SQL

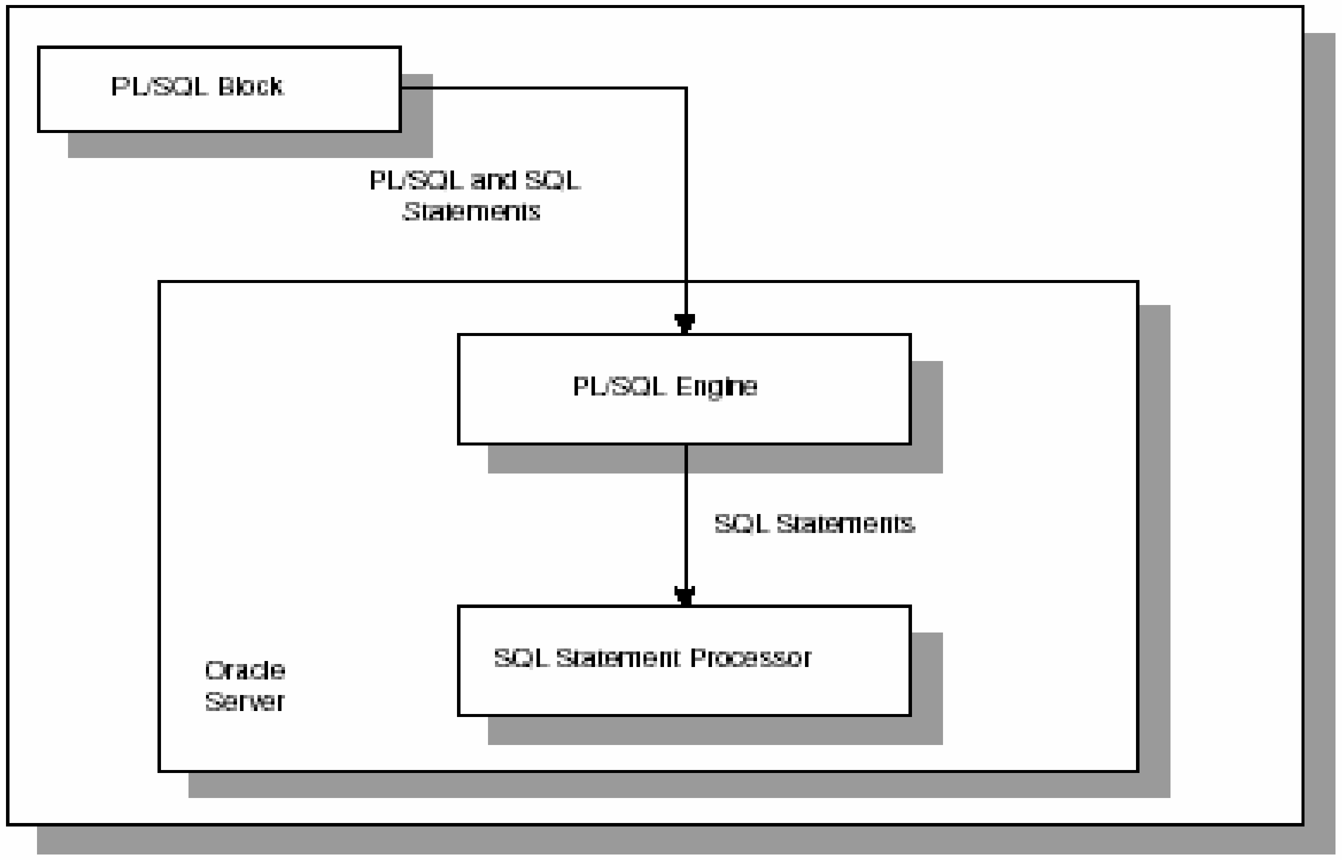


Figure 2.1 ■ The PL/SQL engine and Oracle server.

PL/SQL

- When the PL/SQL engine is located on the client, as it is in the Oracle Developer Tools, the PL/SQL processing is done on the client side.
- All SQL statements that are embedded within the PL/SQL block are sent to the Oracle server for further processing.
- When PL/SQL block contains no SQL statement, the entire block is executed on the client side.

DIFFERENCE BETWEEN PL/SQL AND SQL

- When a SQL statement is issued on the client computer, the request is made to the database on the server, and the result is sent back to the client.
- As a result, a single SQL statement causes two trips on the network. If multiple SELECT statements are issued, the network traffic increase significantly very fast. For example, four SELECT statements cause eight network trips.
- If these statements are part of the PL/SQL block, they are sent to the server as a single unit. The SQL statements in this PL/SQL program are executed at the server and the result set is sent back as a single unit. There is still only one network trip made as is in case of a single SELECT statement.

Comparison of SQL*PLUS and PL/SQL

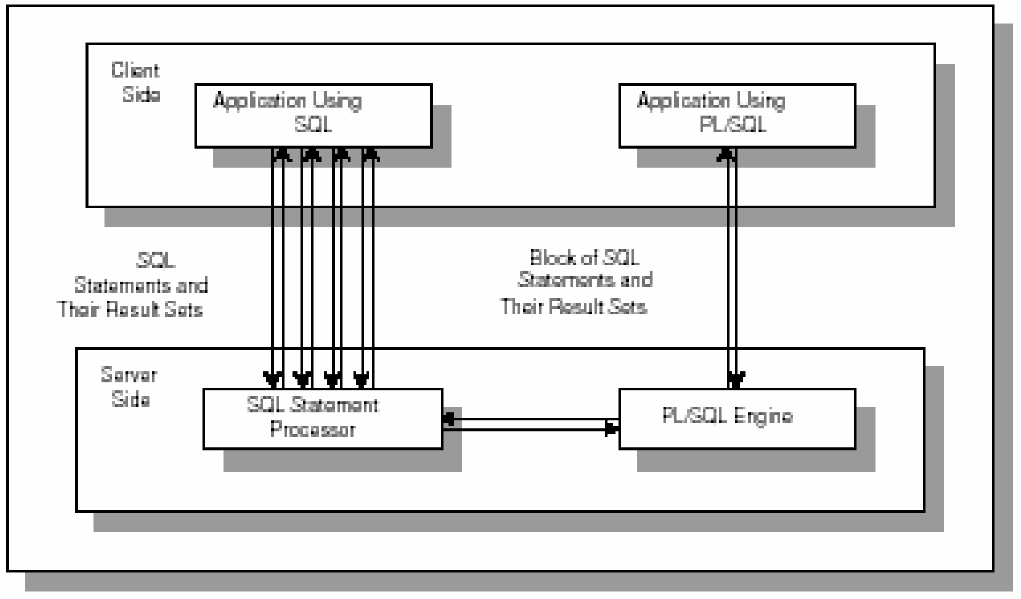


Figure 2.2 ■ **PL/SQL in client-server architecture.**

PL/SQL BLOCKS

- PL/SQL blocks can be divided into two groups:
  - Named
  - Anonymous
- Named blocks are used when creating subroutines. These subroutines are procedures, functions, and packages.
- The subroutines can be stored in the database and referenced by their names later on.
- In addition, subroutines can be defined within the anonymous PL/SQL block.
- Anonymous PL/SQL blocks do not have names. As a result,they cannot be stored in the database and referenced later.

PL/SQL BLOCK STRUCTURE

- PL/SQL blocks contain three sections
  - Declare section
  - Executable section and
  - Exception-handling section.
- The executable section is the only mandatory section of the block.
- Both the declaration and exception handling sections are optional.

PL/SQL BLOCK STRUCTURE

```
DECLARE
    Declaration statements
BEGIN
    Executable statements
EXCETION
    Exception handling statements
END ;
```

DECLARATION SECTION

- The *declaration section* is the first section of the PL/SQL block.
- It contains definitions of PL/SQL identifiers such as variables, constants, cursors and so on.
- Example

```
 DECLARE
      v_first_name VARCHAR2(35) ;
      v_last_name VARCHAR2(35) ;
      v_counter NUMBER := 0 ;
```

EXECUTABLE SECTION

- The executable section is the next section of the PL/SQL block.
- This section contains executable statements that allow you to manipulate the variables that have been declared in the declaration section.

```
BEGIN
    SELECT first_name, last_name
        INTO v_first_name, v_last_name
        FROM student
        WHERE student_id = 123 ;
    DBMS_OUTPUT.PUT_LINE
    ('Student name :' || v_first_name ||'  '||
    v_last_name);
END;
```

EXCEPTION-HANDLING SECTION

- The *exception-handling section* is the last section of the PL/SQL block.
- This section contains statements that are executed when a runtime error occurs within a block.
- Runtime errors occur while the program is running and cannot be detected by the PL/SQL compiler.

```
EXCEPTION
     WHEN NO_DATA_FOUND THEN
     DBMS_OUTPUT.PUT_LINE
         (‘ There is no student with student id 123 ’);
END;
```

HOW PL/SQL GETS EXECUTED

- Every time an anonymous block is executed, the code is sent to the PL/SQL engine on the server where it is compiled.
- The named PL/SQL block is compiled only at the time of its creation, or if it has been changed.
- The compilation process includes syntax checking, binding and p - code generation.
- Syntax checking involves checking PL/SQL code for syntax or compilation errors.
- Once the programmer corrects syntax errors, the compiler can assign a storage address to program variables that are used to hold data for Oracle. This process is called ***Binding.***

HOW PL/SQL GETS EXECUTED

- After binding, p - code is generated for the PL/SQL block.
- P - code is a list of instructions to the PL/SQL engine.
- For named blocks, p - code is stored in the database, and it is used the next time the program is executed.
- Once the process of compilation has completed successfully, the status for a named PL/SQL block is set to VALID, and also stored in the database.
- If the compilation process was not successful, the status for a named PL/SQL block is set to INVALID.

PL/SQL IN SQL*PLUS

- SQL*Plus is an interactive tool that allows you to type SQL or PL/SQL statements at the command prompt.
- These statements are then sent to the database. Once they are processed, the results are sent back from the database and displayed on the screen.
- There are some differences between entering SQL and PL/SQL statements.

SQL EXAMPLE

- SELECT first_name, last_name

  FROM student;
- The semicolon terminates this SELECT statement. Therefore, as soon as you type semicolon and hit the ENTER key, the result set is displayed to you.

PL/SQL EXAMPLE

```
DECLARE
    v_first_name VARCHAR2(35);
    v_last_name VARCHAR2(35);
BEGIN
    SELECT first_name, last_name
    INTO v_first_name, v_last_name
    FROM student
    WHERE student_id = 123;
    DBMS_OUTPUT.PUT_LINE
    ('Student name: '||v_first_name||' '||v_last_name);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE
                    ('There is no student with student id 123');
END;
.
/
```

PL/SQL EXAMPLE

- There are two additional lines at the end of the block containing "." and "/". The "." marks the end of the PL/SQL block and is optional.
- The "/" executes the PL/SQL block and is required.
- When SQL*Plus reads SQL statement, it knows that the semicolon marks the end of the statement. Therefore, the statement is complete and can be sent to the database.
- When SQL*Plus reads a PL/SQL block, a semicolon marks the end of the individual statement within the block. In other words, it is not a block terminator.

PL/SQL EXAMPLE

- Therefore, SQL*Plus needs to know when the block has ended. As you have seen in the example, it can be done with period and forward slash.

EXECUTING PL/SQL

- PL/SQL can be executed directly in SQL*Plus. A PL/SQL program is normally saved with an .sql extension.
- To execute an anonymous PL/SQL program, simply type the following command at the SQL prompt:

```
SQL> @DisplayAge
```

GENERATING OUTPUT

- Like other programming languages, PL/SQL provides a procedure (i.e. PUT_LINE) to allow the user to display the output on the screen.
- For a user to able to view a result on the screen, two steps are required.
  - First, before executing any PL/SQL program, type the following command at the SQL prompt (Note: you need to type in this command only once for every SQL*PLUS session):

    ```
    SQL>   SET SERVEROUTPUT ON;
    ```
  - Or put the command at the beginning of the program, right before the declaration section.

GENERATING OUTPUT

- Second, use DBMS_OUTPUT.PUT_LINE in your executable section to display any message you want to the screen.
- Syntax for displaying a message:

  DBMS_OUTPUT.PUT_LINE(<string>);

  - in which PUT_LINE is the procedure to generate the output on the screen, and DBMS_OUTPUT is the package to which the PUT_LINE belongs.
- DBMS_OUTPUT_PUT_LINE(‘My age is ‘ || num_age);

SUBSTITUTION VARIABLES

- SQL*Plus allows a PL/SQL block to receive input information with the help of substitution variables.
- Substitution variables cannot be used to output the values because no memory is allocated for them.
- SQL*Plus will substitute a variable before the PL/SQL block is sent to the database.
- Substitution variables are usually prefixed by the ampersand(&) character or double ampersand (&&) character.

EXAMPLE

```
DECLARE
    v_student_id NUMBER := &sv_student_id;
    v_first_name VARCHAR2(35);
    v_last_name VARCHAR2(35);
BEGIN
    SELECT first_name, last_name
    INTO v_first_name, v_last_name
    FROM student
    WHERE student_id = v_student_id;
    DBMS_OUTPUT.PUT_LINE
           ('Student name: '||v_first_name||' '||v_last_name);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('There is no such student');
END;
```

EXAMPLE

- When this example is executed, the user is asked to provide a value for the student ID.
- The example shown above uses a single ampersand for the substitution variable.
- When a single ampersand is used throughout the PL/SQL block, the user is asked to provide a value for each occurrence of the substitution variable.

EXAMPLE

```
BEGIN
      DBMS_OUTPUT.PUT_LINE('Today is '||'&sv_day');
      DBMS_OUTPUT.PUT_LINE('Tomorrow will be '||' &sv_day');
END;
```

Output

```
Enter value for sv_day: Monday
old 2: DBMS_OUTPUT.PUT_LINE('Today is '||' &sv_day');
new 2: DBMS_OUTPUT.PUT_LINE('Today is '||' Monday');
Enter value for sv_day: Tuesday
old 3: DBMS_OUTPUT.PUT_LINE('Tomorrow will be '||'
      &sv_day');
new 3: DBMS_OUTPUT.PUT_LINE('Tomorrow will be '||'
      Tuesday');
Today is Monday
Tomorrow will be Tuesday
PL/SQL procedure successfully completed.
```

EXAMPLE

- When a substitution variable is used in the script, the output produced by the program contains the statements that show how the substitution was done.
- If you do not want to see these lines displayed in the output produced by the script, use the SET command option before you run the script as shown below:

  SET VERIFY OFF;

EXAMPLE

- Then, the output will be:

```
Enter value for sv_day: Monday
Enter value for sv_day: Tuesday
Today is Monday
Tomorrow will be Tuesday
PL/SQL procedure successfully completed.
```

- The substitution variable sv_day appears twice in this PL/SQL block. As a result, when this example is run, the user is asked twice to provide the value for the same variable.

EXAMPLE

```
BEGIN
    DBMS_OUTPUT.PUT_LINE('Today is '||'&&sv_day');
    DBMS_OUTPUT.PUT_LINE('Tomorrow will be ‘||’
    &sv_day');
END;
```

- In this example, substitution variable sv_day is prefixed by double ampersand in the first DBMS_OUTPUT.PUT_LINE statement. As a result, this version of the example produces different output.

OUTPUT

```
Enter value for sv_day: Monday
old 2: DBMS_OUTPUT.PUT_LINE('Today is '||' &&sv_day');
new 2: DBMS_OUTPUT.PUT_LINE('Today is '||' Monday');
old 3: DBMS_OUTPUT.PUT_LINE('Tomorrow will be '||'
    &sv_day');
new 3: DBMS_OUTPUT.PUT_LINE('Tomorrow will be '||'
    Monday');
Today is Monday
Tomorrow will be Monday
PL/SQL procedure successfully completed.
```

- It is clear that the user is asked only once to provide the value for the substitution variable sv_day.
- As a result, both DBMS_OUTPUT.PUT_LINE statements use the value of Monday entered previously by the user.

Substitution Variables

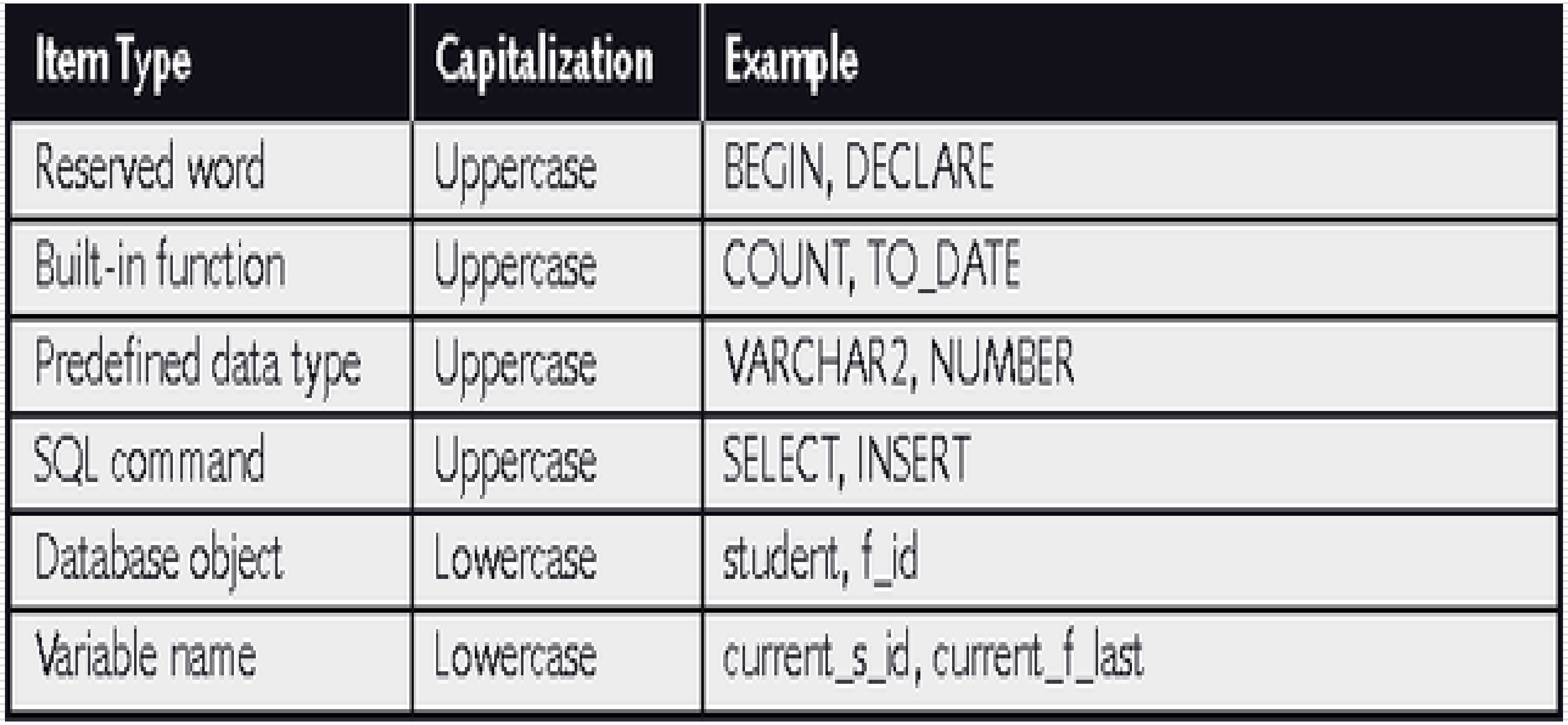
- Ampersand(&) character and double ampersand (&&) characters are the default characters that denote substitution variables.
- There is a special SET command option available in SQL*Plus that allows to change the default character (&) to any other character or disable the substitution variable feature.
- This SET command has the following syntax:

SET DEFINE *character*
or
SET DEFINE ON
or
SET DEFINE OFF

Substitution Variables

- The first set command option changes the prefix of the substitution variable from an ampersand to another character. This character cannot be alphanumeric or white space.
- The second (ON option) and third (OFF option) control whether SQL*Plus will look for substitution variables or not.
- In addition, ON option changes the value of the *character* back to the ampersand.

Fundamentals of PL/SQL



Item Type	Capitalization	Example
Reserved word	Uppercase	BEGIN, DECLARE
Built-in function	Uppercase	COUNT, TO_DATE
Predefined data type	Uppercase	VARCHAR2, NUMBER
SQL command	Uppercase	SELECT, INSERT
Database object	Lowercase	student, f_id
Variable name	Lowercase	current_s_id, current_f_last

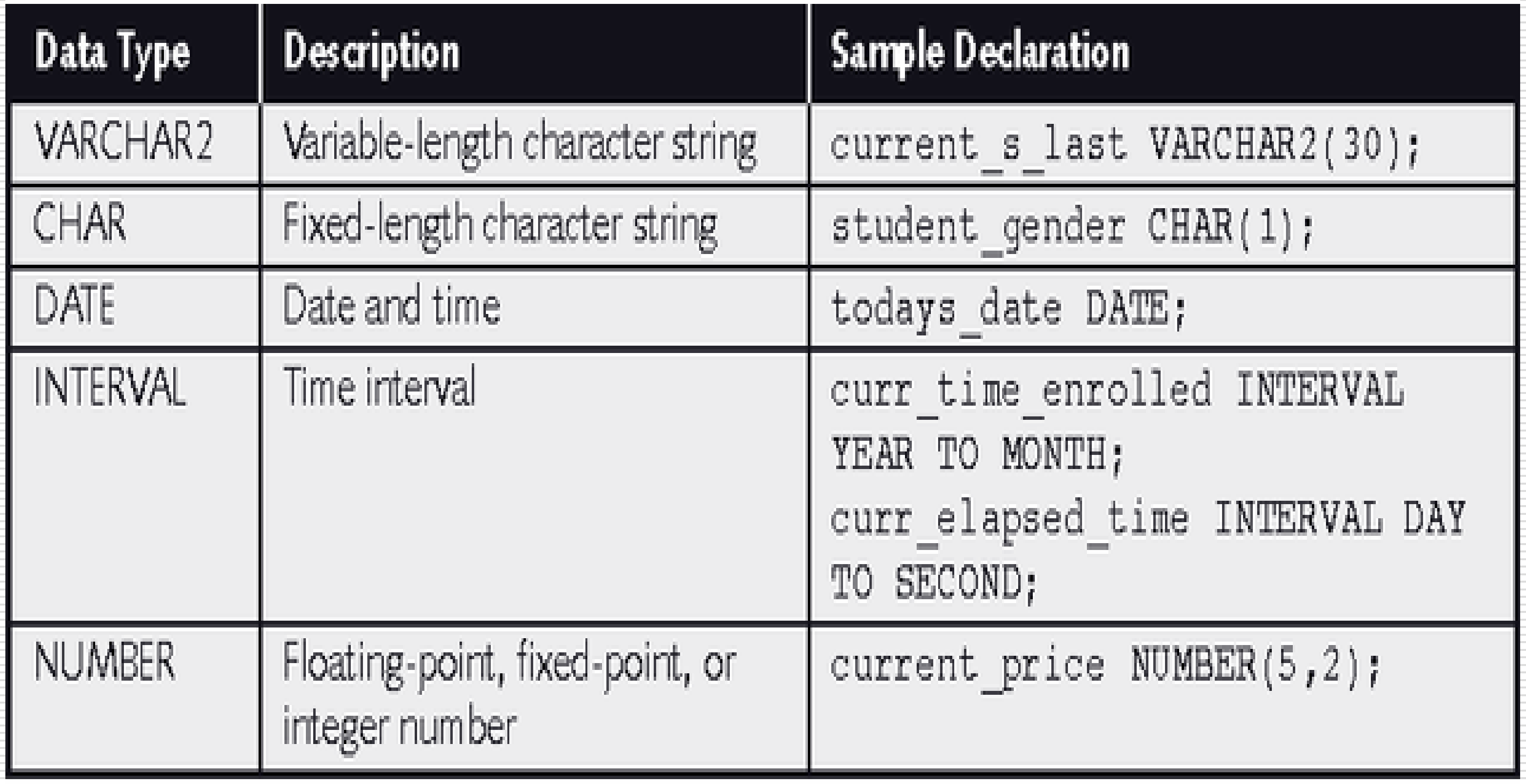
Table 4-1 PL/SQL command capitalization styles

Variables and Data Types

- Variables
  - Used to store numbers, character strings, dates, and other data values
  - Avoid using keywords, table names and column names as variable names
  - Must be declared with data type before use: *variable_name data_type_declaration*;

Scalar Data Types

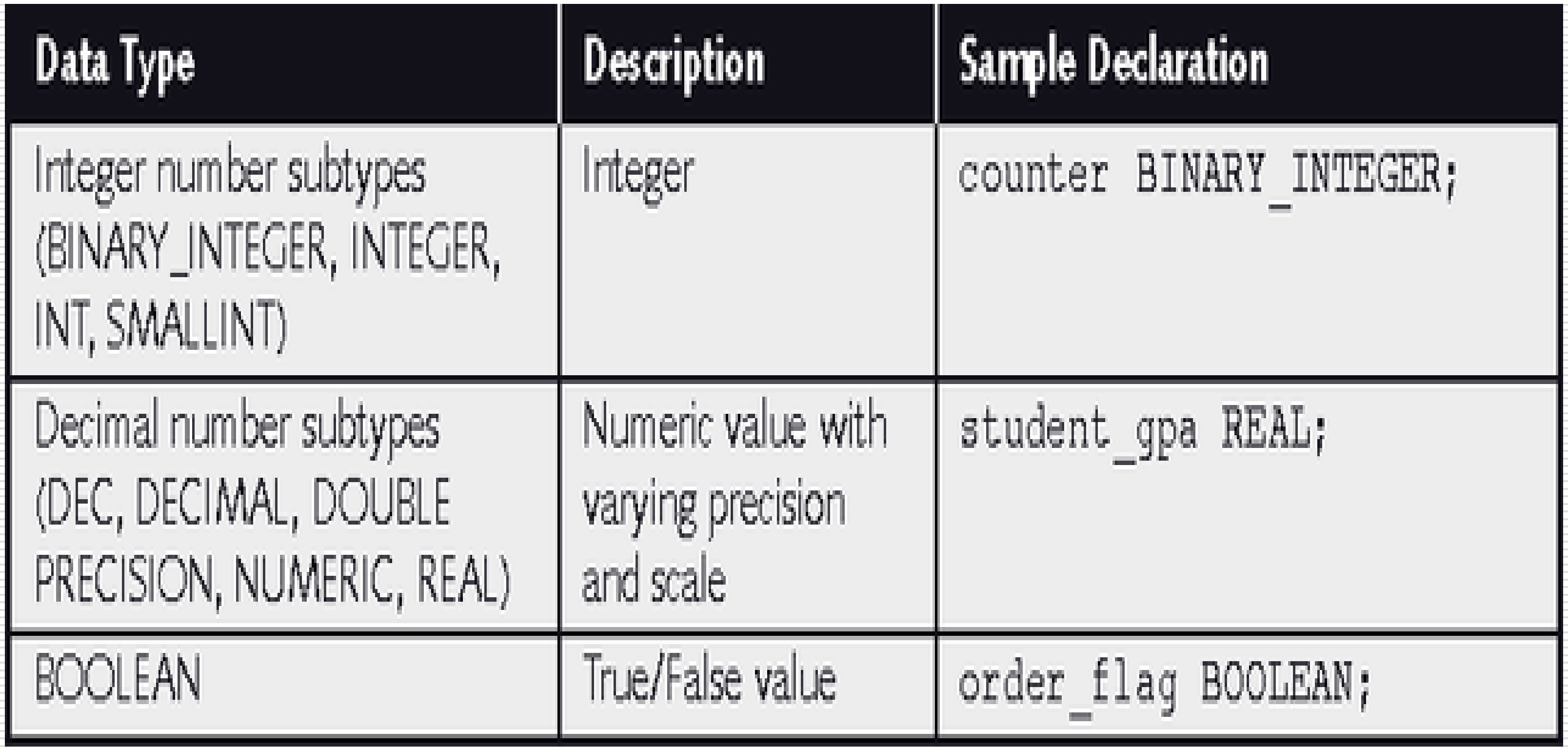
- Represent a single value



Data Type	Description	Sample Declaration
VARCHAR2	Variable-length character string	`current_s_last VARCHAR2(30);`
CHAR	Fixed-length character string	`student_gender CHAR(1);`
DATE	Date and time	`todays_date DATE;`
INTERVAL	Time interval	`curr_time_enrolled INTERVAL YEAR TO MONTH;` `curr_elapsed_time INTERVAL DAY TO SECOND;`
NUMBER	Floating-point, fixed-point, or integer number	`current_price NUMBER(5,2);`

Table 4-2 Scalar database data types

Scalar Data Types



Data Type	Description	Sample Declaration
Integer number subtypes (BINARY_INTEGER, INTEGER, INT, SMALLINT)	Integer	`counter BINARY_INTEGER;`
Decimal number subtypes (DEC, DECIMAL, DOUBLE PRECISION, NUMERIC, REAL)	Numeric value with varying precision and scale	`student_gpa REAL;`
BOOLEAN	True/False value	`order_flag BOOLEAN;`

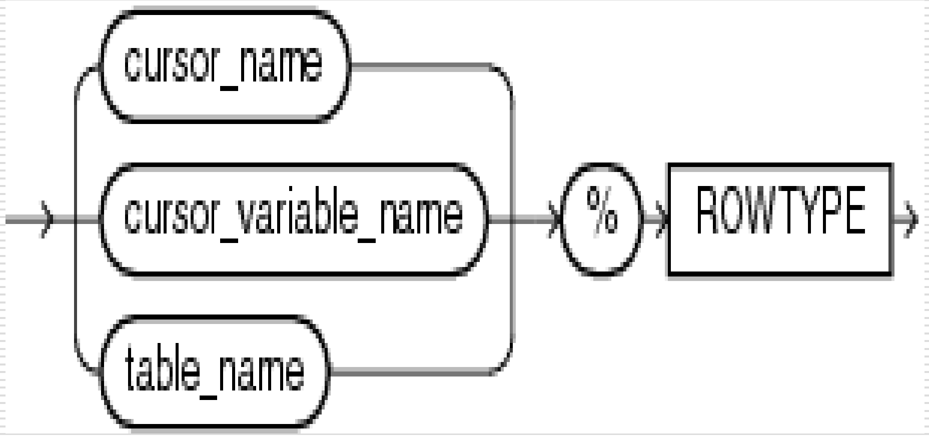
Table 4-3 General scalar data types

Composite and Reference Variables

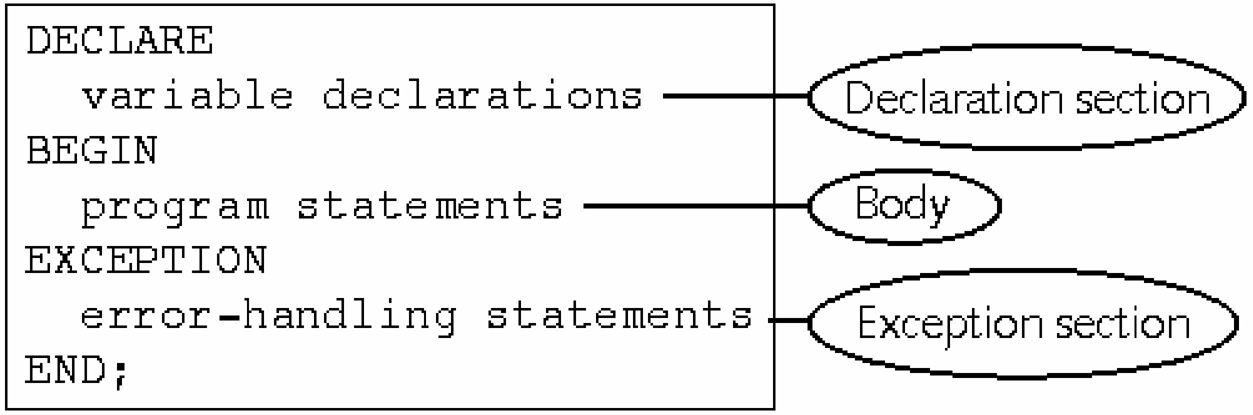
- Composite variables
  - RECORD: contains multiple scalar values, similar to a table record
  - TABLE: tabular structure with multiple columns and rows
  - VARRAY: variable-sized array
- Reference variables
  - Directly reference a specific database field or record and assume the data type of the associated field or record
  - %TYPE: same data type as a database field

Composite and Reference Variables

- %ROWTYPE: same data type as a database record



PL/SQL Program Blocks

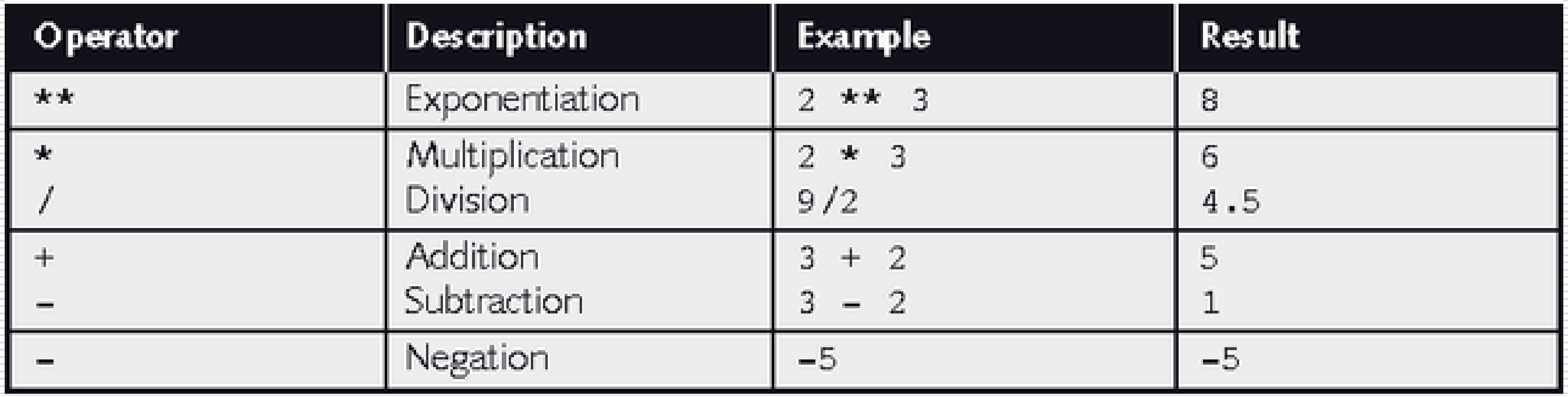


```
DECLARE
   variable declarations
BEGIN
   program statements
EXCEPTION
   error-handling statements
END;
```

Figure 4-1 Structure of a PL/SQL program block

- Comments:
  - Not executed by interpreter
  - Enclosed between /* and */
  - On one line beginning with --

Arithmetic Operators



Operator	Description	Example	Result
**	Exponentiation	2 ** 3	8
*	Multiplication	2 * 3	6
/	Division	9/2	4.5
+	Addition	3 + 2	5
-	Subtraction	3 - 2	1
-	Negation	-5	-5

Table 4-5 PL/SQL arithmetic operators in describing order of precedence

Assignment Statements

- Assigns a value to a variable

  variable_name := *value*;

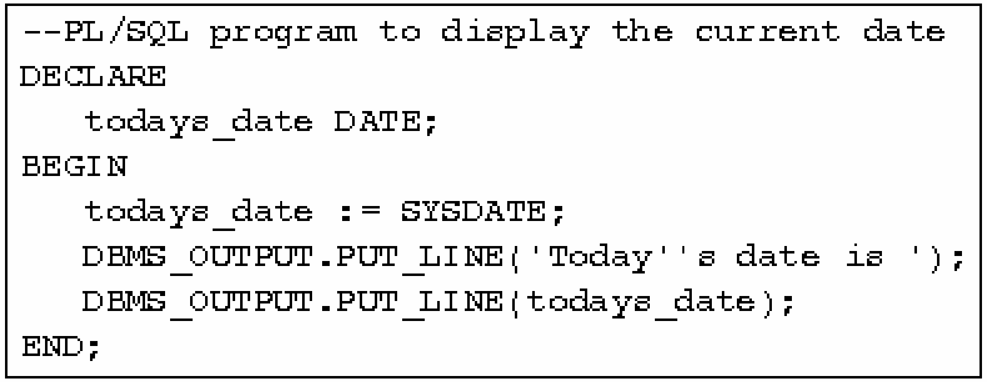
- Value can be a literal:

```
current_s_first_name := 'John';
```

- Value can be another variable:

```
current_s_first_name := s_first_name;
```

Executing a PL/SQL Program in SQL*Plus

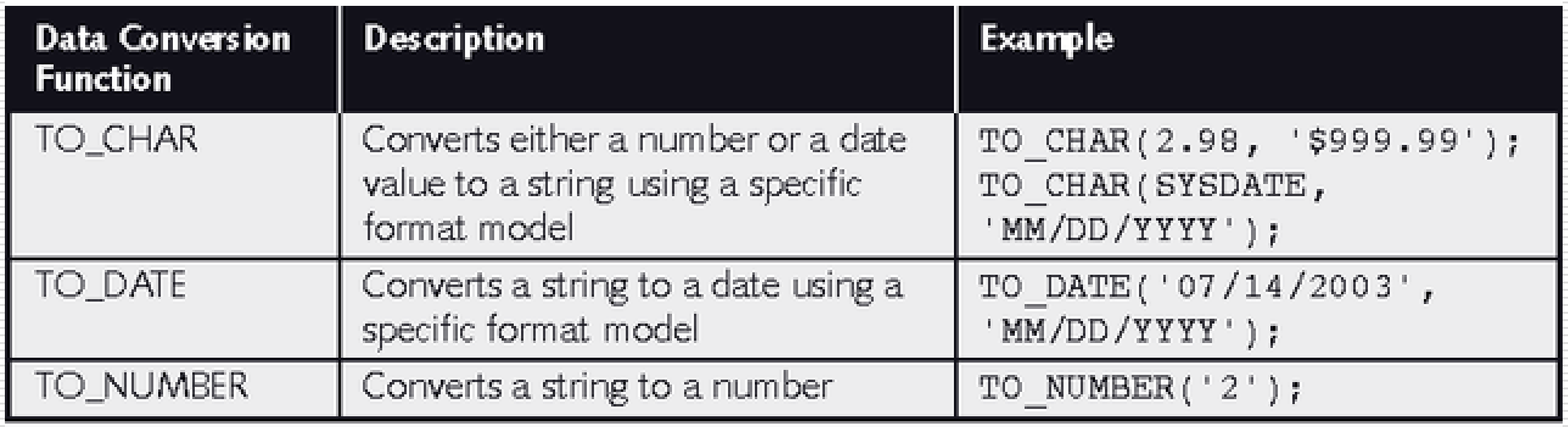


```
--PL/SQL program to display the current date
DECLARE
   todays_date DATE;
BEGIN
   todays_date := SYSDATE;
   DBMS_OUTPUT.PUT_LINE('Today''s date is ');
   DBMS_OUTPUT.PUT_LINE(todays_date);
END;
```

Figure 4-2 PL/SQL program commands

- Create program in text editor
- Paste into SQL*Plus window
- Press Enter, type / then enter to execute

PL/SQL Data Conversion Functions



Data Conversion Function	Description	Example
TO_CHAR	Converts either a number or a date value to a string using a specific format model	`TO_CHAR(2.98, '$999.99');` `TO_CHAR(SYSDATE, 'MM/DD/YYYY');`
TO_DATE	Converts a string to a date using a specific format model	`TO_DATE('07/14/2003', 'MM/DD/YYYY');`
TO_NUMBER	Converts a string to a number	`TO_NUMBER('2');`

Table 4-6 PL/SQL data conversion functions

Manipulating Character Strings with PL/SQL

- To concatenate two strings in PL/SQL, you use the double bar (||) operator:

  new_string := string1 || string2;
- To remove blank leading spaces use the LTRIM function:

  string := LTRIM(string_variable_name);
- To remove blank trailing spaces use the RTRIM function:

  string := RTRIM(string_variable_name);
- To find the number of characters in a character string use the LENGTH function:

  string_length := LENGTH(string_variable_name);

Manipulating Character Strings with PL/SQL

- To change case, use UPPER, LOWER, INITCAP
- INSTR function searches a string for a specific substring:

  start_position := INSTR(original_string, substring);
- SUBSTR function extracts a specific number of characters from a character string, starting at a given point:

  extracted_string := SUBSTR(string_variable, starting_point, number_of_characters);

PL/SQL Decision Control Structures

- Use IF/THEN structure to execute code if condition is true
  - IF *condition* THEN

    commands that execute if condition is TRUE;

    END IF;
- If condition evaluates to NULL it is considered false
- Use IF/THEN/ELSE to execute code if condition is true or false
  - IF *condition* THEN

    commands that execute if condition is TRUE;

    ELSE

    commands that execute if condition is FALSE;

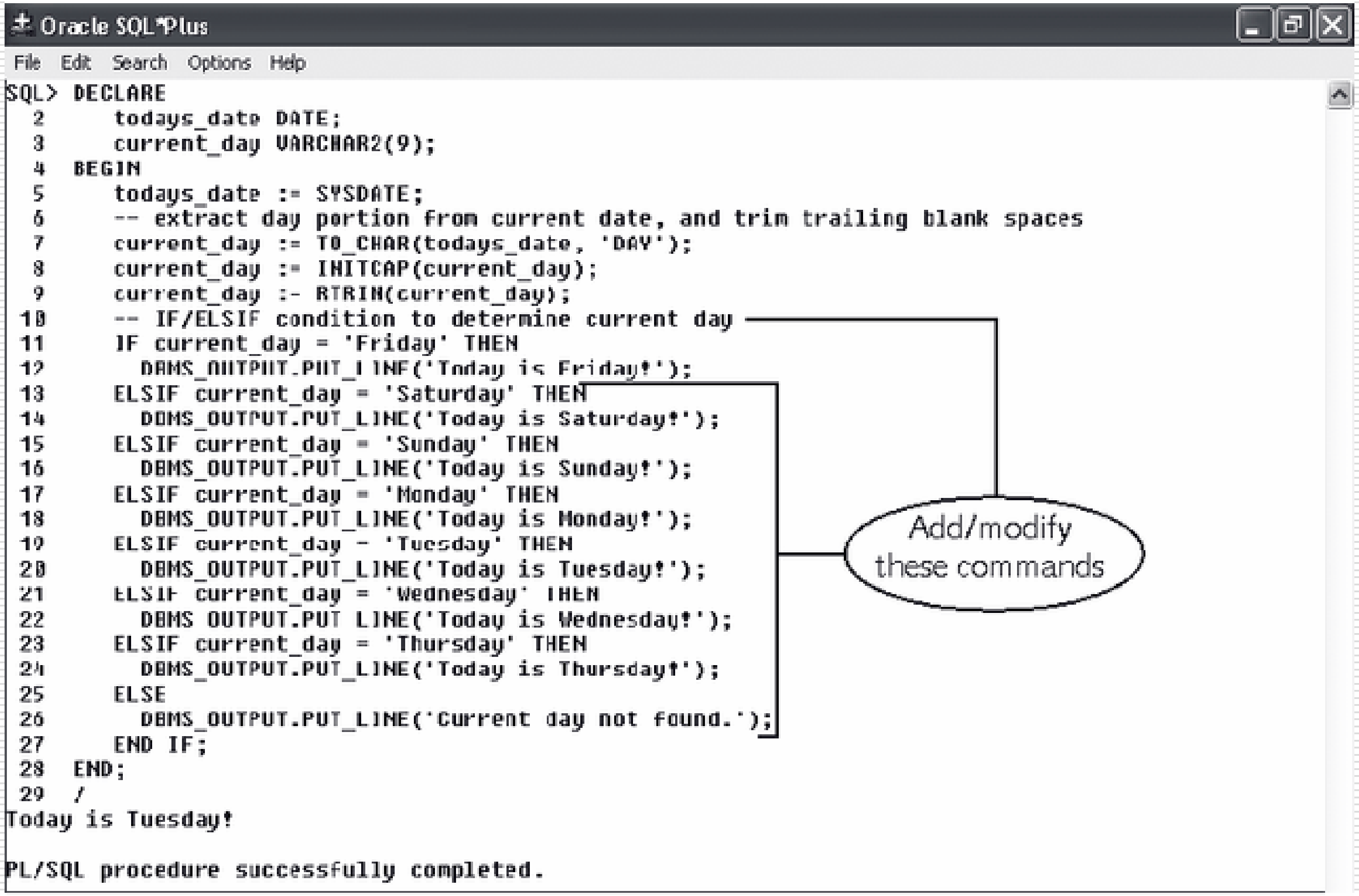
    END IF;
- Can be nested – be sure to end nested statements

PL/SQL Decision Control Structures

- Use IF/ELSIF to evaluate many conditions:
  - IF *condition1* THEN

```
    commands that execute if condition1 is TRUE;
   ELSIF condition2 THEN
    commands that execute if condition2 is TRUE;
   ELSIF condition3 THEN
commands that execute if condition3 is TRUE;
   ...
   ELSE
commands that execute if none of the
    conditions are TRUE;
   END IF;
```

IF/ELSIF Example



```
SQL> DECLARE
  2     todays_date DATE;
  3     current_day VARCHAR2(9);
  4  BEGIN
  5     todays_date := SYSDATE;
  6     -- extract day portion from current date, and trim trailing blank spaces
  7     current_day := TO_CHAR(todays_date, 'DAY');
  8     current_day := INITCAP(current_day);
  9     current_day := RTRIM(current_day);
 10     -- IF/ELSIF condition to determine current day
 11     IF current_day = 'Friday' THEN
 12       DBMS_OUTPUT.PUT_LINE('Today is Friday!');
 13     ELSIF current_day = 'Saturday' THEN
 14       DBMS_OUTPUT.PUT_LINE('Today is Saturday!');
 15     ELSIF current_day = 'Sunday' THEN
 16       DBMS_OUTPUT.PUT_LINE('Today is Sunday!');
 17     ELSIF current_day = 'Monday' THEN
 18       DBMS_OUTPUT.PUT_LINE('Today is Monday!');
 19     ELSIF current_day = 'Tuesday' THEN
 20       DBMS_OUTPUT.PUT_LINE('Today is Tuesday!');
 21     ELSIF current_day = 'Wednesday' THEN
 22       DBMS_OUTPUT.PUT_LINE('Today is Wednesday!');
 23     ELSIF current_day = 'Thursday' THEN
 24       DBMS_OUTPUT.PUT_LINE('Today is Thursday!');
 25     ELSE
 26       DBMS_OUTPUT.PUT_LINE('Current day not found.');
 27     END IF;
 28  END;
 29  /
Today is Tuesday!

PL/SQL procedure successfully completed.
```

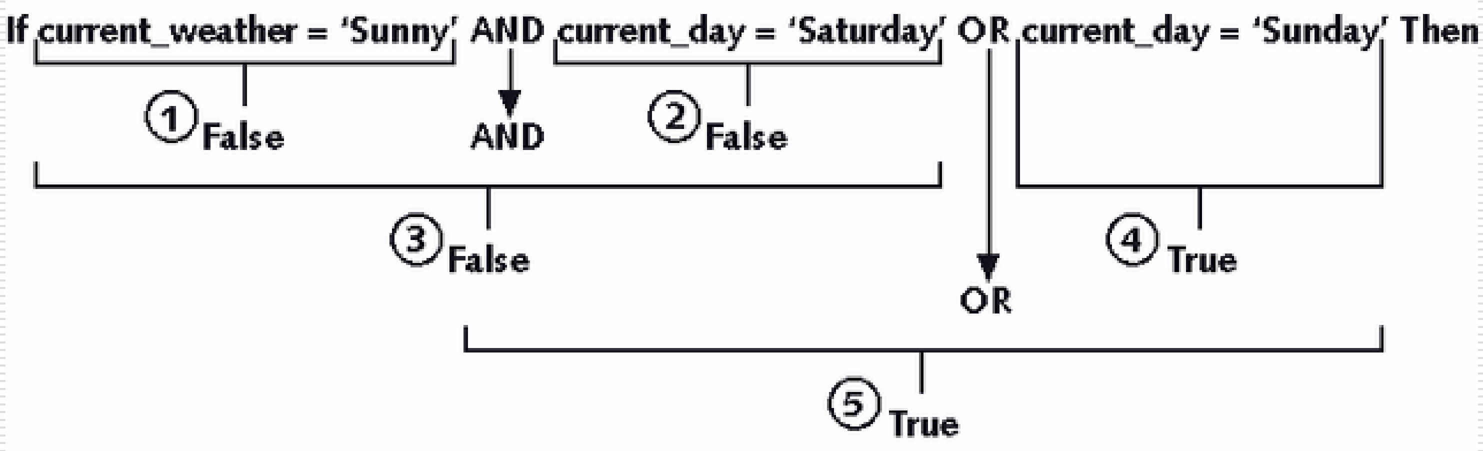
Figure 4-17 Using an IF/ELSIF structure

Example

```
declare
a number:=&num1;
b number:=&num2;
begin
if a>b then
dbms_output.put_line(a||'is greatest');
else
dbms_output.put_line(b||'is greatest');
end if;
end;
```

Complex Conditions

- Created with logical operators AND, OR and NOT
- AND is evaluated before OR
- Use () to set precedence



If current_weather = 'Sunny' AND current_day = 'Saturday' OR current_day = 'Sunday' Then
1 False
AND
2 False
3 False
OR
4 True
5 True

Figure 4-19 Evaluating AND and OR in an expression

Using SQL Queries in PL/SQL Programs

- Action queries can be used as in SQL*Plus
- May use variables in action queries
- DDL commands may not be used in PL/SQL

Loops

- Program structure that executes a series of program statements, and periodically evaluates an exit condition to determine if the loop should repeat or exit
- Pretest loop: evaluates the exit condition before any program commands execute
- Posttest loop: executes one or more program commands before the loop evaluates the exit condition for the first time
- PL/SQL has 5 loop structures

Simple loops

```
LOOP
STATEMENT 1;
STATEMENT 2;
….
STATEMENT N;
END LOOP;
```

The LOOP...EXIT Loop

```
LOOP
  [*program statements*]
  IF *condition* THEN
    EXIT;
  END IF;
  [*additional program statements*]
END LOOP
```

The LOOP...EXIT WHEN Loop

```
LOOP
    program statements
    EXIT WHEN condition;
END LOOP;
```

Example

```
declare
n number:=&limit;
f1 number:=0;
f2 number:=1;
f3 number;
begin
loop
f3:=f1+f2;
dbms_output.put_line(f3);
f1:=f2;
f2:=f3;
exit when f3>n;
end loop;
end;
```

The WHILE...LOOP

```
WHILE condition LOOP
    program statements
END LOOP;
```

Example

```
declare
n number:=&limit;
f1 number:=0;
f2 number:=1;
f3 number;
begin
dbms_output.put_line(f1);
dbms_output.put_line(f2);
f3:=f1+f2;
while f3<=n loop
dbms_output.put_line(f3);
f1:=f2;
f2:=f3;
f3:=f1+f2;
end loop;
end;
```

The Numeric FOR Loop

```
FOR counter_variable IN start_value ..
   end_value
LOOP
     program statements
END LOOP;
```

Example

```
declare
i number:=1;
j number:=2;
prod number(3):=1;
begin
for i in 1..10
loop
dbms_output.put_line(i||'*'||j||'='||i*j);
end loop;
end;
```

LOOP LABELS

- Loops can be labeled in the similar manner as PL/SQL blocks.

```
<<label_name>>
FOR LOOP_COUNTER IN
  LOWER_LIMIT..UPPER_LIMIT
LOOP
STATEMENT 1;
…
STATEMENT N;
END LOOP label_name;
```

LOOP LABELS

- The label must appear right before the beginning of the loop.
- This syntax example shows that the label can be optionally used at the end of the loop statement.
- It is very helpful to label nested loops because labels improve readability.

```
BEGIN
  <<outer_loop>>
  FOR i IN 1..3
  LOOP
        DBMS_OUTPUT.PUT_LINE('i = '||i);
        <<inner_loop>>
        FOR j IN 1..2
        LOOP
                DBMS_OUTPUT.PUT_LINE('j = '||j);
        END LOOP inner_loop;
  END LOOP outer_loop;
END;
```