

PL/SQL - Overview

Gagan Deep

Founder & Director


Rozy Computech Services

Kurukshetra. rozygag@yahoo.com

www.rozyph.com

What is PL/SQL ?

- Procedural Language – SQL
- An extension to SQL with design features of programming languages (procedural and object oriented)
- PL/SQL and Java are both supported as internal host languages within Oracle products.



The PL/SQL procedural language was developed by Oracle Corporation in the late 1980s as procedural extension language for SQL and the Oracle relational database. Following are notable facts about PL/SQL:

- PL/SQL is a completely portable, high-performance transaction-processing language.
- PL/SQL provides a built-in interpreted and OS independent programming environment.

- PL/SQL can also directly be called from the command-line SQL*Plus interface.
- Direct call can also be made from external programming language calls to database.
- PL/SQL's general syntax is based on that of ADA and Pascal programming language.
- Apart from Oracle, PL/SQL is available in TimesTen in-memory database and IBM DB2.

Why PL/SQL ?

- Acts as host language for stored procedures and triggers.
- Provides the ability to add middle tier business logic to client/server applications.
- Provides Portability of code from one environment to another
- Improves performance of multi-query transactions.
- Provides error handling

Features of PL/SQL


PL/SQL has the following features:


- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types.
- It offers a variety of programming structures.
- It supports structured programming through functions and procedures.
- It supports object oriented programming.
- It supports developing web applications and server pages.

Advantages of PL/SQL

PL/SQL has the following advantages :

- SQL is the standard database language and PL/SQL is strongly integrated with SQL. PL/SQL supports both **static** and **dynamic** SQL.
- **Static SQL** supports DML operations and transaction control from PL/SQL block.
- **Dynamic SQL** is SQL allows embedding DDL statements in PL/SQL blocks.

- 
- PL/SQL allows sending an entire block of statements to the database at one time. This reduces network traffic and provides high performance for the applications.
 - PL/SQL give high productivity to programmers as it can query, transform, and update data in a database.
 - PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented.

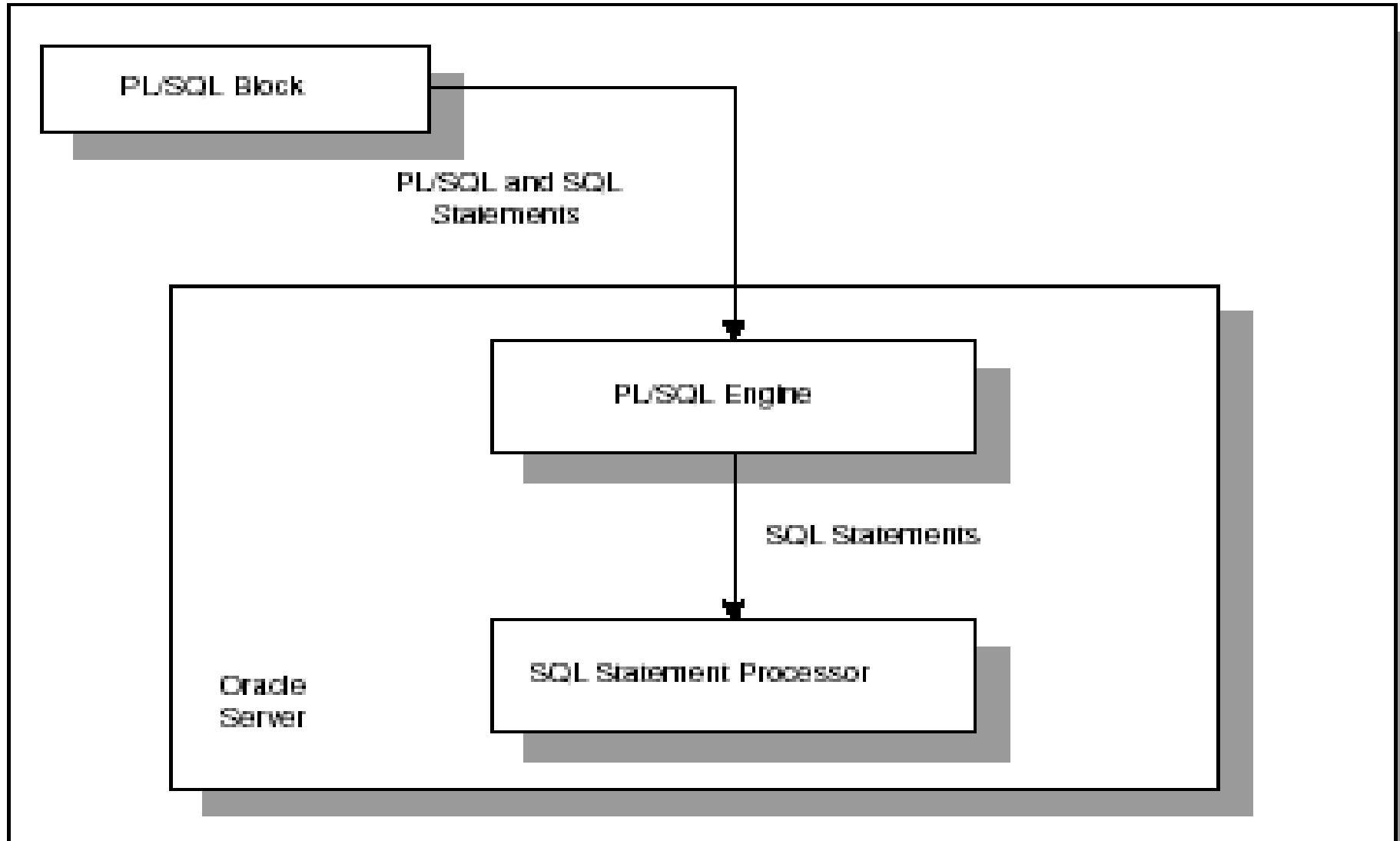
- 
- Applications written in PL/SQL are fully portable.
 - PL/SQL provides high security level.
 - PL/SQL provides access to predefined SQL packages.
 - PL/SQL provides support for Object-Oriented Programming.
 - PL/SQL provides support for Developing Web Applications and Server Pages.

Client-Server Architecture

- PL/SQL is a part of the Oracle RDBMS, and it can reside in two environments, the client and the server.
- Many Oracle applications are built using client-server architecture.
- The Oracle database resides on the server.
- The program(C, Java, or PL/SQL) that makes requests against this database resides on the client machine.

- 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 according to the Figure 1.

Figure 1 : The PL/SQL Engine and Oracle Server



- 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.

Comparison of 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 set 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.

Figure 2 : The PL/SQL in client – server architecture

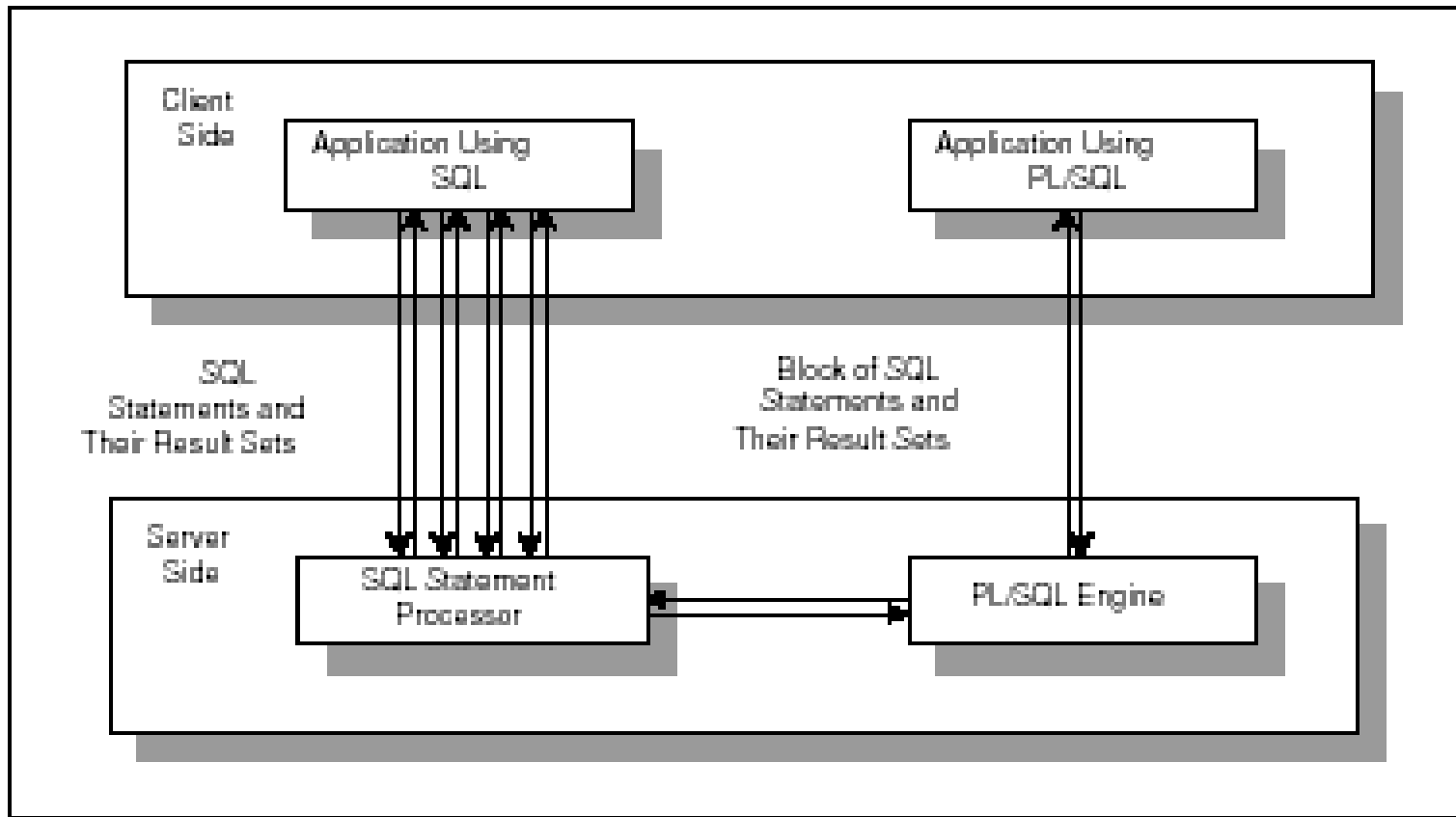


Figure 2.2 ■ PL/SQL in client-server architecture.

PL/SQL Block-Structure

PL/SQL is a block-structured language, meaning that PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub-parts:

1 Declarations :

This section starts with the keyword **DECLARE**. It is an **optional section** and defines all variables, cursors, subprograms, and other elements to be used in the program.

2 Executable Commands

This section is enclosed between the keywords **BEGIN** and **END** and it is a **mandatory section**. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a **NULL** command to indicate that nothing should be executed.

3 Exception Handling

This section starts with the keyword **EXCEPTION**. This section is again **optional** and **contains exception(s)** that handle errors in the program.

Every PL/SQL statement end with a semicolon (;).

PL/SQL blocks **can be nested** within other PL/SQL blocks using **BEGIN** and **END**.

PL/SQL Block Structure

DECLARE (optional)

- variable declarations

BEGIN (mandatory)

- SQL statements
- PL/SQL statements or sub-blocks

EXCEPTION (optional)

- actions to perform when errors occur

END; (mandatory)

PL/SQL Block Types

Anonymous

```
DECLARE  
BEGIN  
    -statements  
EXCEPTION  
END;
```

Procedure

```
PROCEDURE <name>  
IS  
BEGIN  
    -statements  
EXCEPTION  
END;
```

Function

```
FUNCTION <name>  
RETURN <datatype>  
IS  
BEGIN  
    -statements  
EXCEPTION  
END;
```

Example

```
DECLARE
message varchar2(20):= 'Hello, World!';
BEGIN
dbms_output.put_line(message);
END;
/
```

Hello, World !

PL/SQL procedure successfully completed.

The **end;** line signals the end of the PL/SQL block.

To run the code from SQL command line, you may need to type **/** at the beginning of the first blank line after the last line of the code.

The PL/SQL Identifiers

PL/SQL identifiers are constants, variables, exceptions, procedures, cursors, and reserved words. The identifiers consist of a letter optionally followed by more, letters numerals, dollar signs, underscores, and number signs and should not exceed 30 characters.

- By default, **identifiers are not case-sensitive**. So you can use **integer** or **INTEGER** to represent a numeric value. You cannot use a reserved keyword as an identifier.

The PL/SQL Delimiters

A delimiter is a symbol with a special meaning. Following is the list of delimiters in PL/SQL:

Delimiter	Description
+ , - , * , /	Addition, subtraction/negation, multiplication, division
%	Attribute indicator
'	Character string delimiter
.	Component selector
(,)	Expression or list delimiter
:	Host variable indicator
,	Item separator

Similarly, there are more delimiters

The PL/SQL Comments

- Program comments are explanatory statements that you can include in the PL/SQL code that you write and helps anyone reading it's source code. All programming languages allow for some form of comments.
- The PL/SQL supports **single line** and **multi-line comments**. All characters available inside any comment are ignored by PL/SQL compiler. The PL/SQL single-line comments start with the delimiter **--** (double hyphen) and **multi-line comments** are enclosed by **/*** and ***/**.

Example

```
DECLARE
```

```
-- variable declaration
```

```
message varchar2(20):= 'Hello, World!';
```

```
BEGIN
```

```
/*
```

```
PL/SQL executable statement(s)
```

```
*/
```

```
dbms_output.put_line(message);
```

```
END;
```

```
/
```

- After Execution in SQL the result is : **Hello, World!**
- PL/SQL procedure successfully completed.

Data Types

PL/SQL variables, constants and parameters must have a valid data types which specifies a storage format, constraints, and valid range of values. **PL/SQL is strongly typed.** The following is the list of data types:

Category	Description
Scalar	Single values with no internal components, such as a NUMBER, DATE, or BOOLEAN.
Large Object (LOB)	Pointers to large objects that are stored separately from other data items, such as text, graphic images, video clips, and sound waveforms.
Composite	Data items that have internal components that can be accessed individually. For example, collections and records.
Reference	Pointers to other data items.

Scalar Data Types

PL/SQL Scalar Data Types and Subtypes come under the following categories:

Date Type	Description
Numeric	Numeric values, on which arithmetic operations are performed.
Character	Alphanumeric values that represent single characters or strings of characters.
Boolean	Logical values, on which logical operations are performed.
Datetime	Dates and times.

PL/SQL Numeric Data Types

Data Type	Description
FLOAT	ANSI and IBM specific floating-point type with maximum precision of 126 binary digits (approximately 38 decimal digits)
INTEGER OR INT	ANSI and IBM specific integer type with maximum precision of 38 decimal digits
REAL	Floating-point type with maximum precision of 63 binary digits (approximately 18 decimal digits)
PLS_INTEGER OR BINARY_INTEGER	Signed integer in range -2,147,483,648 through 2,147,483,647, represented in 32 bits
DEC(prec, scale)	ANSI specific fixed-point type with maximum precision of 38 decimal digits.

Example

Following is a valid declaration:

```
DECLARE
```

```
    num1 INTEGER;
```

```
    num2 REAL;
```

```
    num3 DOUBLE PRECISION;
```

```
BEGIN
```

```
    null;
```

```
END;
```

```
/
```

When the above code is compiled and executed, it produces following result:

PL/SQL procedure successfully completed

Character Data Types

Data Type	Description
CHAR	Fixed-length character string with maximum size of 32,767 bytes
VARCHAR2	Variable-length character string with maximum size of 32,767 bytes
RAW	Variable-length binary or byte string with maximum size of 32,767 bytes, not interpreted by PL/SQL
LONG	Variable-length character string with maximum size of 32,760 bytes
ROWID	Physical row identifier, the address of a row in an ordinary table

PL/SQL Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. PL/SQL language is rich in built-in operators and provides following type of operators:

- Arithmetic operators
- Relational operators
- Comparison operators
- Logical operators
- String operators

Precedence of Operators

Operator	Operation
**	exponentiation
+, -	identity, negation
*, /	multiplication, division
+, -,	addition, subtraction, concat
=, <, >, <=, >=, IS NULL, LIKE, IN	Comparison
NOT	logical negation
AND	conjunction
OR	inclusion

Control Structures

- Programming languages provide various control structures that allow for more complicated execution paths.
- Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

PL/SQL Decisions Making

PL/SQL language provides following types of decision making statements.

- IF-THEN Statement
- IF-THEN-ELSE Statement
- IF-THEN-ELSEIF Statement
- Nested IF Statement

Syntax for IF-THEN statement

```
IF condition THEN  
    S;  
END IF;
```

Where *condition* is a Boolean or relational condition and *S* is a simple or compound statement.

Example of an IF-THEN statement is:

```
IF (a <= 20) THEN  
    C:= C+1;  
END IF;
```

DECLARE

Example

```
a number(2) := 10;  
BEGIN  
    a:= 10;  
    IF( a < 20 ) THEN  
dbms_output.put_line('a is less than 20' );  
    END IF;  
dbms_output.put_line('value of a is:' || a);  
END;
```

/
After Execution :

a is less than 20 value of a is : 10

PL/SQL procedure successfully completed.

PL/SQL - Loops

PL/SQL provides the following types of loop to handle the looping requirements

- BASIC LOOP
- WHILE LOOP
- FOR LOOP
- NESTED LOOP

Syntax of Basic Loop

The syntax of a basic loop in PL/SQL language is:

LOOP

Sequence of statements;

END LOOP

Example

```
DECLARE
x number := 10;
BEGIN LOOP
dbms_output.put_line(x);
x := x + 10;
IF x > 50 THEN exit; END IF;
END LOOP;
dbms_output.put_line('After Exit x is: ' || x);
END;
/
```

After Execution is : 10 20 30 40 50 After Exit x is: 60

Syntax of For Loop

```
FOR counter IN i_value .. f_value LOOP  
    sequence_of_statements;  
END LOOP;
```

Following are special characteristics of FOR loop:

The *i_value* and *f_value* of the loop variable or *counter* can be literals, variables, or expressions but must evaluate to numbers. Otherwise, PL/SQL raises the predefined exception VALUE_ERROR.

The *i_value* need not to be 1; however, the **loop counter increment (or decrement) must be 1.**

PL/SQL allows determine the loop range dynamically at run time.

Example

```
DECLARE
a number(2);
BEGIN
FOR a in 10 .. 15 LOOP
dbms_output.put_line('value of a: '
|| a);
END LOOP;
END;
/
```

Result is

value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15

Common PL/SQL String Functions

- CHR(ASCIIvalue)
- ASCII(string)
- LOWER(string)
- SUBSTR(string,start,substrlen)
- LTRIM(string)
- RTRIM(string)
- REPLACE(string, searchstring, replacestring)
- UPPER(string)
- INITCAP(string)
- LENGTH(string)

Common PL/SQL Numeric Functions

- ABS(value)
- ROUND(value, precision)
- MOD(value, divisor)
- SQRT(value)
- TRUNC(value, |precision|)
- LEAST(exp1, exp2...)
- GREATEST(exp1, exp2...)

When is PL/SQL handy

- When something is too complicated for SQL
- When conditional branching and looping are needed.

Thanks !

If you have any query please mail me at

rozygag@yahoo.com

www.rozyph.com