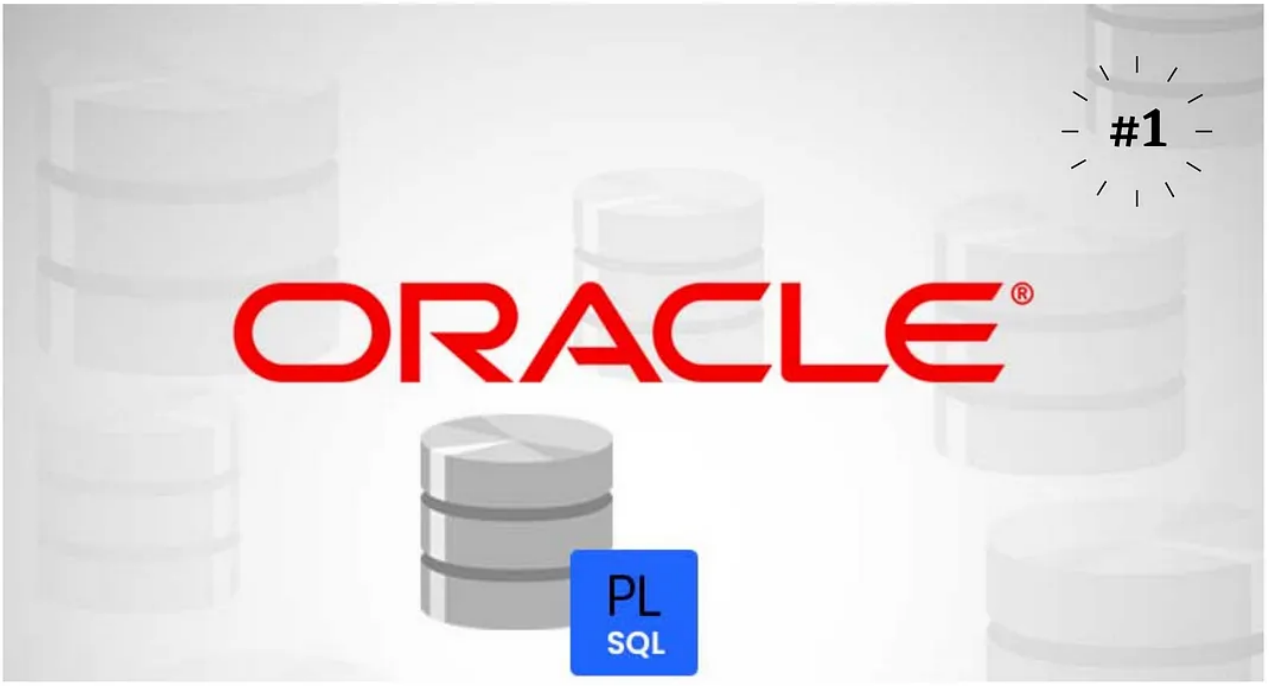
Introduction to Oracle PL/SQL: A Step-by-Step Guide for Beginners #1

Lakshitha Perera · Follow
6 min read · Apr 10, 2023



Introduction to Oracle PL/SQL

Oracle PL/SQL (Procedural Language/Structured Query Language) is a procedural extension of SQL designed specifically for the Oracle Database. It was developed by Oracle Corporation in the late 1980s to enhance the capabilities of SQL, allowing developers to write complex and efficient database applications.

PL/SQL supports various programming constructs, including:

1. **Variables and data types**: PL/SQL supports a wide range of data types, including scalar types (e.g., NUMBER, VARCHAR2, DATE), composite types (e.g., records and collections), and reference types (e.g., cursors).
2. **Control structures**: PL/SQL provides various control structures for conditional execution (IF, CASE) and looping (LOOP, WHILE, FOR).
3. **Procedures and functions**: These are reusable program units that can be called by other PL/SQL blocks or applications. Functions return a value, while procedures do not.
4. **Packages**: Packages are a way to group related procedures, functions, variables, and

other PL/SQL constructs into a single, organized unit.

5. **Triggers**: Triggers are special types of stored procedures that are automatically executed (or fired) in response to specific events in the database, such as inserting, updating, or deleting rows in a table.

6. **Cursors**: Cursors are used to retrieve and manipulate data from result sets returned by SELECT statements.

7. **Exception handling**: PL/SQL provides a robust exception handling mechanism to manage errors and exceptions that may occur during program execution.

PL/SQL is widely used by Oracle Database developers to create and manage database applications, perform data manipulation and validation, and implement business logic in a secure and efficient manner. Its tight integration with SQL and the Oracle Database makes it an essential tool for anyone working with Oracle-based systems.

Setting up the Oracle environment

Setting up the Oracle environment involves installing the Oracle Database software, setting up a development tool, and establishing a connection to the database. Here's a step-by-step guide to help you set up the Oracle environment:

1. Install Oracle Database software: You can download it from the Oracle website:https://www.oracle.com/database/technologies/xe-downloads.html

2. Install a development tool: Download Oracle SQL Developer from the following link: https://www.oracle.com/tools/downloads/sqldev-downloads.html

3. Connect to the database: To connect to the Oracle Database, you need to create a connection in your development tool. In Oracle SQL Developer, follow these steps:

a. Launch Oracle SQL Developer and click on the "New Connection" button (green "+" icon) in the "Connections" panel.

b. In the "New/Select Database Connection" dialog, provide the following information:

- Connection Name: Enter a descriptive name for the connection.
- Username: Enter the username for the database account you want to connect with (e.g., "system" or "sys" for administrative accounts, or a custom user you created during the installation process).
- Password: Enter the password for the database account.
- Hostname: Enter the hostname or IP address of the machine where the Oracle Database is installed (use "localhost" if it's installed on your local machine).
- Port: Enter the port number on which the Oracle Database is listening (the default port

is 1521).

- SID or Service Name: Enter the SID (System Identifier) or Service Name of the Oracle Database instance (e.g., “XE” for Express Edition or “orcl” for other editions).

c. Click the “Test” button to test the connection. If the connection is successful, you will see a “Success” message. If not, double-check the connection parameters and ensure that the Oracle Database service is running.

d. Click the “Connect” button to establish the connection. The new connection will appear in the “Connections” panel.

Now that you have set up the Oracle environment, you can start writing and executing SQL and PL/SQL code using your development tool.

PL/SQL fundamentals

PL/SQL block structure

A PL/SQL block consists of the following sections: DECLARE, BEGIN, EXCEPTION, and END. The DECLARE section (optional) is for declaring variables and other objects. The BEGIN section contains executable code, the EXCEPTION section (optional) handles errors, and the block concludes with the END keyword.

```
DECLARE
  v_name VARCHAR2(100);
BEGIN
  SELECT first_name || ' ' || last_name
  INTO v_name
  FROM employees
  WHERE employee_id = 100;

  DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_name);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Employee not found');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An error occurred');
END;
```

Variables and Data Types

PL/SQL supports various data types, including scalar types (e.g., NUMBER, VARCHAR2, DATE), composite types (e.g., records and collections), and reference types (e.g., cursors). Variables are declared in the DECLARE section of a PL/SQL block.

```
DECLARE
  v_age NUMBER := 30;
  v_name VARCHAR2(50) := 'John Doe';
```

```
  v_join_date DATE := SYSDATE;
BEGIN
  -- Your PL/SQL code here
END;
```

Control Structures

PL/SQL provides control structures for conditional execution and loops, such as IF, CASE, LOOP, WHILE, and FOR.

IF statement

```
DECLARE
  v_age NUMBER := 25;
BEGIN
  IF v_age >= 18 THEN
    DBMS_OUTPUT.PUT_LINE('You are eligible to vote.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('You are not eligible to vote.');
  END IF;
END;
```

CASE statement

```
DECLARE
  v_grade CHAR(1) := 'A';
BEGIN
  CASE v_grade
    WHEN 'A' THEN
      DBMS_OUTPUT.PUT_LINE('Excellent');
    WHEN 'B' THEN
      DBMS_OUTPUT.PUT_LINE('Good');
    WHEN 'C' THEN
      DBMS_OUTPUT.PUT_LINE('Average');
    ELSE
      DBMS_OUTPUT.PUT_LINE('Invalid grade');
  END CASE;
END;
```

LOOP

```
DECLARE
  v_counter NUMBER := 1;
BEGIN
  LOOP
    EXIT WHEN v_counter > 5;
    DBMS_OUTPUT.PUT_LINE('Counter: ' || v_counter);
```

```
    v_counter := v_counter + 1;
  END LOOP;
END;
```

WHILE loop

```
DECLARE
  v_counter NUMBER := 1;
BEGIN
  WHILE v_counter <= 5 LOOP
    DBMS_OUTPUT.PUT_LINE('Counter: ' || v_counter);
    v_counter := v_counter + 1;
  END LOOP;
END;
```

FOR loop

```
BEGIN
  FOR v_counter IN 1..5 LOOP
    DBMS_OUTPUT.PUT_LINE('Counter: ' || v_counter);
  END LOOP;
END;
```

Working with procedures and functions

Introduction to stored procedures and functions

Stored procedures and functions are reusable program units that encapsulate a series of PL/SQL statements for a specific task. They are stored in the database and can be invoked by applications or other PL/SQL blocks. Stored procedures and functions help improve the organization, maintainability, and efficiency of your Oracle Database applications.

The key difference between procedures and functions is that procedures do not return a value, whereas functions return a single value. Both can take parameters as input and/or output.

Creating and executing procedures and functions

To create a procedure, use the CREATE PROCEDURE statement followed by the procedure name, an optional parameter list, and the procedure body enclosed in the BEGIN and END keywords.

Example of creating a procedure:

```
CREATE PROCEDURE add_employee(p_first_name VARCHAR2, p_last_name VARCHAR2, p_age NUMBE
```

```
BEGIN
  -- Add employee logic here
  DBMS_OUTPUT.PUT_LINE('Employee added: ' || p_first_name || ' ' || p_last_name);
END add_employee;
```

To call a procedure, use the procedure name followed by the parameter list enclosed in parentheses.

Example of executing a procedure:

```
BEGIN
  add_employee('John', 'Doe', 30);
END;
```

To create a function, use the CREATE FUNCTION statement followed by the function name, an optional parameter list, the RETURN keyword with the return data type, and the function body enclosed in the BEGIN and END keywords.

Example of creating a function:

```
CREATE FUNCTION calculate_age(p_birth_date DATE) RETURN NUMBER AS
  v_age NUMBER;
BEGIN
  v_age := FLOOR(MONTHS_BETWEEN(SYSDATE, p_birth_date) / 12);
  RETURN v_age;
END calculate_age;
```

To call a function, use the function name with the parameter list enclosed in parentheses in an expression or assignment.

Example of executing a function:

```
DECLARE
  v_birth_date DATE := TO_DATE('1990-01-01', 'YYYY-MM-DD');
  v_age NUMBER;
BEGIN
  v_age := calculate_age(v_birth_date);
  DBMS_OUTPUT.PUT_LINE('Age: ' || v_age);
END;
```

Parameters: IN, OUT, and IN OUT

Parameters in procedures and functions can have three modes: IN, OUT, and IN OUT.

- **IN**: The default mode, where the parameter is used for input only. The calling program provides a value, and the procedure/function cannot modify it.
- **OUT**: The parameter is used for output only. The procedure/function sets the value, and the calling program can use it after the procedure/function execution.
- **IN OUT**: The parameter is used for both input and output. The calling program provides an initial value, and the procedure/function can modify it.

Example with IN, OUT, and IN OUT parameters:

```
CREATE PROCEDURE get_employee_info(
  p_employee_id IN NUMBER,
  p_first_name OUT VARCHAR2,
  p_last_name OUT VARCHAR2,
  p_age IN OUT NUMBER
) AS
BEGIN
  SELECT first_name, last_name
  INTO p_first_name, p_last_name
  FROM employees
  WHERE employee_id = p_employee_id;

  -- Calculate age based on hire_date (for demonstration purposes)
  SELECT FLOOR(MONTHS_BETWEEN(SYSDATE, hire_date) / 12)
  INTO p_age
  FROM employees
  WHERE employee_id = p_employee_id;
END get_employee_info;
```

Example of calling the procedure with IN, OUT, and IN OUT parameters:

```
DECLARE
  v_employee_id NUMBER := 100;
  v_first_name VARCHAR2(50);
  v_last_name VARCHAR2(50);
  v_age NUMBER := 0;
BEGIN
  get_employee_info(v_employee_id, v_first_name, v_last_name, v_age);
  DBMS_OUTPUT.PUT_LINE('Employee: ' || v_first_name || ' ' || v_last_name);
  DBMS_OUTPUT.PUT_LINE('Age: ' || v_age);
END;
```

In this example, the `get_employee_info` the procedure is called with the following parameters:

- `v_employee_id` : IN parameter (input only) to provide the employee ID.
- `v_first_name` : OUT parameter (output only) to receive the employee's first name.
- `v_last_name` : OUT parameter (output only) to receive the employee's last name.
- `v_age` : IN OUT parameter (input and output) to provide an initial age value and receive the updated age value after the procedure execution.

After calling the procedure, the values of the OUT and IN OUT parameters are printed using the `DBMS_OUTPUT.PUT_LINE` function.

Conclusion

In this article, we introduced Oracle PL/SQL. We also guided you through setting up the Oracle environment and covered the fundamentals of PL/SQL, including working with procedures and functions. This foundation prepares you for developing efficient Oracle Database applications.

Stay tuned for upcoming articles on advanced topics, and feel free to reach out with questions or feedback. Happy coding, and see you in the next article!

> *Master Coding: Read, Analyze, Implement, Repeat*