Oracle PL/SQL Tutorial for Beginners

Introduction

Having trouble going beyond simple queries with SQL and implementing logic, loops, and flow control into your database queries? Many beginning programmers often find this leap into procedural programming to be daunting.

The aim of this Oracle PL/SQL tutorial is to provide a structured, step-by-step introduction to blocks, variables, and error handling. You will learn how you can create robust, efficient stored programs directly in Oracle. Click here to see our full **Oracle PL/SQL course syllabus** and start with procedural programming right away!

Why Students or Freshers Learn Oracle PL/SQL?

Learning Oracle PL/SQL is important for freshers aspiring for specialized database and development roles.

- **Procedural Power:** It enables you to incorporate complex business rules and logic, such as loops and conditions, right into the database for efficiency and security.
- **Core Developer Role:** Proficiency is required for Oracle Developer, PL/SQL Programmer, and Database Engineer roles in an enterprise setup.
- **Performance and Security:** PL/SQL allows developers to build Stored Procedures and Functions, thereby centralizing the code, improving query performance, and enhancing the security of the data.
- **Integration with SQL:** It integrates procedural programming with the high-speed data manipulation features of SQL in a unique and powerful way.

Ready to Ace Your Interview? Click here for Top **Oracle PL/SQL Interview Questions and Answers** for Freshers!

Step-by-Step Oracle PL/SQL Tutorial for Beginners

Welcome to your comprehensive, step-by-step tutorial focused on mastering Oracle PL/SQL, or Procedural Language/SQL! The guide shall be useful for beginners, bridging the gap between doing simple SQL queries and doing powerful programming of databases.

We'll cover everything from basic programming block structure to creating reusable stored procedures, using practical code examples to build your confidence.

Step 1. Setting up Your PL/SQL Environment

PL/SQL code runs directly on the Oracle Database server. You want to have the database installed and a client tool where you can send the code. We will use in this book the setup from the SQL tutorial.

1.1 Prerequisites: Oracle Database and SQL Developer

- **Oracle Database XE:** You must have Oracle Database XE up and running. It will provide the environment for the compilation and execution of PL/SQL code.
- **SQL Developer:** Connect to your XE instance using SQL Developer, or another client like SQL*Plus. You should connect as a user – such as SYSTEM, or a user created for you to practice with, that has the required privileges.

1.2 Allowing Output (Essential to Learning)

PL/SQL has the *DBMS_OUTPUT.PUT_LINE* function that displays a message for debugging and testing. Most of the client tools suppress this output by default.

- **In SQL Developer:** Before executing any PL/SQL block, execute the following command in your SQL Worksheet:

SET SERVEROUTPUT ON;

This command only needs to be executed once per session to enable output display.

Step 2: The Anatomy of a PL/SQL Block

The block is the basic unit of PL/SQL programming. Each anonymous (unnamed) program and every stored procedure or function is built around the following three main sections:

2.1 The Anonymous Block Structure

[DECLARE

— Variable, Constant, and Cursor declarations go here.]

BEGIN

— Executable statements (SQL and PL/SQL code) go here.

— This section is mandatory.

[EXCEPTION

— Error handling code goes here.]

END;

/

The / (slash) on a new line instructs the client tool (such as SQL Developer) to execute the whole preceding PL/SQL block.

2.2 Using Variables and Printing Output

The DECLARE section is used to define memory space for data that your program will be using.

DECLARE

— Declare variables with data types

v_message VARCHAR2(100) := ‘Hello, PL/SQL World!’;

v_current_date DATE := SYSDATE;

c_pi CONSTANT NUMBER(3, 2) := 3.14; — Define a constant

BEGIN

```
    — Use DBMS_OUTPUT to display the variable values
    DBMS_OUTPUT.PUT_LINE('1. The message is: ' || v_message);
    DBMS_OUTPUT.PUT_LINE('2. Today is: ' || v_current_date);
    DBMS_OUTPUT.PUT_LINE('3. Pi constant: ' || c_pi);
END;
/
```

- **: = :** Assignment operator used to give a variable an initial value.
- **||:** Concatenation operator used to join strings and variable values.

Step 3: Integrating SQL and PL/SQL

The power of PL/SQL comes because it can execute standard SQL commands to operate on database tables and use procedural logic to control the flow.

3.1 Fetching Data into a Variable (SELECT INTO)

You have to use the SELECT INTO command to bring a single value from a database table into a PL/SQL variable.

Assume an EMPLOYEES table with employee_id and salary in it.

```
DECLARE
    v_employee_id EMPLOYEES.employee_id%TYPE := 1001; — Anchored data type
    v_current_salary EMPLOYEES.salary%TYPE;
    v_tax_amount NUMBER(8, 2);
```

```
BEGIN
    — Fetch the salary for the specified employee ID into the variable
    SELECT salary
    INTO v_current_salary
    FROM EMPLOYEES
    WHERE employee_id = v_employee_id;
    — Perform calculation
    v_tax_amount := v_current_salary * 0.20;
    — Display the result
    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_employee_id);
    DBMS_OUTPUT.PUT_LINE('Salary: ' || v_current_salary);
    DBMS_OUTPUT.PUT_LINE('Estimated Tax (20%): ' || v_tax_amount);
END;
/
```

%TYPE Anchored Data Type: This is highly recommended. It tells PL/SQL to use the exact data type of the specified table column, protecting your code from schema changes.

3.2 Data Modification (DML)

You can execute DML statements like INSERT, UPDATE and DELETE directly within the BEGIN block.

```
DECLARE
    v_employee_id EMPLOYEES.employee_id%TYPE := 1001;
    v_raise_percent NUMBER := 0.05; — 5% raise
BEGIN
    — Update the employee's salary
    UPDATE EMPLOYEES
    SET salary = salary * (1 + v_raise_percent)
    WHERE employee_id = v_employee_id;
    — Check how many rows were affected
    DBMS_OUTPUT.PUT_LINE('Rows updated: ' || SQL%ROWCOUNT);
    — Finalize the change
    COMMIT;
END;
/
```

SQL%ROWCOUNT: System attribute; returns the number of rows affected by the most recently executed DML statement.

Step 4: Control Structures (Logic)

Control structures enable your program to make choices and to execute code repeatedly.

4.1 Conditional Logic (IF-THEN-ELSIF-ELSE)

This structure allows various code paths based on conditions.

```
DECLARE
    v_total_sales NUMBER := 15000;
    v_commission NUMBER := 0;
BEGIN
    IF v_total_sales > 20000 THEN
        v_commission := v_total_sales * 0.10; — 10%
        DBMS_OUTPUT.PUT_LINE('Goal achieved! Commission: ' || v_commission);
    ELSIF v_total_sales > 10000 THEN
        v_commission := v_total_sales * 0.05; — 5%
        DBMS_OUTPUT.PUT_LINE('Decent sales. Commission: ' || v_commission);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Sales target missed. No commission.');
    END IF;
```

```
END;

/
```

4.2 Looping Structures

Loops are used to repeat a block of code multiple times.

Simple Loop (Requires EXIT WHEN)

```
DECLARE

   v_counter NUMBER := 1;

BEGIN

   LOOP

      DBMS_OUTPUT.PUT_LINE('Loop iteration: ' || v_counter);

      v_counter := v_counter + 1;

      EXIT WHEN v_counter > 5; — Exit condition

   END LOOP;

END;

/
```

FOR Loop Numeric

Ideal for executing a code block a specified number of times.

```
BEGIN
```

```
    -- Loop from 1 to 3 (i is implicitly declared and managed by the loop)
    FOR i IN 1..3 LOOP
        DBMS_OUTPUT.PUT_LINE('FOR Loop count: ' || i);
    END LOOP;
END;
/
```

Step 5: Handling Errors (EXCEPTION)

The EXCEPTION section forms the backbone of a good PL/SQL code. Here, you catch runtime errors gracefully and prevent your program from crashing.

5.1 Common Exceptions

Exception Name	When It Occurs
NO_DATA_FOUND	A SELECT INTO returns no rows.
TOO_MANY_ROWS	A SELECT INTO returns more than one row.
OTHERS	Catches any named or unhandled Oracle error.

5.2 Example Error Handling

```
DECLARE
```

```
    v_emp_id EMPLOYEES.employee_id%TYPE := 9999; — This ID likely doesn't exist
    v_salary EMPLOYEES.salary%TYPE;
BEGIN
    — This SELECT will fail with NO_DATA_FOUND
    SELECT salary INTO v_salary
    FROM EMPLOYEES
    WHERE employee_id = v_emp_id;
    DBMS_OUTPUT.PUT_LINE('Employee salary: ' || v_salary); — This line won't run
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('ERROR: Employee ID ' || v_emp_id || ' was not found.');
    WHEN OTHERS THEN
        — OTHERS is the general catch-all; SQLCODE and SQLERRM give details
        DBMS_OUTPUT.PUT_LINE('UNEXPECTED ERROR: ' || SQLERRM);
        ROLLBACK; — Always roll back any failed transaction
END;
/
```

Step 6: Creating Stored Procedures (Reusable Code)

A Stored Procedure is a named, compiled PL/SQL block stored in the database. It accepts parameters, performs an action-as in an update or complex transaction-and does not necessarily return a value.

6.1 Creating a Procedure with Parameters

We will create a procedure to adjust an employee's salary based on inputs.

```
CREATE OR REPLACE PROCEDURE adjust_salary (
    p_employee_id IN EMPLOYEES.employee_id%TYPE, — IN parameter
    p_adjustment_amount IN NUMBER          — IN parameter
)
IS
    — Declarations are often simple or empty here
    v_new_salary EMPLOYEES.salary%TYPE;
BEGIN
    — 1. Update the employee's salary
    UPDATE EMPLOYEES
    SET salary = salary + p_adjustment_amount
    WHERE employee_id = p_employee_id;
    — 2. Verify the new salary
```

```
    SELECT salary INTO v_new_salary
    FROM EMPLOYEES
    WHERE employee_id = p_employee_id;
    DBMS_OUTPUT.PUT_LINE('Salary updated for ID ' || p_employee_id);
    DBMS_OUTPUT.PUT_LINE('New Salary: ' || v_new_salary);
    COMMIT; — Save the changes
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Procedure Failed: Employee ' || p_employee_id || ' not found.');
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Unexpected Database Error.');
END adjust_salary;
/
```

- **CREATE OR REPLACE:** It creates the procedure; if it already exists, it replaces it without an error.
- **IN Parameter:** Utilized to pass values into the procedure.

6.2 Carrying out the Procedure

Once created, you can call the procedure from a SQL Worksheet:

— Assuming employee 1001 exists, give them a $500 bonus

EXEC adjust_salary(p_employee_id => 1001, p_adjustment_amount => 500);

Step 7: Continuing Your PL/SQL Journey

You have successfully navigated through the basic structures of PL/SQL! To learn more:

- **Functions:** Learn to create Functions, comparable to procedures but which must return a value.
- **Cursors:** Learn how to work with explicit cursors and the FOR loops with cursors to process multiple rows resulting from a query.
- **Triggers:** Study Database Triggers, that are PL/SQL blocks which execute automatically as a reaction to some DML or DDL event, for example, before an INSERT.

Ready to apply knowledge to challenging procedural problems and solve complex business requirements? Click here for a curated list of **Oracle PL/SQL Challenges and Solutions** to elevate your expertise!

Real Time Examples for Oracle PL/SQL Tutorial for Learners

Understanding how PL/SQL can solve business problems that require procedural logic added to the data operations is important for learners:

Implementing Complex Business Rules (Stored Procedures)

- **Scenario:** A university has to determine the final grade of a student, comprising complex weighted averages, minimum attendance requirements, grade capping

based on the number of retakes. This multistage logic is difficult to represent in Standard SQL.

- **Core PL/SQL Concepts:** Stored Procedures and Conditional Logic – IF-THEN-ELSE. Create a stored procedure that accepts student and course IDs as inputs, does the computation using variables, and applies the business rules through the use of IF-THEN statements before inserting the final grade in the GRADES table.

Enforcing Data Integrity (Triggers)

- **Scenario:** A healthcare system needs to log changes (medication, dosage, start date) to a patient's prescription in an audit history table before the record is actually modified in the main table, in order to maintain an immutable record for legal compliance.
- **Core PL/SQL Concept:** Database Triggers. Write a PL/SQL trigger to automatically run BEFORE UPDATE on the PRESCRIPTIONS table. The trigger code captures the old and new data values and performs an INSERT into the AUDIT_LOG table, guaranteeing that auditing occurs regardless of the application used for the update.

Batch Processing and Maintenance (Cursor FOR Loops)

- **Scenario:** A telecommunications company needs to process, at the end of each month, those customer accounts which are overdue. Thousands of accounts need to be checked, late fees calculated, and status changed from 'Active' to 'Suspended'.
- **Core PL/SQL Concept:** Explicit Cursors and Cursor FOR Loops. A cursor is used to efficiently iterate over, one by one (process), the subset of rows-overdue accounts-returned by a complex SELECT query. Within the loop, an UPDATE statement applies a fee and changes the status for each customer.

Ready to build something real? Click here for exciting **Oracle PL/SQL Project Ideas** to build your learner portfolio!

FAQs About Oracle PL/SQL Tutorial for Beginners

1. What is PL SQL in Oracle?

PL/SQL is Oracle's procedural extension to SQL. It embeds procedural capabilities-such as variables, conditional statements (IF-THEN), and loops-in SQL so that elaborate business logic can be efficiently executed within the database server.

2. What is the difference between PL SQL and SQL?

SQL is a declarative language to manage and query data; examples include SELECT and UPDATE. PL/SQL is a procedural language to write blocks of code which include logic, control flow, and error handling. PL/SQL uses SQL commands inside its procedural blocks.

3. Is PL SQL backend?

Yes, PL/SQL is a backend technology for the most part. It sits and runs on the database server, processing data, maintaining the business rules, and handling transactions. It acts like a strong middleware layer between the database and the front-end application.

4. Is PL/SQL faster than SQL?

Neither is intrinsically "faster." SQL is quick for an operation involving a single data. PL/SQL is quicker for complicated multi-step operations since it minimizes network traffic by running a single procedural block on the server instead of sending many individual SQL commands from the client.

5. What is the salary of PL SQL developer in TCS?

Salary for PL/SQL Developer at TCS varies widely based on experience, location, and demands of a specific project. For freshers or early-career professionals, the salaries are competitive with the average of the Indian IT industry for development roles and usually start off in the mid-range.

6. What is type in Oracle PLSQL?

A type in Oracle PL/SQL defines the type of data a variable can hold, such as NUMBER, VARCHAR2, and DATE. The main types are scalar types, composite types such as RECORD, and reference types by using the powerful %TYPE and %ROWTYPE attributes.

7. What are the three types of PL/SQL statements?

Broadly speaking, PL/SQL statements fall into three categories: procedural statements such as IF-THEN statements, loops, and assignments; SQL statements such as SELECT INTO, UPDATE; and compilation directives such as DECLARE, BEGIN.

8. Is PL/SQL worth learning?

Yes, PL/SQL is very worth learning, specially if you target enterprise environments – banking, telecom, and finance – which rely on Oracle's scalability and security; it is a high-value niche skill, mandatory for Oracle Developer and Database Engineer roles.

9. Is PL/SQL asked in an interview?

Yes, PL/SQL is quite frequently sought after in interviews that concern positions like Oracle Developer, Database Engineer, and Data Analyst supporting Oracle systems. It focuses on questions related to the writing of stored procedures, cursor management, exception handling, and performance tuning.

10. How many days will it take to learn PL/SQL?

You can learn the basics (blocks, simple logic, procedures) in about 1 to 2 weeks of dedicated effort. You usually become proficient, ready for practical development, including cursors and exception handling, in about 1 to 3 months of consistent practice.

Conclusion

With this, you have successfully laid the foundation of Oracle PL/SQL by learning how to create solid procedural blocks, handle errors, and integrate complex logic with SQL. In the development of efficient, secure, and centralized business applications, these abilities will be key within the Oracle ecosystem.

You are now ready to tackle real-world development challenges. Enroll in our full **Oracle PL/SQL Course in Chennai** now to master database programming!