# PL/SQL Chapter 3: Procedures and Functions

# 📘 PL/SQL Chapter 3: Procedures and Functions

---

## 🆓 Anonymous Blocks

An \*\*anonymous block\*\* is a block of PL/SQL code that:

- Has \*\*no name\*\*

- Is \*\*not stored\*\* in the database

- Is typically used for \*\*temporary tasks, quick tests, or throwaway scripts\*\*

### 💾 Characteristics

- Cannot be reused or called from another block

- Cannot be stored in the database permanently

- Typically saved locally as `.sql` files if needed

### 📌 Example of an Anonymous Block

DECLARE  
 v\_message VARCHAR2(50) := 'Hello from anonymous block';  
BEGIN  
 DBMS\_OUTPUT.PUT\_LINE(v\_message);  
END;

> ⚠️ Anonymous blocks are \*\*not suitable\*\* for large, reusable, or complex applications. For those, use \*\*procedures or functions\*\*.

---

## 🔧 What is a Procedure?

A \*\*procedure\*\* is a named block of PL/SQL statements that can be stored in the database and \*\*executed later by name\*\*.

Instead of rewriting the same PL/SQL logic repeatedly, you can define a procedure \*\*once\*\* and then call it wherever required.

This procedure can accept input parameters, process logic, and optionally return output.

---

## 💾 Syntax: Creating a Procedure

CREATE OR REPLACE PROCEDURE procedure\_name (  
 parameter1 [IN | OUT | IN OUT] datatype DEFAULT default\_value,  
 parameter2 [IN | OUT | IN OUT] datatype  
)  
AS  
 -- Declaration Section  
BEGIN  
 -- Executable Section  
EXCEPTION  
 -- Exception Handling  
END procedure\_name;

- `IN`: Value passed \*\*into\*\* the procedure (read-only)

- `OUT`: Value passed \*\*back\*\* from the procedure (write-only)

- `IN OUT`: Value \*\*passed in and modified\*\*, then returned

> ❗ While defining data types for parameters, \*\*do not specify lengths\*\* (e.g., avoid `VARCHAR2(20)` or `NUMBER(10)`). Just use `VARCHAR2` or `NUMBER`.

---

## 🔖 Parameter Modes Explained

| Mode | Description | Direction |

|----------|---------------------------------------------|-----------|

| `IN` | Accepts input, value cannot be modified | Caller ➡ Procedure |

| `OUT` | Returns output, uninitialized at start | Procedure ➡ Caller |

| `IN OUT` | Accepts and returns a modifiable value | ↔ Bidirectional |

---

## ✅ Benefits of Using Procedures

- \*\*Reusability\*\*: Define once, reuse many times

- \*\*Modularity\*\*: Break down complex logic into smaller units

- \*\*Maintainability\*\*: Update logic in one place

- \*\*Security\*\*: Can hide implementation details using private procedures

---

## 🔪 Example: Procedure with IN OUT Mode

CREATE OR REPLACE PROCEDURE ADD\_CUSTOMER(  
 c\_id IN OUT NUMBER,  
 c\_fname IN VARCHAR2,  
 c\_lname IN VARCHAR2,  
 c\_email IN VARCHAR2,  
 c\_phone IN VARCHAR2,  
 c\_address IN VARCHAR2,  
 c\_city IN VARCHAR2,  
 c\_state IN VARCHAR2,  
 c\_zip IN VARCHAR2,  
 c\_region IN VARCHAR2  
)  
AS  
BEGIN  
 INSERT INTO customer (  
 customer\_id, first\_name, last\_name, email, phone,  
 address, city, state, zip, region  
 ) VALUES (  
 c\_id, c\_fname, c\_lname, c\_email, c\_phone,  
 c\_address, c\_city, c\_state, c\_zip, c\_region  
 );  
  
 SELECT COUNT(1) INTO c\_id FROM customer;  
  
 COMMIT;  
 DBMS\_OUTPUT.PUT\_LINE('Data successfully inserted.');  
END ADD\_CUSTOMER;

### ▶️ Calling the IN OUT Procedure

DECLARE  
 tcount NUMBER := 45;  
BEGIN  
 ADD\_CUSTOMER(  
 c\_id => tcount,  
 c\_fname => 'Aria',  
 c\_lname => 'White',  
 c\_email => 'aria.white@example.com',  
 c\_phone => '9993331111',  
 c\_address => '101 Maple St',  
 c\_city => 'Atlanta',  
 c\_state => 'GA',  
 c\_zip => '30301',  
 c\_region => 'Southeast'  
 );  
 DBMS\_OUTPUT.PUT\_LINE('Total records = ' || tcount);  
END;

> 🔄 The same `c\_id` variable is used to send a value \*\*into\*\* the procedure and then \*\*receive a new value\*\* (total customer count) back.

---

## 🧑‍🎓 Summary of Procedure Calls

| Method | Key Benefit |

|--------------------|------------------------------|

| Positional | Simpler, but strict on order |

| Named Association | Flexible ordering |

> Both methods run the same compiled procedure and insert data correctly. Use named association for clarity in complex calls.

---

## 📌 Structure Recap

| Section | Purpose |

|-------------|----------------------------------------|

| `DECLARE` | Declare variables/constants (optional) |

| `BEGIN` | Write logic and SQL statements |

| `EXCEPTION` | Handle errors (optional) |

> In the upcoming sections, we’ll implement procedures and also explore \*\*functions\*\*—another modular unit that returns a value.

---

## 📊 What is a Function?

A \*\*function\*\* is similar to a procedure—it's a named block of PL/SQL—but it \*\*returns a single value\*\*.

Functions can also be used \*\*inside SQL expressions\*\*, unlike procedures.

---

## 📄 Syntax: Creating a Function

CREATE OR REPLACE FUNCTION function\_name (  
 parameter1 [IN | OUT | IN OUT] datatype  
)  
RETURN return\_datatype  
AS  
 -- Declaration section  
BEGIN  
 -- Executable logic  
 RETURN value;  
END function\_name;

---

## 🔪 Example: Function to Count Sales on a Given Date

CREATE OR REPLACE FUNCTION find\_salescount (  
 p\_sales\_date IN DATE  
) RETURN NUMBER  
AS  
 number\_of\_sales NUMBER;  
BEGIN  
 SELECT COUNT(\*) INTO number\_of\_sales  
 FROM sales  
 WHERE sales\_date = p\_sales\_date;  
  
 RETURN number\_of\_sales;  
END find\_salescount;

Once compiled, the function is stored in the database. You can view it under the \*\*Functions\*\* section in SQL Developer.

> 📌 Just like procedures, functions use parameter modes (`IN`, `OUT`, `IN OUT`) but always return a value.

---

## ▶️ Calling a Function

### 1. Inside a SQL Statement

SELECT find\_salescount(DATE '2024-01-01') FROM dual;

> You can call functions in SQL queries—\*\*not procedures\*\*.

### 2. Inside a PL/SQL Block

DECLARE  
 scount NUMBER := 0;  
BEGIN  
 scount := find\_salescount(DATE '2024-01-01');  
 DBMS\_OUTPUT.PUT\_LINE(scount);  
END;

> 🌟 Use functions when you need to return a single value and want to include the call in SQL expressions.