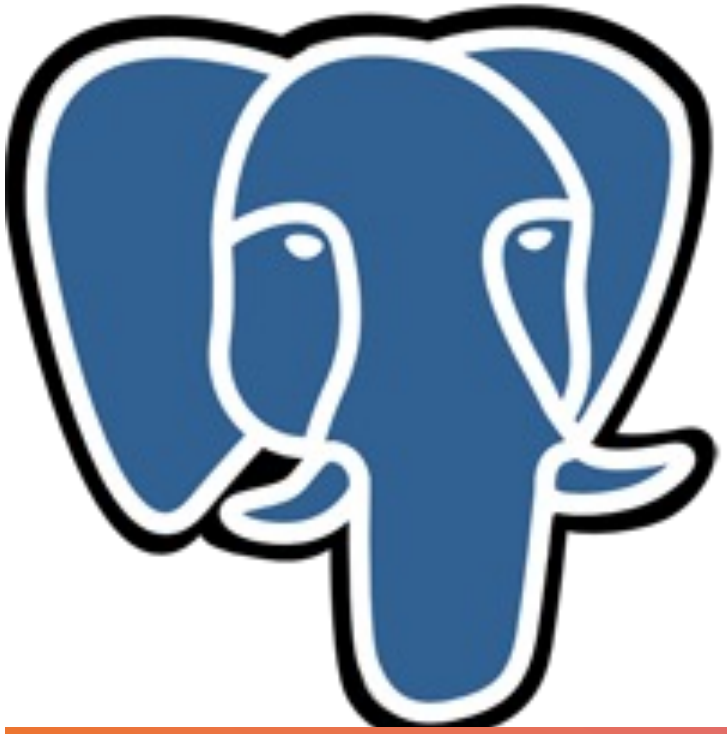


PostgreSQL

PL/SQL vs PL/pgSQL

Surendra Panpaliya

Agenda

- Syntax differences: DECLARE, BEGIN...END
- Control structures: IF, CASE, LOOP
- EXCEPTION handling in PostgreSQL

Hands-On

Write basic PL/pgSQL functions (factorial, greeting message)

Assignment

Rewrite an Oracle PL/SQL block into PL/pgSQL function

What is PL/SQL?

Procedural Language/Structured Query Language

is Oracle Corporation's procedural extension to SQL.

What is PL/SQL?

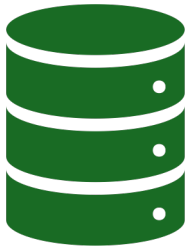
It allows for procedural constructs

such as variables, loops, and conditionals

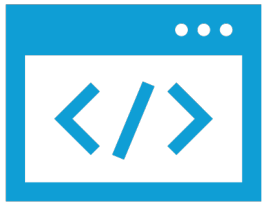
in addition to SQL.

What is PL/SQL?

Developed by: **Oracle**



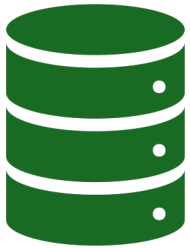
Used with: **Oracle Database**



Syntax: Similar to Ada and Pascal

What is PL/pgSQL?

Procedural Language/

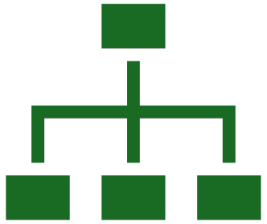


PostgreSQL Structured Query Language

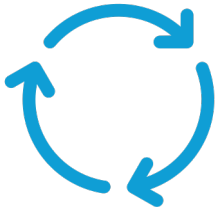
is PostgreSQL's native procedural language.

What is PL/pgSQL?

Extends SQL



by allowing control structures and



procedural logic.

What is PL/pgSQL?

Developed by: **PostgreSQL Community**

Used with: **PostgreSQL**

Syntax: Similar to PL/SQL, designed for PostgreSQL extensibility

Key Similarities

Feature	PL/SQL	PL/pgSQL
SQL Support	Yes	Yes
Procedural Constructs	Yes (IF, LOOP, WHILE, etc.)	Yes
Variable Declaration	Yes	Yes
Exception Handling	Yes (BEGIN ... EXCEPTION)	Yes
Cursor Support	Yes	Yes
Triggers	Yes	Yes

Key Differences

Aspect	PL/SQL (Oracle)	PL/pgSQL (PostgreSQL)
Vendor	Oracle Corporation	Open-source PostgreSQL
Compilation	Compiled to bytecode and stored in DB	Compiled to internal representation
Packages	Fully supports packages	No native package concept
Autonomous Transactions	Supported	Not supported directly

Key Differences

Aspect	PL/SQL (Oracle)	PL/pgSQL (PostgreSQL)
Data Types	Rich Oracle-specific types (%TYPE, %ROWTYPE)	Supports similar, but slightly different behavior
Exception Handling	More extensive with custom exceptions	Basic exception handling
Tooling	Tightly integrated with Oracle ecosystem	Uses psql, pgAdmin, or third-party tools
Permissions & Roles	Advanced role management	Simpler, evolving in recent versions

PL/SQL (Oracle)

```
DECLARE
  v_total NUMBER := 0;
BEGIN
  SELECT COUNT(*) INTO v_total FROM employees;
  DBMS_OUTPUT.PUT_LINE('Total Employees: ' || v_total);
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

PL/pgSQL (PostgreSQL)

```
DO $$
DECLARE
  v_total INTEGER := 0;
BEGIN
  SELECT COUNT(*) INTO v_total FROM employees;
  RAISE NOTICE 'Total Employees: %', v_total;
EXCEPTION
  WHEN OTHERS THEN
    RAISE NOTICE 'Error: %', SQLERRM;
END;
$$;
```

Summary

PL/SQL	PL/pgSQL
Proprietary and rich in features	Open-source and lightweight
Best for Oracle environments	Best for PostgreSQL environments
Strong tool and community support (Oracle-specific)	Open-source flexibility with active community

Syntax differences

DECLARE, BEGIN...END

Structure Overview: PL/SQL vs PL/pgSQL

Element	Oracle PL/SQL	PostgreSQL PL/pgSQL
Anonymous Block	Uses DECLARE, BEGIN...END; inside the block	Uses DO $$ DECLARE ... BEGIN ... END $$;
Block Terminator	Ends with END;	Ends with END; inside DO $$ ... $$;
Execution	Executed via SQL*Plus, TOAD, etc.	Executed via psql, pgAdmin, etc.

DECLARE Block – Syntax Difference

Oracle PL/SQL

```
DECLARE
  v_name VARCHAR2(100);
  v_count NUMBER;
BEGIN
  v_name := 'Amdocs';
  v_count := 1;
  DBMS_OUTPUT.PUT_LINE('Name: ' || v_name || ', Count: ' ||
v_count);
END;
```

PostgreSQL PL/pgSQL

```
DO $$
DECLARE
  v_name TEXT;
  v_count INTEGER;
BEGIN
  v_name := 'Amdocs';
  v_count := 1;
  RAISE NOTICE 'Name: %, Count: %', v_name, v_count;
END;
$$;
```

Explanation

DO $$ ... $$; is used to define

an anonymous code block in PostgreSQL.

PostgreSQL uses TEXT instead of VARCHAR2.

Instead of DBMS_OUTPUT.PUT_LINE,

PostgreSQL uses RAISE NOTICE.

BEGIN ... END Usage in Procedures

Oracle Procedure

```
CREATE OR REPLACE PROCEDURE show_employee_count IS
  v_total NUMBER;
BEGIN
  SELECT COUNT(*) INTO v_total FROM employees;
  DBMS_OUTPUT.PUT_LINE('Total employees: ' || v_total);
END;
```

PostgreSQL Function (since DO can’t return)

```
CREATE OR REPLACE FUNCTION show_employee_count()
RETURNS VOID AS $$
DECLARE
  v_total INTEGER;
BEGIN
  SELECT COUNT(*) INTO v_total FROM employees;
  RAISE NOTICE 'Total employees: %', v_total;
END;
$$ LANGUAGE plpgsql;
```

Note

- PostgreSQL prefers
- FUNCTION over PROCEDURE for returning values or VOID.
- PostgreSQL block must be wrapped in
- $$ with LANGUAGE plpgsql.

Note

- Return type
- RETURNS VOID is
- required even if nothing is returned.

Summary: Key Syntax Changes

Feature	Oracle PL/SQL	PostgreSQL PL/pgSQL
Anonymous Block	DECLARE ... BEGIN ... END;	DO $$ DECLARE ... BEGIN ... END $$;
Variable Data Types	VARCHAR2, NUMBER	TEXT, INTEGER, NUMERIC
Output Statement	DBMS_OUTPUT.PUT_LINE	RAISE NOTICE
Execution Wrapper	Direct (no wrapper needed)	Must wrap with DO $$ or FUNCTION
String Concatenation	`’	

Block Structure: DECLARE, BEGIN...END

Oracle PL/SQL	PostgreSQL PL/pgSQL
DECLARE block is optional	DECLARE block is optional
Uses BEGIN ... END; to start executable code	Same: BEGIN ... END;
PL/SQL procedures/functions must end with / when run in SQL*Plus	No trailing / required in PostgreSQL

Example: Block Structure

Oracle PL/SQL

```
BEGIN
   DBMS_OUTPUT.PUT_LINE('Hello from PL/SQL');
END;
/
```

Example: Block Structure

PostgreSQL PL/pgSQL

```
DO $$
BEGIN
   RAISE NOTICE 'Hello from PL/pgSQL';
END;
$$;
```

Variable Declaration

Oracle	PostgreSQL
variable_name datatype;	variable_name datatype;
Use %TYPE, %ROWTYPE	PostgreSQL supports %TYPE

Oracle PL/SQL

```
DECLARE
   v_name VARCHAR2(100);
BEGIN
   v_name := 'John';
END;
/
```

PostgreSQL PL/pgSQL

```
DO $$
DECLARE
   v_name VARCHAR(100);
BEGIN
   v_name := 'John';
END;
$$;
```

Control Structures in PL/SQL vs PL/pgSQL

Control Structure	Oracle PL/SQL	PostgreSQL PL/pgSQL
IF...THEN...END IF	Supported	Supported
CASE	Supported	Supported
LOOP, WHILE, FOR	Supported	Supported

IF Statement Oracle PL/SQL

```
DECLARE
  v_count NUMBER := 10;
BEGIN
  IF v_count > 5 THEN
    DBMS_OUTPUT.PUT_LINE('More than 5');
  ELSIF v_count = 5 THEN
    DBMS_OUTPUT.PUT_LINE('Exactly 5');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Less than 5');
  END IF;
END;
```

IF Statement PostgreSQL PL/pgSQL

```
DO $$
DECLARE
 v_count INTEGER := 10;
BEGIN
 IF v_count > 5 THEN
  RAISE NOTICE 'More than 5';
 ELSIF v_count = 5 THEN
  RAISE NOTICE 'Exactly 5';
 ELSE
  RAISE NOTICE 'Less than 5';
 END IF;
END;
$$;
```

CASE Statement Oracle PL/SQL

```
DECLARE
  v_status VARCHAR2(10) := 'A';
BEGIN
  CASE v_status
    WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Active');
    WHEN 'I' THEN DBMS_OUTPUT.PUT_LINE('Inactive');
    ELSE DBMS_OUTPUT.PUT_LINE('Unknown');
  END CASE;
END;
```

CASE Statement PostgreSQL PL/pgSQL

```
DO $$
DECLARE
 v_status TEXT := 'A';
BEGIN
 CASE v_status
  WHEN 'A' THEN RAISE NOTICE 'Active';
  WHEN 'I' THEN RAISE NOTICE 'Inactive';
  ELSE RAISE NOTICE 'Unknown';
 END CASE;
END;
$$;
```

PostgreSQL also supports **searched CASE**

```
CASE
  WHEN salary > 10000 THEN 'High'
  WHEN salary > 5000 THEN 'Medium'
  ELSE 'Low'
END
```

LOOP Structures Oracle PL/SQL

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

LOOP Structures PostgreSQL PL/pgSQL

```
DO $$
DECLARE
  i INTEGER := 1;
BEGIN
  LOOP
    EXIT WHEN i > 5;
    RAISE NOTICE 'i = %', i;
    i := i + 1;
  END LOOP;
END;
$$;
```

WHILE LOOP Oracle PL/SQL

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

WHILE LOOP PostgreSQL PL/pgSQL

```
DO $$
DECLARE
 i INTEGER := 1;
BEGIN
 WHILE i <= 5 LOOP
  RAISE NOTICE 'i = %', i;
  i := i + 1;
 END LOOP;
END;
$$;
```

FOR LOOP Oracle PL/SQL

```
BEGIN
  FOR i IN 1..5 LOOP
    DBMS_OUTPUT.PUT_LINE('i = ' || i);
  END LOOP;
END;
```

FOR LOOP PostgreSQL PL/pgSQL

```
DO $$
BEGIN
  FOR i IN 1..5 LOOP
    RAISE NOTICE 'i = %', i;
  END LOOP;
END;
$$;
```

Summary

Feature	Oracle PL/SQL Example	PostgreSQL PL/pgSQL Equivalent
IF Condition	IF ... THEN ... ELSE ... END IF	Same syntax
CASE Statement	CASE WHEN ... THEN ... END	Same syntax
Basic LOOP	LOOP ... EXIT WHEN ... END LOOP	Same, inside DO $$ ... $$;
WHILE LOOP	WHILE condition LOOP ... END	Same syntax
FOR LOOP	FOR i IN 1..N LOOP ... END	Same, but use RAISE NOTICE for output
Output	DBMS_OUTPUT.PUT_LINE()	RAISE NOTICE

What is EXCEPTION Handling in PostgreSQL

Allows you to catch and

manage **runtime errors**

division by zero,

null value violation,

foreign key violation

Oracle vs PostgreSQL Exception Syntax

Feature	Oracle PL/SQL	PostgreSQL PL/pgSQL
Block keyword	EXCEPTION	EXCEPTION
Default error catch	WHEN OTHERS THEN	WHEN OTHERS THEN
Error output	SQLERRM, SQLCODE	SQLERRM (limited), no SQLCODE
Logging function	DBMS_OUTPUT.PUT_LINE	RAISE NOTICE, RAISE EXCEPTION

Basic Syntax in PostgreSQL

```
DO $$
BEGIN
  -- risky operation
  PERFORM 1 / 0;
EXCEPTION
  WHEN division_by_zero THEN
    RAISE NOTICE 'Division by zero occurred!';
  WHEN OTHERS THEN
    RAISE NOTICE 'Some error: %', SQLERRM;
END;
$$;
```

Common Exception Types

Exception Type	Meaning
division_by_zero	Arithmetic error
undefined_table	Table does not exist
unique_violation	Duplicate value for a unique key
foreign_key_violation	Referential integrity failure
no_data_found	No rows returned from SELECT INTO
others	Generic catch-all like Oracle

Oracle PL/SQL

```
DECLARE
  v_name employees.name%TYPE;
BEGIN
  SELECT name INTO v_name FROM employees WHERE emp_id =
999;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No such employee!');
END;
```

PostgreSQL PL/pgSQL

```
DO $$
DECLARE
  v_name TEXT;
BEGIN
  SELECT name INTO v_name FROM employees WHERE emp_id = 999;
  RAISE NOTICE 'Name: %', v_name;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE NOTICE 'No such employee!';
END;
$$;
```

Rethrow or Custom Exceptions

```
DO $$
BEGIN
  RAISE EXCEPTION 'Custom error: %', 'Invalid operation';
END;
$$;
```

Rethrow or Custom Exceptions

```
-- re-raise caught exception

DO $$
BEGIN
  PERFORM 10 / 0;
EXCEPTION
  WHEN division_by_zero THEN
    RAISE EXCEPTION 'Critical: Division failed!';
END;
$$;
```

Best Practices for Amdocs Developers

Tip	PostgreSQL Approach
Catch specific errors first	List them before WHEN OTHERS
Avoid silent fails	Always log with RAISE NOTICE or RAISE EXCEPTION
Validate before risky operations	Use IF or EXISTS checks
Use GET STACKED DIAGNOSTICS	To capture error details (advanced)

Advanced Example with Diagnostics

```
DO $$
DECLARE
  err_msg TEXT;
BEGIN
  PERFORM 1 / 0;
EXCEPTION
  WHEN OTHERS THEN
    GET STACKED DIAGNOSTICS err_msg = MESSAGE_TEXT;
    RAISE NOTICE 'Error occurred: %', err_msg;
END;
$$;
```

Summary Table

Concept	Oracle PL/SQL	PostgreSQL PL/pgSQL
Exception block syntax	EXCEPTION ... WHEN ...	Same
Print error	DBMS_OUTPUT.PUT_LINE(SQLERRM)	RAISE NOTICE '...', SQLERRM
Catch all	WHEN OTHERS THEN	Same
Common exception types	NO_DATA_FOUND, ZERO_DIVIDE	no_data_found, division_by_zero
Custom error throw	RAISE_APPLICATION_ERROR	RAISE EXCEPTION 'message'

Control Structures (**IF-ELSE)**

Oracle PL/SQL	PostgreSQL PL/pgSQL
Same syntax	Same syntax

Oracle

```
IF salary > 5000 THEN
   DBMS_OUTPUT.PUT_LINE('High salary');
ELSE
   DBMS_OUTPUT.PUT_LINE('Low salary');
END IF;
```

PostgreSQL

```
IF salary > 5000 THEN
   RAISE NOTICE 'High salary';
ELSE
   RAISE NOTICE 'Low salary';
END IF;
```

CASE

Oracle	PostgreSQL
CASE WHEN … THEN … ELSE … END	Same syntax

LOOP Structures

Oracle	PostgreSQL
LOOP … END LOOP;	LOOP … END LOOP;
FOR and WHILE loops	Same in PostgreSQL

Oracle PL/SQL

```
FOR i IN 1..5 LOOP
   DBMS_OUTPUT.PUT_LINE(i);
END LOOP;
```

PostgreSQL PL/pgSQL

```
DO $$
BEGIN
   FOR i IN 1..5 LOOP
      RAISE NOTICE '%', i;
   END LOOP;
END;
$$;
```

Exception Handling

Oracle PL/SQL	PostgreSQL PL/pgSQL
EXCEPTION WHEN … THEN	Same
Predefined exceptions like NO_DATA_FOUND	Use WHEN OTHERS or specific SQLSTATE

Oracle PL/SQL

```
BEGIN
   SELECT salary INTO v_salary FROM employees WHERE employee_id = 100;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Employee not found');
END;
/
```

PostgreSQL PL/pgSQL

```
DO $$
DECLARE
   v_salary NUMERIC;
BEGIN
   SELECT salary INTO v_salary FROM employees WHERE employee_id =
100;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      RAISE NOTICE 'Employee not found';
END;
$$;
```

Note

In PostgreSQL,

NO_DATA_FOUND maps to:

EXCEPTION

WHEN NO_DATA_FOUND THEN -- works in PL/pgSQL

Or

use WHEN OTHERS for generic exceptions.

Summary Table

Feature	Oracle PL/SQL	PostgreSQL PL/pgSQL
Block Start	DECLARE (optional), BEGIN	DECLARE (optional), BEGIN
Output	DBMS_OUTPUT.PUT_LINE	RAISE NOTICE
Loops	LOOP, FOR, WHILE	Same
Exception	EXCEPTION WHEN	EXCEPTION WHEN
%TYPE, %ROWTYPE	Supported	Supported

Oracle

```
BEGIN
   INSERT INTO employees VALUES (100, 'Dev');
EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
      DBMS_OUTPUT.PUT_LINE('Duplicate employee');
END;
/
```

PostgreSQL

```
DO $$
BEGIN
   INSERT INTO employees VALUES (100, 'Dev');
EXCEPTION
   WHEN unique_violation THEN
      RAISE NOTICE 'Duplicate employee';
END;
$$;
```

Conclusion

Concept	Difference
Syntax	Mostly similar
Output	DBMS_OUTPUT.PUT_LINE vs RAISE NOTICE
Exception	Same structure, PostgreSQL uses SQLSTATE codes internally
Loops & Control	Almost identical

What is PL/pgSQL Function?

- A **Function in PostgreSQL** is a **named block of code** that:
- Takes **input parameters**
- Performs calculations or logic
- Returns a **result**

Syntax Structure

```
CREATE OR REPLACE FUNCTION function_name(parameters)
RETURNS return_type AS $$
DECLARE
   -- variable declarations
BEGIN
   -- statements
   RETURN value;
END;
$$ LANGUAGE plpgsql;
```

Function 1: Greeting Message

Create a function greet_user
that accepts a name and
returns a greeting message.

Create Function

```
CREATE OR REPLACE FUNCTION greet_user(p_name TEXT)

RETURNS TEXT AS $$
DECLARE
   greeting TEXT;
BEGIN
   greeting := 'Hello, ' || p_name || '! Welcome to PL/pgSQL.';
   RETURN greeting;
END;
$$ LANGUAGE plpgsql;
```

Call the Function

```
SELECT greet_user('Amdocs Developer’);
```

Output:

Hello, Amdocs Developer! Welcome to PL/pgSQL.

Function 2: Factorial Calculation

Create a function factorial that returns n!

Function 2: Factorial Calculation

```
CREATE OR REPLACE FUNCTION factorial(n INT)
RETURNS BIGINT AS $$
DECLARE
   result BIGINT := 1;
   i INT;
BEGIN
   IF n < 0 THEN
      RAISE EXCEPTION 'Factorial is not defined for negative numbers';
   END IF;
```

Function 2: Factorial Calculation

```
    FOR i IN 1..n LOOP
        result := result * i;
    END LOOP;

    RETURN result;
END;
$$ LANGUAGE plpgsql;
```

Call the Function

SELECT factorial(5);

Output:

120

Thank You

- Surendra Panpaliya
- Founder and CEO
- GKTCS Innovations
- https://www.gktcs.com