Chapter 2 — Advanced SQL

Procedural SQL Programming

Mr. Laïdi FOUGHALI

l.foughali@univ-skikda.dz

(Course materials ⇒ al-moualime.com)

University of Skikda — Department of Computer Science

1st Year Master RSD/AI

Advanced Databases (ADB)

Octobre 12, 2025

Version 1.0 (Initial) — 2025-10-25 à 07:55:41

Plan

General Objective

- The **standard SQL language** has, since **SQL-86 (ISO/IEC 9075 :1986)**, been a **declarative** language : it describes the expected result without imposing how to obtain it.
- Over time, applications required **sequential** processing, **conditional** tests, and **repeated** actions directly on the server side.
- The **SQL/PSM — Persistent Stored Modules** standard (**ISO/IEC 9075-4 :1996**) introduced **procedural SQL programming**.
- This evolution turns the DBMS into a full application engine, capable of executing internal logical programs ensuring **transactional coherence**, **security**, and **performance**.
- It improves **portability** and reduces client–server round trips.

[ISO/IEC 9075-4 :2016 — SQL/PSM ; Silberschatz et al., 2019]

Origins of Procedural SQL

- The early **SQL-86** and **SQL-92** standards defined a purely **declarative** language : one specifies the *result*, not the execution method.
- Evolving needs led to introducing **sequential** processing, **conditions**, and **loops** on the server side.
- The **SQL3** project (1992) formalized this extension with **SQL/PSM — Persistent Stored Modules** (**ISO/IEC 9075-4 :1996**), which **adds** :
  - `BEGIN ... END` blocks,
  - **local variables**,
  - **control flow** (`IF`, `WHILE`, `LOOP`),
  - **exception handling** (`HANDLER`).
- This standard inspired several procedural languages, **notably** :
  - **PL/pgSQL** (Procedural Language / PostgreSQL SQL — PostgreSQL, 1996),
  - **SQL/PSM for MySQL** (partial compliance with the standard, MySQL, 2005).
- **Goal :** make SQL capable of executing internal logical programs, combining declarative power with procedural control.

[ISO/IEC 9075-4 :1996 ; Elmasri & Navathe (2016) ; Gardarin (2003)]

Philosophy of Procedural SQL

- **Principle : bring application logic closer to the data** by executing processing directly **within the DBMS**.
- **Effects : fewer network exchanges**, **reduced latency**, better **transactional coherence** and **security**.
- The **DBMS becomes an integrated application engine** : **stored procedures**, **triggers**, and **business rules** without external dependencies.
- **Maintained duality** :
  - **Declarative** SQL : *desired result*,
  - **Procedural** SQL : *logical sequence to obtain it*.
- Ultimately, SQL remains **expressive**, **structured**, and **operational**, faithful to the relational model and adapted to modern needs.

[Silberschatz et al. (2019) ; ISO/IEC 9075-4 :2016]

PostgreSQL — Course Scope

- **Course scope :** all **practical examples** use **PostgreSQL** (procedures, functions, triggers, PL/pgSQL).
- **Theoretical reference :** content is based on the **SQL standards** (**ISO/IEC 9075**, including **SQL/PSM**) ; PostgreSQL-specific extensions are clearly identified.
- **Why PostgreSQL ?**
  - Reliable **open source**, backed by a large community.
  - **Highly compliant** with SQL standards, with well-documented extensions.
  - **Industrial-grade** system : robust, performant, rich tooling (replication, JSONB, FDW, etc.).
- **Global adoption :**
  - **Amazon** : *Aurora* and *AWS RDS for PostgreSQL*.
  - **Google** : *Cloud SQL for PostgreSQL*.
  - **Meta (Instagram)** : transactional storage based on PostgreSQL.
- **Pedagogical value :** a **standards-based** foundation and a technology **used in production**.

[ISO/IEC 9075 ; PostgreSQL Documentation ; AWS & Google Cloud]

Logical Structure of a Procedural Block

A **procedural block** is an executable unit grouping several SQL statements executed sequentially, with **local variables** and built-in **error handling**.
Objective : enable **imperative** logic directly within the DBMS.
Minimal composition :

- `DECLARE` section — declaration of local variables ;
- Main block `BEGIN ... END` — statements and control flow ;
- Optional `EXCEPTION` section — error handling.

Block execution runs within a **single transaction**, ensuring coherence and atomicity.

[ISO/IEC 9075-4 :2016]

Standard Syntax of a Procedural Block

Procedural block template per the **SQL/PSM** standard :

```
[DECLARE
    variable_1 type [DEFAULT value];
    variable_2 type;
    ...
]
BEGIN
    -- Sequential SQL statements
    -- Control structures: IF, WHILE, LOOP, FOR, CASE, etc.
EXCEPTION
    -- Error handling (DECLARE CONTINUE/EXIT HANDLER)
END;
```

Key principles :

- Blocks can be **nested** ;
- Variables are **local** to their block ;
- Unhandled exceptions are **propagated** to the parent block.

[ISO/IEC 9075-4 :2016 — Persistent Stored Modules]

Illustrative Procedural Block

Example of a procedural block executed directly on the server :

```
DO $$
DECLARE
  v_total DECIMAL(10,2);
BEGIN
  SELECT SUM(amount) INTO v_total FROM Invoices;
  RAISE NOTICE 'Total sales: %', v_total;
END $$;
```

Takeaways :

- `DO` executes an anonymous block without creating a persistent object ;
- `RAISE NOTICE` emits an informational message during execution ;
- `$$ ... $$` (*dollar-quoting*) delimits the block code.

[PostgreSQL 16 — PL/pgSQL Reference Manual]

Declaring and Using Variables

Variable declaration :

```
DECLARE v_amount DECIMAL(10,2);
DECLARE v_rate   DECIMAL(5,2) DEFAULT 0.15;
```

Assignment :

```
v_amount := 1000;
v_amount := v_amount * (1 + v_rate);
```

Notes :

- The := operator is used for local assignment ;
- `SELECT ... INTO` assigns a query result to a variable.

[PostgreSQL 16 — PL/pgSQL Variables]

Stored Procedure : Principle

- A **stored procedure** is a named block registered in the DBMS catalog.
- It executes multiple statements in a single call, improving coherence and performance.
- It accepts `IN`, `OUT`, and `INOUT` parameters.
- Unlike a function, it does not return a direct value ; results are passed via parameters or data effects.

[ISO/IEC 9075-4 :2016 ; PostgreSQL 16]

Creating a Stored Procedure

```
CREATE PROCEDURE P_AdjustBalance(
  IN p_id INT,
  IN p_amount DECIMAL(10,2),
  INOUT p_balance DECIMAL(10,2)
)
LANGUAGE plpgsql AS $$
BEGIN
  UPDATE Accounts
     SET balance = balance + p_amount
   WHERE id = p_id;

  SELECT balance INTO p_balance
    FROM Accounts WHERE id = p_id;
END $$;
```

Notes :

- `IN`, `OUT`, and `INOUT` parameters are supported since version 11 ;
- The `CALL` statement executes the procedure on the server ;
- Each `BEGIN...END` block runs within a local transaction context.

[PostgreSQL 16 — PL/pgSQL Reference Manual]

Executing a Stored Procedure

Execute a procedure with the `CALL` command :

```
CALL P_AdjustBalance(101, 200.00, p_balance);
```

Principles :

- The call executes the stored block on the server ;
- `OUT` or `INOUT` parameters retrieve results ;
- A procedure is not used with `SELECT`, unlike functions.

[PostgreSQL 16 — CALL Statement]

Creating an SQL Function

A **function** is a named procedural block that returns a **single value** and can be invoked in a query.

```
CREATE OR REPLACE FUNCTION F_VAT(amount DECIMAL(10,2))
RETURNS DECIMAL(10,2) AS $$
BEGIN
  RETURN amount * 0.19;
END;
$$ LANGUAGE plpgsql;

-- Invocation
SELECT F_VAT(1000);
```

Characteristics :

- Declared via `CREATE FUNCTION` ;
- Return value provided by the `RETURN` statement ;
- Functions cannot execute `COMMIT` nor `ROLLBACK`.

[PostgreSQL 16 — CREATE FUNCTION]

Control Structures

Control structures enable conditional or repetitive execution :

```
IF condition THEN
  statements;
ELSIF other_condition THEN
  other_statements;
ELSE
  default_statements;
END IF;

WHILE condition LOOP
  statement;
END LOOP;
```

Other forms : `FOR`, `LOOP`, `EXIT WHEN`, `CASE`.

[PostgreSQL 16 — Control Structures]

Exception Handling

Runtime errors are handled in an `EXCEPTION` block :

```
BEGIN
  IF p_amount <= 0 THEN
    RAISE EXCEPTION 'Invalid amount: %', p_amount;
  END IF;
EXCEPTION
  WHEN others THEN
    RAISE NOTICE 'Captured error: %', SQLERRM;
END;
```

Severity levels : `NOTICE`, `WARNING`, `EXCEPTION`.

[PostgreSQL 16 — Exception Handling]

Cursors and Sequential Processing

Cursors allow row-by-row processing of a result set :

```
DO $$
DECLARE
  c_inv CURSOR FOR SELECT id, amount FROM Invoices;
  v_id INT; v_amount DECIMAL;
BEGIN
  OPEN c_inv;
  LOOP
    FETCH c_inv INTO v_id, v_amount;
    EXIT WHEN NOT FOUND;
    RAISE NOTICE 'Invoice %: %', v_id, v_amount;
  END LOOP;
  CLOSE c_inv;
END $$;
```

[PostgreSQL 16 — Cursors in PL/pgSQL]

Triggers

A **trigger** automatically executes a function when an event (insert, update, delete) occurs on a table.

```
CREATE OR REPLACE FUNCTION F_LogSale()
RETURNS trigger AS $$
BEGIN
  INSERT INTO SalesLog VALUES (NEW.id, now());
  RETURN NEW;
END $$ LANGUAGE plpgsql;

CREATE TRIGGER T_LogSale
AFTER INSERT ON Invoices
FOR EACH ROW EXECUTE FUNCTION F_LogSale();
```

[PostgreSQL 16 — Triggers and Trigger Functions]

Transaction Management

A **transaction** groups several statements executed atomically :

```
START TRANSACTION;
  UPDATE Accounts SET balance = balance - 500 WHERE id = 1;
  UPDATE Accounts SET balance = balance + 500 WHERE id = 2;
COMMIT;

-- Rollback
ROLLBACK;
```

Procedural blocks run in a **single transactional context** : on error, a global `ROLLBACK` is triggered automatically.

[PostgreSQL 16 — Transactions and Error Handling]

Cross-DBMS Comparison

DBMS	Language	Specifics
SQL Standard	SQL/PSM	ISO 9075-4 syntax ; cross-DBMS portability.
PostgreSQL	PL/pgSQL	Compliant with SQL/PSM ; extensions (`TRIGGER`, `DO`, `RAISE`).
Oracle	PL/SQL	Packages, fine-grained exception handling, high performance.
SQL Server	T-SQL	Close to PSM ; Microsoft proprietary.
MySQL	SQL/PSM	Partial implementation ; limited `EXCEPTION`.

[ISO/IEC 9075-4 :2016 ; Oracle, PostgreSQL, MySQL, SQL Server Docs]

Conclusion

- **SQL/PSM** extends SQL into the procedural paradigm, bringing business logic closer to data ;
- Modern DBMSs (Oracle, PostgreSQL, SQL Server, MySQL) embrace this philosophy with extensions ;
- Procedural SQL improves **performance**, **security**, and **maintainability**.

[Silberschatz et al., 2019 ; ISO/IEC 9075-4 :2016]