PL/pgSQL

PL/pgSQL 是 PostgreSQL 中的过程化扩展语言，结合了 SQL 与过程化编程的特性，能够让用户在数据库中实现复杂的业务逻辑控制。它支持编写函数、存储过程、触发器等，提供了传统编程语言的控制结构，并与 SQL 紧密集成。

1. PL/pgSQL 的用途

PL/pgSQL 可以用于以下场景：

- **存储过程与函数**：编写复杂的数据库操作逻辑，存储于数据库中，能够通过函数调用进行复用。
- **触发器**：在特定数据库事件（如插入、更新、删除）发生时，自动触发执行自定义逻辑。
- **数据验证**：通过触发器或函数进行复杂的数据验证，确保数据一致性和完整性。
- **业务逻辑嵌入数据库**：将应用程序的业务逻辑直接嵌入到数据库中，减少应用程序与数据库之间的通信，提升性能。

2. PL/pgSQL 的语法结构

PL/pgSQL 提供了类似传统编程语言的语法结构，包括变量声明、条件判断、循环控制、错误处理等。以下是各个方面的详细介绍。

2.1 函数和存储过程

- **存储过程（Stored Procedure）** 是存储在数据库中的程序，它们可以直接调用并执行数据库操作。
- **函数（Function）** 则类似于存储过程，但可以返回值（如单个值或多个元组）。

函数的定义语法：

```
CREATE OR REPLACE FUNCTION funcName(param1 type, param2 type, ...)
RETURNS returnType AS $$
DECLARE
  variable declarations;
BEGIN
  -- 代码逻辑
  RETURN some_value;
END;
$$ LANGUAGE plpgsql;
```

- 参数可以有三种模式：
  - **IN**：输入参数，只能读取。
  - **OUT**：输出参数，只能写入。
  - **INOUT**：既能读取，也能写入。

2.2 示例：银行取款操作

以下是一个简单的银行取款函数示例，模拟从银行账户中扣款：

```
CREATE FUNCTION withdraw(acctNum text, amount integer) RETURNS text AS $$
DECLARE
  bal integer;
BEGIN
  SELECT balance INTO bal FROM Accounts WHERE acctNo = acctNum;
  IF (bal < amount) THEN
    RETURN 'Insufficient Funds';
  ELSE
    UPDATE Accounts SET balance = balance - amount WHERE acctNo = acctNum;
    SELECT balance INTO bal FROM Accounts WHERE acctNo = acctNum;
    RETURN 'New Balance: ' || bal;
  END IF;
END;
$$ LANGUAGE plpgsql;
```

- **逻辑解释**：
  - 首先查询账户余额 (`SELECT balance INTO bal`)，如果余额不足 (`IF bal < amount`)，则返回 "余额不足"。
  - 否则，扣减金额并更新账户余额。

3. 控制结构

PL/pgSQL 提供了多种控制结构，类似于传统的编程语言，用于实现复杂逻辑。

3.1 条件控制

- **IF ... THEN ... ELSE**：

```
IF condition THEN
  -- 语句块
ELSE
  -- 其他情况
END IF;
```

- **IF ... ELSIF ... ELSE**：

```
IF condition1 THEN
  -- 语句块1
ELSIF condition2 THEN
  -- 语句块2
ELSE
  -- 其他情况
END IF;
```

3.2 循环控制

- **基本循环**：`LOOP ... END LOOP;`

```
LOOP
  -- 语句块
  EXIT WHEN condition;
END LOOP;
```

- **计数循环**：`FOR i IN 1..10 LOOP ... END LOOP;`

```
FOR i IN 1..10 LOOP
  -- i 会依次取值 1 到 10
END LOOP;
```

4. 游标（Cursor）

游标是一个指针，用于逐行遍历查询结果集。

- **声明游标**：

```
<cursor_name> CURSOR FOR <query>;
```

- **操作游标**：
  - **打开游标**：`OPEN <cursor_name>;`
  - **提取数据**：`FETCH <cursor_name> INTO <variable>;`
  - **关闭游标**：`CLOSE <cursor_name>;`

示例：计算员工的总工资：

```
CREATE FUNCTION totalSalary() RETURNS real AS $$
DECLARE
  employee RECORD;
  totalSalary REAL := 0;
BEGIN
  FOR employee IN SELECT * FROM Employees LOOP
    totalSalary := totalSalary + employee.salary;
  END LOOP;
  RETURN totalSalary;
END;
$$ LANGUAGE plpgsql;
```

- **解释**：该函数通过游标逐行遍历 `Employees` 表中的每个员工，累加其工资以计算总和。

5. 触发器（Trigger）

触发器是数据库中的特殊程序，在特定的事件（如 `INSERT`、`UPDATE` 或 `DELETE`）发生时自动执行。

- **触发器的定义**：

```
CREATE TRIGGER TriggerName
AFTER/BEFORE Event ON TableName
FOR EACH ROW/STATEMENT
EXECUTE PROCEDURE FunctionName(args...);
```

- **触发器的作用**：例如在员工表中插入新员工时，自动更新部门的工资总和。

触发器函数示例：当新员工加入时更新部门工资总和：

```
CREATE FUNCTION totalSalary1() RETURNS trigger AS $$
BEGIN
  IF (NEW.dept IS NOT NULL) THEN
    UPDATE Department
    SET totSal = totSal + NEW.salary
    WHERE Department.id = NEW.dept;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TotalSalary1
AFTER INSERT ON Employees
FOR EACH ROW EXECUTE PROCEDURE totalSalary1();
```

- **解释**：在 `Employees` 表中插入新员工后，触发器自动执行 `totalSalary1()` 函数，更新对应部门的工资总和。

6. 错误处理（Exceptions）

PL/pgSQL 支持通过 `EXCEPTION` 处理运行过程中发生的异常。

- **示例**：处理除零异常：

```
BEGIN
  -- 代码块
EXCEPTION
  WHEN division_by_zero THEN
    RAISE NOTICE 'Caught division_by_zero';
END;
```

    - 如果代码块中发生了除零错误，将会捕获并输出 `Caught division_by_zero` 的通知。

7. 用户自定义函数（UDF）

PL/pgSQL 支持用户创建自定义函数（UDF），可以用于查询、复杂计算等。

- **类型**：
  - **SQL 函数**：直接在 SQL 中调用的函数。
  - **过程化语言函数**：如 PL/pgSQL 函数，支持复杂逻辑控制。