14. PSM

永久存储模块 (Persistent Stored Module, PSM)：在 SQL 2016被提出，可以写过程与函数，并将其存入数据库中。永久存储模块为数据库模式的一部分，并且可以在 SQL 查询或其他命令中被使用，在商用 DBMS 中被称为**存储过程**。使用 PSM 的优势如下：

- 我们可以从任何地方调用 PSM 过程，包括通用 SQL 接口、其他 PSM 与嵌入式 SQL；
- 同时，PSM 函数可以被用作表达式的一部分，如：

```
INSERT INTO SC VALUES(getSno('james'), 'CS123', 100)
```

创建过程/函数

```
CREATE PROCEDURE name(parameters)
  local declarations;
  procedure body;
CREATE FUNCTION name(parameters) RETURNS type
  local declarations;
  function body;
```

- `body` 中只允许一条语句（不过可以通过复合语句进行转化）。
- `parameter` 中的参数是三元组：`mode para_name type`，其中
  - `mode` 可以是输入 `IN`（默认），输出 `OUT` 或输入输出 `INOUT`；
  - 函数的参数只能是 `IN` 模式，以防止意外修改。
- 例如，

```
CREATE PROCEDURE rename(IN oldName VARCHAR(50), IN newName VARCHAR(50))
  UPDATE S SET dept=newName WHERE dept=oldName;
```

调用过程/函数：使用 `CALL` 命令。

```
CALL proc-name (argument-list);
```

`CALL` 既可以在通用 SQL 接口中使用，也可以在另一个 PSM 过程/函数中使用，也可以在原语言程序中使用（此时需要加入 `EXEC SQL` 标识符表示执行的是 SQL 语句）。

返回语句：使用 `RETURN` 命令，仅在函数中出现，设置函数的返回值。

```
RETURN expression;
```

- **注意**：返回语句不会终止函数！接下来的语句将会继续执行，很有可能接下来的语句将会修改返回值，从而函数完成后返回值与初始 `RETURN` 时的已经不同。

变量：声明局部变量采用如下格式，局部变量将只在过程/函数执行过程中保留。

```
DECLARE name type;
```

- **注意**：只能在过程/函数体 `body` 前进行变量声明！

赋值：使用如下语句进行变量赋值。

```
SET variable = expression;
```

- `NULL` 可以为合法的 `expression`；
- `expression` 也可以是一个 SQL 询问——只要其返回标量。

复合语句（语句块）：使用如下语句进行语句的复合，SQL 将会将其看成单语句执行；可以出现在任何地方。

```
BEGIN [[NOT] ATOMIC]
  local declarations
  statement list
END;
```

语句标签：使用如下语句为 `statement` 加上标签 `label`，经常在循环中被使用。

```
label: statement;
```

分支语句：使用如下语句进行条件判断：

```
IF condition THEN
  statement list
ELSEIF condition THEN
  statement list
...
ELSE
  statement list
END IF;
```

- `condition` 与 `WHERE` 后跟的要求相同；
- `statement list` 可以不用复合语句 `BEGIN ... END;`；
- 最终的 `ELSE` 是可选的。

SQL 查询：在 PSM 中使用查询有如下几种方式：

1. 子查询的使用遵循 SQL 语法（如在 `condition` 中）；
2. 返回单值标量的查询可以用在赋值语句的右边；
3. 一个单行的 `SELECT` 是一个合法的语句，可以通过 `INTO` 将结果存入变量或参数中；
4. 游标的使用与嵌入 SQL 基本相同。

`LOOP` 语句：使用如下语句进行无限循环。

```
LOOP
  statement list
END LOOP;
```

- 如果后续有带标签的语句，可以在循环过程中使用 `LEAVE` 跳出循环

  ```
  LOOP
    ...
    LEAVE myloop
    ...
  END LOOP;
  ...
  myloop: statement;
  ```

- `LOOP` 经常被用在使用游标获取查询元组时。
- 可以通过定义条件来方便地使用 `SQLSTATE` 返回的状态。

  ```
  DECLARE cond_name CONDITION FOR SQLSTATE value;
  ```

  一个例子如下：

  ```
  DECLARE Not_Found for SQLSTATE '02000';
  ```

`FOR` 语句：使用如下语句进行游标遍历循环。

```
FOR loopvar AS csrname CURSOR FOR query DO
  statement list
END FOR;
```

- 仅被用来遍历游标，该语句可以处理所有游标使用的细节；
- 询问中的 `SELECT` 后选出的属性被看作（已定义好的）局部变量。
- `loopvar` 和 `csrname` 在实际中基本不会使用，主要是为了迁移到 `LOOP` 的方便（SQL 会被翻译为 `LOOP` 进行处理）；这两个字段也是可选的，PSM 会为其安排默认值（名字）。

条件循环语句：使用如下语句进行条件循环。

```
WHILE condition DO
  statement list
END WHILE;
REPEAT
  statement list
UNTIL condition
END REPEAT;
```

异常处理器 (exception handler)：每当执行时发生错误，会有相应代码唤醒异常处理器，其定义如下：

```
DECLARE where-to-go-next HANDLER
FOR condition-list
  statement;
```

- `where-to-go-next` 参数可选以下几项：
  - `CONTINUE`：异常处理器语句执行完毕后，继续执行触发异常的语句的下一条语句；
  - `EXIT`：异常处理器语句执行完毕后，离开当前定义 `HANDLER` 的 `BEGIN...END` 语句块，然后从该语句块的下个语句继续执行。
  - `UNDO`：与 `EXIT` 相同，除了撤销该语句块内执行的所有操作。
- `conditoin-list` 是定义的触发异常处理器的条件或者表达式列表，一般有 `SQLSTATE 'xxxxx'` 形式。