数据库系统原理

Database System Principles

四川大学计算机学院

段 磊

leiduan@scu.edu.cn

2014.9

第八章 数据库编程

- 标准SQL缺少流程控制能力，难以实现应用业务中的逻辑控制
- SQL编程技术可以有效克服SQL语言实现复杂应用方面的不足，提高应用系统和RDBMS间的互操作性

本章目录

8.1 嵌入式SQL

- SQL语言提供了两种不同的使用方式。
  - 一种是在终端交互式方式下使用，我们前面介绍的就是做为独立语言由用户在交互环境下使用的SQL语言。
  - 另一种是将SQL语言嵌入到某种高级语言如Pascal、COBOL、FORTRAN、C中使用，利用高级语言的过程性结构来弥补SQL语言在实现复杂应用方面的不足，这种方式下使用的SQL语言称为嵌入式SQL（Embedded SQL），而嵌入SQL的高级语言称为主语言或宿主语言。

8.1.1 一般形式

1. 语句级接口 如PB Script。
2. 调用级接口 如JDBC ODBC。

在有的嵌入式SQL中，为了能够区分SQL语句与主语言语句，所有SQL语句都必须加前缀EXEC SQL

如：C中嵌入可能形式

EXEC SQL DROP TABLE　Student;

8.1.2 嵌入式SQL与主语言间的通信

将SQL嵌入到高级语言中混合编程，SQL语句负责操纵数据库，高级语言语句负责控制程序流程。

最重要的是两者之间的通信问题：

1. 向主语言传递SQL语句的执行状态；
2. 主语言向SQL提供参数(主变量实现)；
3. SQL查询结果交还主语言处理。

嵌入式SQL与主语言间的通信

一、SQL通信区

SQL语句执行后，系统要反馈给应用程序若干信息，主要包括描述系统当前工作状态和运行环境的各种数据，这些信息将送到SQL通信区SQLCA中。应用程序从SQLCA中取出这些状态信息，据此决定接下来执行的语句。

如下所示的SQLCA数据连接字。

嵌入式SQL与主语言间的通信

```
SQLCA.DBMS = "MSS"

SQLCA.Database = “Student"

SQLCA.LogId = "sa"

SQLCA.LogPass = ""

SQLCA.ServerName ="ZTQ\ZTQ_TEST"

CONNECT USING SQLCA;
```

嵌入式SQL与主语言间的通信

二、主变量

SQL语句中使用主语言的变量。SQL语句中主变量前加冒号。

输入主变量，输出主变量(相对于SQL语句)。

输入主变量：应用程序赋值，SQL使用。

实际上SQL把它当常量处理。

输出主变量：由SQL语句对其赋值或设置状态信息，返回应用程序。

指示变量：附加在主变量上，指示所指主变量的值(空值)或条件。

嵌入式SQL与主语言间的通信

三、游标

为处理集合结果而引入。

四、程序实例

注意：定义游标后的Select 目标列是表属性名，不是主变量。

8.1.3 不用游标的SQL语句

- 适用于全部说明性语句、数据定义、控制语句，Select结果为单个记录的查询语句，非CURRENT形式的UPDATE和DELETE语句。

8.1.4 使用游标的SQL语句

一、查询结果为多条记录的SELECT语句。

本例要查询deptname系的所有学生的学号、姓名、性别和年龄。

首先定义游标SX，将其与查询结果集（即deptname系的所有学生的学号、姓名、性别和年龄）相联系（步骤①）。这时相应的SELECT语句并没有真正执行。

然后打开游标SX，这时DBMS执行与SX与相联系的SELECT语句，即查询deptname系的所有学生的学号、姓名、性别和年龄（步骤②），之后SX处于活动状态。

使用游标的SQL语句

- 接下来在一个循环结构中逐行取结果集中的数据，分别将学号Sno、姓名Sname、性别Ssex和年龄Sage送至主变量HSno、HSname、HSsex和HSage中（步骤③）。主语言语句将对这些主变量做进一步处理。
  最后关闭游标SX（步骤④）。这时SX不再与deptname系的学生数据相联系。
  被关闭的游标SX实际上可以再次被打开，与新的查询结果相联系。例如，可以在例1中再加上一层外循环，每次对deptname赋新的值，这样SX就每次和不同的系的学生集合相联系。

使用游标的SQL语句

```
String Cname,Cno='1';
DECLARE CX CURSOR FOR
    SELECT Cname
          FROM course;
OPEN CX;
FETCH CX INTO :cname;
do while SQLCA.SQLCODE = 0
    ddlb_1.AddItem(cname);
    FETCH CX INTO :cname;
loop
close CX;
```

使用游标的SQL语句

二、CURRENT形式的UPDATE和DELETE语句。

8.1.5 动态SQL

- 查询对象、查询条件、要查询的属性列不确定。
- 特点：在查询执行时临时组装。
- 实现技术：实际不是SQL技术，是程序设计技术。

动态SQL

- Power Builder的四种动态SQL格式
  - 无参数、无返回结果的SQL语句。如建表、插入元组值等场合

```
String Mysql
Mysql=″INSERT INTO Course″&
          +″VALUES(‘5′,′JAVA′)″;
EXECUTE IMMEDIATE：Mysql；
```

动态SQL

- 需要参数，无返回值

```
INT  Emp_id_var = 56

PREPARE SQLSA FROM "DELETE FROM
   employee WHERE emp_id=?" ;

EXECUTE SQLSA USING : Emp_id_var ;
```

动态SQL

- 有返回值的
  - 编译时能确定参数个数和返回结果字段
- 有返回值的
  - 编译时可能无法确定参数个数和返回结果字段

- JDBC相关内容参阅“实验环节4”

本章目录

8.2 存储过程

- SQL：1999标准SQL-invoked routines
  - SQL-invoked function
  - SQL-invoked procedure
- PL/SQL(Procedural Language/SQL)
  - 最常用的一种编写存储过程语言
  - 结合了SQL数据操纵能力和过程化语言的流程控制能力

8.2.1 PL/SQL块结构

- 块是PL/SQL程序的基本结构

```
DECLARE
... /* 变量、常量、游标、异常等的声明定义 */
BEGIN
..... /* 主要处理功能部分 */
EXECEPTION
..... /* 异常处理部分 */
END;
```

8.2.2 变量常量的定义

- 常量定义
  - 〈常量名〉〈数据类型〉CONSTANT:=〈常量值表达式〉;
  - sage0 INT CONSTANT:=25;
- 变量定义
  - 〈变量名〉〈数据类型〉[NOT NULL] [[:=]〈初始值表达式〉];

变量常量的定义

- sage INT;
  sage INT 25;
  sage INT :=25+sage0;
  sage INT NOT NULL :=25;
- 赋值语句
  - 〈变量名〉:=〈表达式〉;
  - sage:=sage+4;

8.2.3 控制结构

- 条件语句（可嵌套）
  - IF 〈条件〉 THEN
    〈语句块〉
    [ELSE
    〈语句块〉]
    END IF;
- 循环
  - LOOP
  - WHILE-LOOP
  - FOR-LOOP
- 错误（异常）处理

8.2.4 存储过程

- 存储过程：命名PL/SQL块
  - 匿名和命名PL/SQL块
- 优点
  - 高效率（预解析、预编译）
  - 低通信量（提交请求简单）
  - 方便规则实施和变化

存储过程

- 创建、修改、执行和删除
  - 创建
    - CREATE PROCEDURE 〈过程名〉([参数1, 参数2,…]) AS 〈PL/SQL块〉
  - 修改：
    - ALTER PROCEDURE ……
  - 执行
    - CALL|PERFORM PROCEDURE 〈过程名〉([参数1, 参数2,…]);
  - 删除
    - DROP PROCEDURE 〈过程名〉;

触发器 （复习）

- 触发器的要点
  - 事件 insert、update、delete
  - 条件
  - 满足条件后执行的动作
- 例子
  - 银行存款透支后开一等额贷款账户

触发器 （复习）

- **create trigger** *overdraft-trigger* **after update on** *account*
  referencing new row as *nrow* //nrow引用新值
  for each row
  when *nrow.balance* < 0 //等待的时机 余额为负
  begin atomic
  insert into *loan* **values**
  (n.*row.account-number, nrow.branch-name, nrow.balance*);
  update *account* **set** *balance* = 0 //把存款余额改为**0**
  where *account.account-number*
  = nrow.account-number
  end

本章目录

8.3 ODBC编程

8.3.1数据库互联概述

- DB种类繁多，接口不一
- ODBC
  - Open Database Connectivity,开放式数据库连接性
  - 一组DB访问规范API接口定义
    - 开放式连接
    - 传送查询和更新并得到结果

8.3.2 ODBC原理

- 应用程序 (后面有例子)
- 驱动程序管理器
  - ODBC32.DLL，微软的ODBC管理（连接驱动程序、管理数据源……）
- 数据库驱动程序
  - 一般由DBMS厂商提供
- 数据源
  - 在连接中，用DSN代表用户名、服务器名、数据库名等

8.3.3 ODBC API基础

- 打开数据库联机 SQLConnect(). 参数包括
  - connection handle 连接句柄
  - the server to which to connect 服务器
  - the user identifier 用户标识
  - password 口令
- 一种重要的参数类型
  - SQL_NTS SQL_NTS表示null结束的字符串

代码实例(1)

```
int ODBCexample()  // C函数
{  RETCODE error;
   HENV     env;      // environment  环境
   HDBC     conn;  /* database connection */
    //连接
   SQLAllocEnv(&env);   //分配环境
   SQLAllocConnect(env, &conn);      连接
   连接 服务器 用户，指定 口令，类型
```

代码实例(2)

```
SQLConnect(conn, "aura.bell-labs.com", SQL_NTS, "avi",
   SQL_NTS, "avipasswd", SQL_NTS);
  {
   char branchname[80];//支行名称
   float  balance;余额;
   HSTMT   stmt;
   int  lenOut1, lenOut2
   SQLAllocStmt(conn, &stmt);  分配语句空间
   char * sqlquery = "select branch_name, sum
            (balance) from account
                  group by branch_name";
```

代码实例(3)

```
error = SQLExecDirect(stmt, sqlquery, SQL_NTS);  //真正送SQL到DB Server
  if (error == SQL_SUCCESS) {
      SQLBindCol(stmt, 1, SQL_C_CHAR,   branchname , 80, &lenOut1);
      SQLBindCol(stmt, 2, SQL_C_FLOAT, &balance, 0 , &lenOut2);
        while (SQLFetch(stmt) >= SQL_SUCCESS) {
          printf (" %s  %g\n", branchname, balance);
      }
}
```

代码实例(4)

```
//善后工作：
    SQLFreeStmt(stmt, SQL_DROP); }
  SQLDisconnect(conn);       // 断开连接
  SQLFreeConnect(conn);    //释放空间
  SQLFreeEnv(env);           //释放环境空间
  }
```

Any Question?

Thank you !