数据库系统概论

An Introduction to Database System

第八章 数据库编程

8.1 嵌入式SQL

- SQL语言提供了两种不同的使用方式：
  - 交互式
  - 嵌入式
- 为什么要引入嵌入式SQL
  - SQL语言是非过程性语言，面向集合，功能丰富，但缺少流程控制能力，及与用户的交互能力
  - 应用业务中的逻辑控制需要高级语言
- 在程序设计的环境下，SQL语句要做某些必要的扩充

8.1 嵌入式SQL（CONT）

- 主语言
  - 嵌入式SQL是将SQL语句嵌入程序设计语言中，被嵌入的程序设计语言，如C、C++、Java，称为宿主语言，简称主语言。
- 为了区分SQL语句与主语言语句， 所有SQL语句必须加前缀EXEC SQL，以(;)结束:

  EXEC SQL <SQL语句>;

8.1 嵌入式SQL

1. 嵌入式SQL语句与主语言之间的通信
2. 不使用游标的SQL语句
3. 使用游标的SQL语句
4. 小结

嵌入式SQL语句与主语言之间的通信

- 将SQL嵌入到高级语言中混合编程，程序中会含有两种不同计算模型的语句
  - SQL语句
    - 描述性的面向集合的语句
    - 负责操纵数据库
  - 高级语言语句
    - 过程性的面向记录的语句
    - 负责控制程序流程
  - 它们之间应该如何通信？

嵌入式SQL语句与主语言之间的通信（续）

- 1. SQL通信区（SQLCA）
  - 向主语言传递SQL语句的执行状态信息
  - 使主语言能够据此控制程序流程
- 2. 主变量
  - 主语言向SQL语句提供参数
  - 将SQL语句查询数据库的结果交主语言进一步处理
- 3. 游标
  - 协调集合性操作语言与过程性操作语言的不匹配

一、SQL通信区

- SQLCA： SQL Communication Area
  - SQLCA是一个数据结构
- SQLCA的用途
  - SQL语句执行后，RDBMS以下信息送到SQL通信区SQLCA中
    - 描述系统当前工作状态
    - 描述运行环境
  - 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

SQL通信区

- SQLCA使用方法：
  - 定义SQLCA
    - 用EXEC SQL INCLUDE SQLCA定义
  - 使用SQLCA
    - SQLCA中有一个存放每次执行SQL语句后返回代码的变量SQLCODE
    - 如果SQLCODE等于预定义的常量SUCCESS，则表示SQL语句成功，否则表示出错
    - 应用程序每执行完一条SQL 语句之后都应该测试一下SQLCODE的值，以了解该SQL语句执行情况并做相应处理

二、主变量与指示变量

- 主变量
  - 嵌入式SQL语句中可以使用主语言的程序变量来输入或输出数据
  - 在SQL语句中使用的主语言程序变量简称为主变量（Host Variable）
- 指示变量：
  - 一个主变量可以附带一个指示变量（Indicator Variable）。用途稍后介绍。

主变量（续）

- 在SQL语句中使用主变量和指示变量的方法
  - 1) 说明主变量和指示变量

    BEGIN DECLARE SECTION

    .....

    .....　　(说明主变量和指示变量)

    .....

    END DECLARE SECTION
  - 2) 使用主变量
    - 说明之后的主变量可以在SQL语句中任何一个能够使用表达式的地方出现
    - SQL语句中的主变量名前要加冒号（:）作为标志
  - 3) 使用指示变量
    - 指示变量前也必须加冒号标志
    - 必须紧跟在所指主变量之后

三、游标（cursor）

- 为什么要使用游标
  - SQL语言与主语言具有不同数据处理方式
  - SQL语言是面向集合的，一条SQL语句原则上可以产生或处理多条记录
  - 主语言是面向记录的，一组主变量一次只能存放一条记录
  - 游标用来协调这两种不同的处理方式

游标（续）

- 游标
  - 游标是系统为用户开设的一个数据缓冲区，存放SQL语句的执行结果
  - 每个游标区都有一个名字
  - 用户可以用SQL语句逐一从游标中获取记录，并赋给主变量，交由主语言进一步处理

四、建立和关闭数据库连接

- 建立数据库连接

  EXEC SQL CONNECT TO *target* [AS *connection-name*] [USER *user-name*];

  *target*是要连接的数据库服务器：

  - 常见的服务器标识串，如<dbname>@<hostname>:<port>
  - DEFAULT

  *connect-name*是可选的连接名，连接名必须是一个有效的标识符，在整个程序内只有一个连接时可以不指定连接名

- 关闭数据库连接

  EXEC SQL DISCONNECT [*connection-name*];

- 程序运行过程中可以修改当前连接：

  EXEC SQL SET CONNECTION *connection-name* | DEFAULT;

五、程序实例

[例1]依次检查某个系的学生记录，交互式更新某些学生年龄。

```
EXEC SQL BEGIN DEC LARE SECTION;  /*主变量说明开始*/
    char HSno[64];
    char HSname[64];
    char HSsex[64];
    int    HSage;
    char deptname[64];
    int    NEWAGE;
EXEC SQL END DECLARE SECTION;     /*主变量说明结束*/
long SQLCODE;
EXEC SQL INCLUDE sqlca;              /*定义SQL通信区*/
```

程序实例（续）

```
int main(void)                                          /*C语言主程序开始*/
{
    int     count = 0;
    char  yn;                                           /*变量yn代表yes或no*/
    printf("Please choose the department name(CS/MA/IS): ");
    scanf("%s", deptname);                              /*为主变量deptname赋值*/
    EXEC SQL CONNECT TO TEST@localhost:54321 USER
     "SYSTEM" /"MANAGER";                               /*连接数据库TEST*/
    EXEC SQL DECLARE SX CURSOR FOR   /*定义游标*/
            SELECT Sno, Sname, Ssex, Sage   /*SX对应语句的执行结果*/
            FROM Student
            WHERE SDept = :deptname;
    EXEC SQL OPEN SX;       /*打开游标SX便指向查询结果的第一行之前*/
```

程序实例（续）

```
for ( ; ; )                          /*用循环结构逐条处理结果集中的记录*/
{
    EXEC SQL FETCH SX INTO :HSno, :HSname, :HSsex,:HSage;
                                     /*推进游标，将当前数据放入主变量*/
    if (sqlca.sqlcode != 0)          /* sqlcode != 0,表示操作不成功*/
      break;                         /*利用SQLCA中的状态信息决定何时退出循环*/
    if(count++ == 0)                 /*如果是第一行的话，先打出行头*/
      printf("\n%-10s %-20s %-10s %-10s\n", "Sno", "Sname", "Ssex", "Sage");
      printf("%-10s %-20s %-10s %-10d\n", HSno, HSname, HSsex, HSage);
      printf("UPDATE AGE(y/n)?"); /*询问用户是否要更新该学生的年龄*/
     do{
            scanf("%c",&yn);
      }
      while(yn != 'N' && yn != 'n' && yn != 'Y' && yn != 'y');
```

程序实例（续）

```
    if (yn == 'y' || yn == 'Y')  {              /*如果选择更新操作*/
            printf("INPUT NEW AGE:");
            scanf("%d",&NEWAGE);          /*用户输入新年龄到主变量中*/
            /*对当前游标指向的学生年龄进行更新*/
            EXEC SQL UPDATE Student
                    SET Sage = :NEWAGE
                    WHERE CURRENT OF SX ;
    }}

    EXEC SQL CLOSE SX;              /*关闭游标SX不再和查询结果对应*/
    EXEC SQL COMMIT WORK;          /*提交更新*/
    EXEC SQL DISCONNECT TEST; /*断开数据库连接*/
}
```

8.1 嵌入式SQL

1. 嵌入式SQL语句与主语言之间的通信
2. 不使用游标的SQL语句
3. 使用游标的SQL语句
4. 小结

8.1.3 不用游标的SQL语句

- 不用游标的SQL语句的种类
  - 说明式语句
  - 数据定义语句
  - 数据控制语句
  - 查询结果为单记录的SELECT语句
  - 非CURRENT形式的增删改语句

不用游标的SQL语句（续）

- 一、查询结果为单记录的SELECT语句
- 二、非CURRENT形式的增删改语句

一、查询结果为单记录的SELECT语句

- 这类语句不需要使用游标，只需要用INTO子句指定存放查询结果的主变量

[例2] 根据学生号码查询学生信息。假设已经把要查询的学生的学号赋给了主变量givensno。

```
EXEC SQL SELECT Sno，Sname，Ssex，Sage，Sdept
                INTO  :Hsno，  : Hname ，:Hsex，:Hage，:Hdept
                FROM  Student
                WHERE Sno=:givensno；
```

查询结果为单记录的SELECT语句（续）

(1) INTO子句、WHERE子句和HAVING短语的条件表达式中均可以使用主变量

(2) 如果查询结果实际上并不是单条记录，而是多条记录，则程序出错，RDBMS会在SQLCA中返回错误信息

(3) 查询返回的记录中，可能某些列为空值NULL。

为了表示空值，在INTO子句的主变量后面可以跟有指示变量，当查询到的某个数据项为空值时，系统自动将指示变量置为负值。因此当指示变量为负值时，不管主变量为何值，均认为值为NULL。

注：指示变量只能用在INTO子句。

查询结果为单记录的SELECT语句（续）

[例3] 查询某个学生选修某门课程的成绩。假设已经把将要查询的学生的学号赋给了主变量givensno，将课程号赋给了主变量givencno。

```
EXEC SQL SELECT Sno，Cno，Grade
        INTO :Hsno，:Hcno，:Hgrade:Gradeid  /*指示变量*/
        FROM  SC
        WHERE Sno=:givensno AND Cno=:givencno；
```

如果Gradeid < 0，不论Hgrade为何值，均认为该学生成绩为空值。

二、非CURRENT形式的增删改语句

- 在UPDATE的SET子句和WHERE子句中可以使用主变量，SET子句还可以使用指示变量

[例4] 修改某个学生选修1号课程的成绩。

```
EXEC SQL UPDATE SC
        SET Grade=:newgrade        /*修改的成绩已赋给主变量*/
        WHERE Sno=:givensno;       /*学号赋给主变量givensno*/
```

非CURRENT形式的增删改语句（续）

[例5] 将计算机系全体学生年龄置NULL值。

```
Sageid=-1；
EXEC SQL UPDATE Student
                SET Sage=:Raise :Sageid
                WHERE Sdept= ‘CS’；
```

将指示变量Sageid赋一个负值后，无论主变量Raise为何值，RDBMS都会将CS系所有学生的年龄置空值 。

等价于：

```
EXEC SQL UPDATE Student
                SET Sage=NULL
                WHERE Sdept= 'CS'；
```

非CURRENT形式的增删改语句（续）

[例6] 某个学生退学了，现要将有关他的所有选课记录删除掉。假设该学生的姓名已赋给主变量stdname。

```
EXEC SQL DELETE
         FROM SC
         WHERE Sno=
             (SELECT Sno
              FROM Student
              WHERE Sname=:stdname)；
```

非CURRENT形式的增删改语句（续）

[例7] 某个学生新选修了某门课程，将有关记录插入SC表中。假设插入的学号已赋给主变量stdno，课程号已赋给主变量couno。

```
gradeid=-1；              /*用作指示变量，赋为负值*/
EXEC SQL INSERT
        INTO SC(Sno，Cno，Grade)
        VALUES(:stdno，:couno，:gr :gradeid)；
```

由于该学生刚选修课程，成绩应为空，所以要把指示变量赋为负值

8.1 嵌入式SQL

1. 嵌入式SQL语句与主语言之间的通信
2. 不使用游标的SQL语句
3. 使用游标的SQL语句
4. 小结

8.1.4 使用游标的SQL语句

- 必须使用游标的SQL语句
  - 查询结果为多条记录的SELECT语句
  - CURRENT形式的UPDATE语句
  - CURRENT形式的DELETE语句

使用游标的SQL语句（续）

- 一、 查询结果为多条记录的SELECT语句
- 二、CURRENT形式的UPDATE和DELETE语句

一、 查询结果为多条记录的SELECT语句

- 使用游标的步骤

1. 说明游标

2. 打开游标

3.推进游标指针并取当前记录

4. 关闭游标

1. 说明游标

- 使用DECLARE语句
- 语句格式

  EXEC SQL DECLARE <游标名> CURSOR
  FOR <SELECT语句>;

- 功能
  - 是一条说明性语句，这时DBMS并不执行SELECT指定的查询操作。

2. 打开游标

- 使用OPEN语句
- 语句格式

  EXEC SQL OPEN <游标名>;

- 功能
  - 打开游标实际上是执行相应的SELECT语句，把所有满足查询条件的记录从指定表取到缓冲区中
  - 这时游标处于活动状态，指针指向查询结果集中第一条记录之前

3.推进游标指针并取当前记录

- 使用FETCH语句
- 语句格式

EXEC SQL FETCH [[NEXT|PRIOR|

FIRST|LAST] FROM] <游标名>

INTO <主变量>[<指示变量>][,<主变量>[<指示变量>]]...;

推进游标指针并取当前记录（续）

- 功能
  - ①按指定方向推动游标指针，②然后将缓冲区中的当前记录取出来送至主变量供主语言进一步处理
  - NEXT|PRIOR|FIRST|LAST：指定推动游标指针的方式
    - NEXT：向前推进一条记录
    - PRIOR：向回退一条记录
    - FIRST：推向第一条记录
    - LAST：推向最后一条记录
    - 缺省值为NEXT

4. 关闭游标

- 使用CLOSE语句
- 语句格式

  EXEC SQL CLOSE <游标名>;

- 功能
  - 关闭游标，释放结果集占用的缓冲区及其他资源
- 说明
  - 游标被关闭后，就不再和原来的查询结果集相联系
  - 被关闭的游标可以再次被打开，与新的查询结果相联系

二、CURRENT形式的UPDATE语句和DELETE语句

- CURRENT形式的UPDATE语句和DELETE语句的用途
  - 面向集合的操作
  - 修改或删除所有满足条件的记录

CURRENT形式的UPDATE语句和DELETE语句(续)

- 使用步骤
  - 用带游标的SELECT语句查出所有满足条件的记录
  - 从中进一步找出要修改或删除的记录
  - 用CURRENT形式的UPDATE语句和DELETE语句修改或删除之
  - UPDATE语句和DELETE语句中的子句：

    WHERE CURRENT OF <游标名>

    表示修改或删除的是最近一次取出的记录，即游标指针指向的记录

CURRENT形式的UPDATE语句和DELETE语句(续)

- 不能使用CURRENT形式的UPDATE语句和DELETE语句：
  - 当游标定义中的SELECT语句带有GROUP BY或ORDER BY子句
  - 该SELECT语句相当于定义了一个不可更新的视图

8.1 嵌入式SQL

1. 嵌入式SQL语句与主语言之间的通信
2. 不使用游标的SQL语句
3. 使用游标的SQL语句
4. 小结

8.1.6 小 结

- 在嵌入式SQL中，SQL语句与主语言语句分工非常明确
  - SQL语句
    - 直接与数据库打交道，取出数据库中的数据。
  - 主语言语句
    - 控制程序流程
    - 对取出的数据做进一步加工处理

小结（续）

- SQL语言是面向集合的，一条SQL语句原则上可以产生或处理多条记录
- 主语言是面向记录的，一组主变量一次只能存放一条记录
  - 仅使用主变量并不能完全满足SQL语句向应用程序输出数据的要求
  - 嵌入式SQL引入了游标的概念，用来协调这两种不同的处理方式

8.2 存储过程

8.2.1 PL/SQL的块结构

8.2.2 变量常量的定义

8.2.3 控制结构

8.2.4 存储过程

8.2.5 小结

8.2.1 PL/SQL的块结构

- PL/SQL：
  - 是编写数据库存储过程（过程性SQL）的一种过程语言
  - SQL的扩展
  - 增加了过程化语句功能
  - 基本结构是块
    - 块之间可以互相嵌套
    - 每个块完成一个逻辑操作

PL/SQL的块结构（续）

- **PL/SQL块的基本结构：**

1.定义部分

DECLARE

------变量、常量、游标、异常等

- 定义的变量、常量等只能在该基本块中使用
- 当基本块执行结束时，定义就不再存在

PL/SQL的块结构（续）

- **PL/SQL块的基本结构(续)：**

2.执行部分

BEGIN

------SQL语句、PL/SQL的流程控制语句

EXCEPTION

------异常处理部分

END;

8.2 存储过程

8.2.1 PL/SQL的块结构

8.2.2 变量常量的定义

8.2.3 控制结构

8.2.4 存储过程

8.2.5 小结

8.2.2 变量常量的定义

1. **PL/SQL中定义变量的语法形式是:**
   - 变量名 数据类型 ［ ［NOT NULL］:=初值表达式］
2. **常量的定义类似于变量的定义:**
   - 常量名 数据类型 CONSTANT :=常量表达式
   - 常量必须给一个值，并且该值在存在期间或常量的作用域内不能改变。如果试图修改它，PL/SQL将返回一个异常。
3. **赋值语句**
   - 变量名称:=表达式

8.2 存储过程

8.2.1 PL/SQL的块结构

8.2.2 变量常量的定义

8.2.3 控制结构

8.2.4 存储过程

8.2.5 小结

8.2.3 控制结构

- **PL/SQL 功能：**

  一、条件控制语句

  二、循环控制语句

  三、错误处理

控制结构（续）

一、 条件控制语句

IF-THEN， IF-THEN-ELSE和嵌套的IF语句

1. IF *condition* THEN
   Sequence_of_statements;
   END IF
2. IF condition THEN
   Sequence_of_statements1;
   ELSE
   Sequence_of_statements2;
   END IF;
3. 在THEN和ELSE子句中还可以再包括IF语句，即IF语句可以嵌套

控制结构（续）

二、循环控制语句

LOOP， WHILE-LOOP和FOR-LOOP

1.最简单的循环语句LOOP

LOOP

Sequence_of_statements;

END LOOP;

多数数据库服务器的PL/SQL都提供EXIT、BREAK或LEAVE等循环结束语句，保证LOOP语句块能够结束。

控制结构（续）

二、循环控制语句（续）

2. WHILE-LOOP

WHILE condition LOOP

Sequence_of_statements;

END LOOP;

- 每次执行循环体语句之前，首先对条件进行求值
- 如果条件为真，则执行循环体内的语句序列。
- 如果条件为假，则跳过循环并把控制传递给下一个语句

3. FOR-LOOP

FOR count IN ［REVERSE］*bound1* … *bound2* LOOP

Sequence_of_statements;

END LOOP;

控制结构（续）

三、错误处理：

- 如果PL/SQL在执行时出现异常，则应该让程序在产生异常的语句处停下来，根据异常的类型去执行异常处理语句
- SQL标准对数据库服务器提供什么样的异常处理做出了建议

8.2 存储过程

- 8.2.1 PL/SQL的块结构
- 8.2.2 变量常量的定义
- 8.2.3 控制结构
- 8.2.4 存储过程
- 8.2.5 小结

8.2.4 存储过程

- PL/SQL块类型：
  - 命名块：编译后保存在数据库中，可以被反复调用，运行速度较快。存储过程和函数是命名块
  - 匿名块：每次执行时都要进行编译，它不能被存储到数据库中，也不能在其他的PL/SQL块中调用

存储过程（续）

- 一、 存储过程的优点
- 二、 存储过程的用户接口
- 三、 游标

存储过程（续）

- 存储过程：由PL/SQL语句书写的过程，经编译和优化后存储在数据库服务器中，使用时只要调用即可。
- 一、存储过程的优点：

  1. 运行效率高
  2. 降低了客户机和服务器之间的通信量

存储过程（续）

- 二、 存储过程的用户接口：

  1. 创建存储过程

  2. 执行存储过程

  3. 删除存储过程

二、 存储过程的用户接口

1. 创建存储过程：

CREATE Procedure 过程名（［参数1，参数2，...］）

AS

<PL/SQL块>；

- 过程名：数据库服务器合法的对象标识
- 参数列表：用名字来标识调用时给出的参数值，必须指定值的数据类型。参数也可以定义输入参数、输出参数或输入/输出参数。默认为输入参数。
- 过程体：是一个<PL/SQL块>。包括声明部分和可执行语句部分

[例11] 利用存储过程来实现下面的应用: 从一个账户转指定数额的款项到另一个账户中。

```
CREATE PROCEDURE TRANSFER(inAccount INT， outAccount INT， amount FLOAT)
   AS
   DECLARE
      totalDeposit FLOAT;
   BEGIN                          /* 检查转出账户的余额 */
      SELECT total INTO totalDeposit FROM ACCOUNT WHERE ACCOUNTNUM=outAccount;
      IF totalDeposit IS NULL THEN    /* 账户不存在或账户中没有存款 */
         ROLLBACK;
         RETURN;
      END IF;
      IF totalDeposit < amount THEN              /* 账户账户存款不足 */
         ROLLBACK;
         RETURN;
      END IF;
      UPDATE account SET total=total-amount WHERE ACCOUNTNUM=outAccount;
                                          /* 修改转出账户，减去转出额 */
      UPDATE account SET total=total + amount WHERE ACCOUNTNUM=inAccount;
                                          /* 修改转入账户，增加转出额 */
      COMMIT;                             /* 提交转账事务 */
   END;
```

存储过程的用户接口（续）

- 重命名存储过程

ALTER Procedure 过程名1 RENAME TO 过程名2;

存储过程的用户接口（续）

❖2. 执行存储过程：

CALL/PERFORM Procedure 过程名(［参数1，参数2，...］)；

- 使用CALL或者PERFORM等方式激活存储过程的执行。
- 在PL/SQL中，数据库服务器支持在过程体中调用其他存储过程

［例12］从账户01003815868转一万元到01003813828账户中。

CALL Procedure TRANSFER(01003813828，01003815868，10000)；

存储过程的用户接口（续）

3. 删除存储过程

DROP PROCEDURE 过程名（）；

三、游 标

- 在PL/SQL中，如果SELECT语句只返回一条记录，可以将该结果存放到变量中。
- 当查询返回多条记录时，就要使用游标对结果集进行处理
- 一个游标与一个SQL语句相关联。
- PL/SQL中的游标由PL/SQL引擎管理

8.2 存储过程

- 8.2.1 PL/SQL的块结构
- 8.2.2 变量常量的定义
- 8.2.3 控制结构
- 8.2.4 存储过程
- 8.2.5 小结

8.2.5 小 结

- 存储过程的优点
  - 经编译和优化后存储在数据库服务器中，运行效率高
  - 降低客户机和服务器之间的通信量

下课了。。。

休息一会儿。。。