数据库系统

第8章 数据库编程

胡 敏

合肥工业大学

jsjxhumin@hfut.edu.cn

uhnim@163.com

第 8 章　数据库编程

概 述

SQL的使用方式

1、交互式：在终端上每输入一条SQL语句，系统立即执行，然后等待用户输入下一条语句。

2、嵌入式（嵌入到某种主语言中使用）：

宿主语言负责：运算、处理、流程控制等

SQL负责：数据库操作

- 为什么要引入嵌入式SQL?
  - SQL语言是非过程性语言

    有些操作对于交互式SQL是不可能的任务。
  - 事务处理应用需要高级语言

    非声明性动作:用户交互、图形化显示数据

概 述

- 将SQL访问数据库的能力，与宿主语言的过程化处理的能力进行综合
- 将SQL语句嵌入宿主语言中
- 引入变量的使用

8.1 嵌入式SQL

8.1.1 嵌入式SQL的处理过程

8.1.2 嵌入式SQL语句与主语言之间的通信

8.1.3 不使用游标的SQL语句

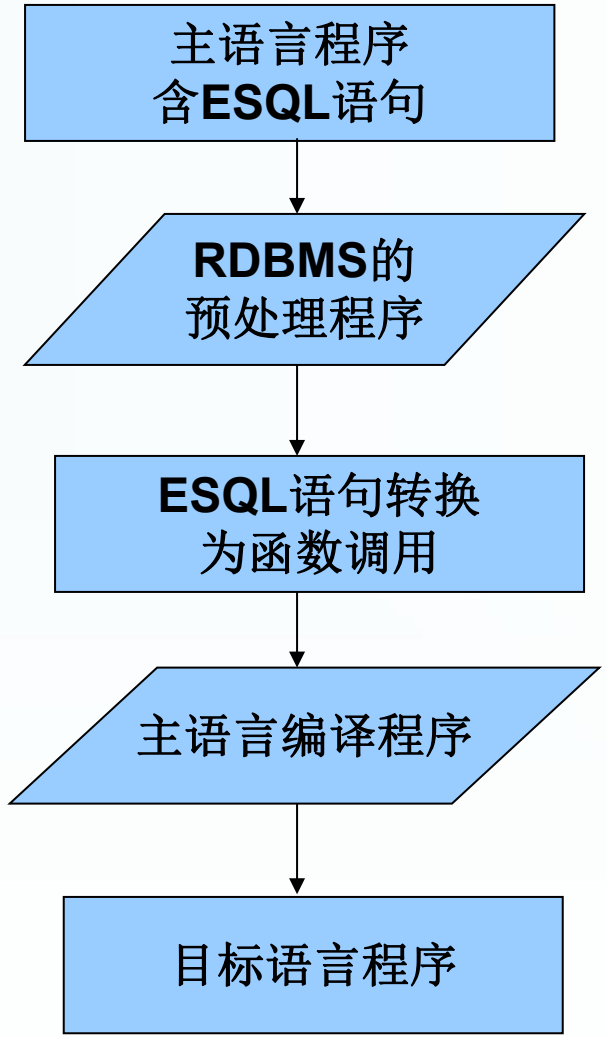
8.1.4 使用游标的SQL语句

8.1.5 动态SQL

8.1.6 小结

8.1.1 嵌入式SQL的处理过程

- 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。
- 处理过程
  - 预编译方法



1．由RDBMS的预处理程序对源程序进行扫描，识别出SQL语句

2．把它们转换成主语言调用语句，以使主语言编译程序能识别它

3．最后由主语言的编译程序将整个源程序编译成目标码。

ESQL基本处理过程

嵌入式SQL的使用规定

- 1、在程序中要区分SQL语句与宿主语言语句

  嵌入的SQL语句以EXEC SQL开始，以分号(;) 或END_EXEC结束（根据具体语言而定）。

  EXEC SQL *〈语句〉* END_EXEC

  举例：

```
EXEC SQL    delete from s
            where  sno='s10';
```

嵌入式SQL的使用规定

2、允许嵌入的SQL语句引用宿主语言的程序变量（共享变量），但有以下规定：

（1)宿主变量出现于SQL语句中时，前面加（:）以区别数据库变量（列名）

宿主变量可出现的地方：SQL的数据操纵语句中可出现常数的任何地方，select等语句的into字句中。

示例：

```
EXEC SQL    select  SNAME ,AGE
            into    :stu_name , :age
            from    s
            where   SNO = :input_no ;
```

共享变量是SQL和宿主语言的接口

(2)共享变量的用法：先由宿主语言的程序定义，并用SQL的DECLARE语句说明

```
EXEC SQL BEGIN DECLARE SECTION
        int   stu_no;
        char stu_name[30];
        int   age;
       char  SQLSTATE[6]
 EXEC SQL END DECLARE SECTION      EXEC SQL
select  sname , age
                 into    :stu_name ,:age
from    s
                 where  sno= :stu_no ;
```

8.1 嵌入式SQL

8.1.1 嵌入式SQL的处理过程

8.1.2 嵌入式SQL与主语言的通信

8.1.3 不使用游标的SQL语句

8.1.4 使用游标的SQL语句

8.1.5 动态SQL

8.1.6 小结

8.1.2 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）

- 数据库工作单元与源程序工作单元之间的通信：

1. SQL通信区（SQL Communication Area，简称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(host variable）
   - 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（cursor）
   - 将SQL语句查询数据库的结果交主语言进一步处理
   - 解决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反馈给应用程序信息
    - ✓ 描述系统当前工作状态
    - ✓ 描述运行环境
  - 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

一、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

一、SQL通信区

- SQL规定，SQLSTATE是一个特殊的共享变量，起着解释SQL语句执行状况的作用，由5个字符组成的字符数组的标准（ISO）返回信息码。
- SQLSTATE:
  - 00000:成功
  - 非0 :出错
  - 02000:未找到元组
- 根据SQLSTATE的值可以控制程序的流向。

嵌入式SQL语句与主语言的通信

- 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
    - 主语言语句
      1. 控制程序流程
      2. 对SQL语句的执行结果做进一步加工处理
    - SQL语句用主变量从主语言中接收执行参数，操纵数据库
  - SQL语句的执行状态由DBMS送至SQLCA中
  - 主语言程序从SQLCA中取出状态信息，据此决定下一步操作
  - 如果SQL语句从数据库中成功地检索出数据，则通过主变量传给主语言做进一步处理
  - SQL语言和主语言的不同数据处理方式通过游标来协调

二、主变量

- 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）

（1）主变量的类型

  - 输入主变量
  - 输出主变量
  - 一个主变量有可能既是输入主变量又是输出主变量

主变量（续）

（2）指示变量

- 一个主变量可以附带一个指示变量（Indicator Variable）
- 一类SQL变量,它被用来管理与其相关联的宿主变量（即在SQL语句中充当输入或输出的变量）。主要用于处理空值（NULL），定义成2字节的整型，如SHORTINT。在SQL语句中引用时,其前也应加“：”(冒号)，而且必须附在其相关联的宿主变量之后，在C语句中，可独立使用。当指示器变量为-1时，表示空值。

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
    - 为与数据库对象名区别，SQL语句中的主变量名前加冒号（:）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

主变量（续）

- 在SQL语句之外(主语言语句中)使用主变量和指示变量的方法
  - 可以直接引用，不必加冒号

```
EXEC SQL BEGIN DECLARE SECTION
Int       dept_number;
Short     ind_num;
Char      emp_name;

EXEC SQL END DECLARE SECTION
Scanf("90d %s", & dept- number , dept – name );

If (dept_number ==0)
     Ind_num =-1;
Else
     Ind_num =0;
EXEC SQL INSERT INTO DEPT (DEPTNO, DNAME)VALUES(:dept_number:ind_num , :dept_name);
```

其中Ind_num是dept_number的指示器变量。当输入的dept_number 值是-1时，则向DEPT 表的DEPTNO列插入空值

程序实例

```
#include <stdio.h>
#include <stdlib.h>

EXEC SQL BEGIN DECLARE SECTION; /**主变量说明开始*/
        char HSno[9];
         char HSname[20];
         char HSex[2];
         int HSage; char
         Hdept[20];
 EXEC SQL END DECLARE SECTION; /**主变量说明结束*/
        long SQLCODE;

  EXEC SQL INCLUDE sqlca;                    /**定义SQL通信区*/
  int main()                                 /**c语言主程序开始 */
  {

 printf("Please input the sno:");/**输入要查找的学生学号 */
 scanf("%s", HSno);

   /**  连接数据库（数据库名为test，主机名为localhost，
  端口号为54321'、用户名密码为SYSTEMmanager */
EXEC SQL CONNECT TO TEST@localhost:54321 AS CONN1
                USER "SYSTEM“ / “manager";

EXEC SQL     SELECT
             Sno,Sname,Ssex,Sage,Sdept
             INTO:Hsno,:Hname,:Hsex,:Hage,:Hde
             pt FROM  Student
             WHERE Sno=:HSno;

if (sqlca.sqlcode == 0)
 {
          printf("\n% - 9s% - 20s% - 2s% - 4s% - 20s\n",
"Sno", "Sname", "Ssex", "Sage", "Sdept");
           printf("% - 9s% - 20s% - 2s% - 4s% - 20s\n", Hsno,
Hname, Hsex, HSage, Hdept);
                }

EXEC SQL DISCONNECT CONN1;
return 0;
}
```

8.1 嵌入式SQL

8.1.1 嵌入式SQL的处理过程

8.1.2 嵌入式SQL语句与主语言之间的通信

8.1.3 不使用游标的SQL语句

8.1.4 使用游标的SQL语句

8.1.5 动态SQL

8.1.6 小结

8.1.3 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
  - 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
- SQL的集合处理方式与宿主语言单记录处理方式之间的协调

  SQL：一次一集合。

  C语言：一次一记录。

  游标：在查询结果的记录集合中移动的指针。

  若一个SQL语句返回单个元组，则不用游标。

  若一个SQL语句返回多个元组，则使用游标。

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

（一）不需游标的SQL语句

- 说明性语句
- 数据定义语句
- 数据控制语句

最简单的一类语句，不需返回结果，不使用主变量，在主语言中只需加前缀EXEC SQL和语句结束符即可。

- 查询结果为单记录的SELECT语句
- 非CURRENT形式的UPDATE语句
- 非CURRENT形式的DELETE语句
- INSERT语句

一般均使用主变量

1、说明、数据定义、数据控制语句

见教材P137-138

（一）不需游标的SQL语句

2、查询结果为单记录的SELECT语句

例　根据主变量的值查找学生的信息

```
EXEC SQL SELECT S#，C#，G
    INTO :SNO，:CNO，:G :GID
    FROM SC
    WHERE S#= :GS AND C#=:GC;
```

查询结果存入这三个主变量中

指示变量，<0说明取得的G为空值

- SELECT语句的INTO、WHERE、HAVING子句中可使用主变量
- 可在INTO子句中使用指示变量，以指明某字段是否空值
- 若SQLCODE=100，说明没有满足条件的记录
- 当查询到的记录多于1条时，在SQLCODE中返回错误信息

3、非CURRENT形式的UPDATE语句

例 将计算机系全体学生的成绩置为空值

```
GID = -1;
EXEC SQL UPDATE SC
    SET G = :GG :GID
    WHERE “CS”=
        (SELECT SD
          FROM  S
          WHERE S.S#=SC.S#);
```

这里使用了值为负的指示变量GID，主变量GG可为任意值

- WHERE、SET子句中可以使用主变量，同时SET子句中还可以使用指示变量
- 通过检查SQLCA的值，判别更新是否成功

4、非CURRENT形式的DELETE语句

例　学号在主变量X1和X2之间的学生已毕业，删除他们的信息。

先删除他们的选课信息

```
EXEC SQL DELETE
     FROM SC
     WHERE S# BETWEEN :X1 AND :X2;
```

再删除他们的基本情况信息

```
EXEC SQL DELETE
     FROM  S
    WHERE S# BETWEEN :X1 AND :X2;
```

WHERE子句中可使用主变量

5、INSERT语句

例 插入一条学生选课记录

```
GID = -1;
EXEC SQL INSERT
    INTO SC（S#，C#，G）
    VALUES（:SNO, :CNO, :GG :GID）;
```

欲插入记录的值由主语言存放在三个主变量中，GID指示成绩字段值为空值，GG可为任意值

8.1 嵌入式SQL

8.1.1 嵌入式SQL的处理过程

8.1.2 嵌入式SQL语句与主语言之间的通信

8.1.3 不使用游标的SQL语句

8.1.4 使用游标的SQL语句

8.1.5 动态SQL

8.1.6 小结

8.1.4 使用游标的SQL语句

- 下列情况必须使用游标
- 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

需要游标的数据操作

- 当select语句的结果中包含多个元组时，使用游标可以逐个存取这些元组

活动集：select语句返回的元组的集合。

当前行：活动集中当前处理的那一行。游标即是指向当前行的指针。

游标分类：

滚动游标：游标的位置可以来回移动，可在活动集中取任意元组。

非滚动游标：只能在活动集中顺序地取下一个元组。

更新游标：数据库对游标指向的当前行加锁，当程序读下一行数据时，本行数据解锁，下一行数据加锁。

需要游标的数据操作

游标的使用方法：

1、定义游标：定义一个游标，使之对应一个select语句。

EXEC_SQL　　DECLARE〈游标名〉CURSOR FOR〈SELECT语句〉
END_EXEC

- 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：打开一个游标，执行游标对应的查询，结果集合为该游标的活动集。

EXEC SQL OPEN〈游标名〉　　//执行查询

END_EXEC　　//定位于第一行的前一行

需要游标的数据操作

3、游标推进语句：在活动集中将游标移到特定的行，并取出该行数据放到相应的宿主变量中

EXEC SQL FETCH [NEXT|PRIOR|FIRST|LAST] FROM <游标名> INTO <共享变量名> END_EXEC

注意：

（1）打开游标时，游标中当前行的位置逻辑上应位于第一行。这使不同的提取选项具有下列行为，如果这是打开游标后的第一次提取操作：

FETCH FIRST：提取游标中的第一行。

FETCH NEXT：提取游标中的第一行。

FETCH PRIOR：不提取行。

（2）Transact-SQL 游标限于一次只能提取一行。

需要游标的数据操作

3、游标关闭语句：关闭游标，释放活动集及其所占资源，使它不再和查询结果相联系。需要再使用该游标时，执行open语句。

EXEC SQL CLOSE <游标名> END_EXEC

4. 释放游标

游标结构本身也会占用一定的计算机资源，所以在使用完游标后，应该将游标释放，以回收资源。

语句:

DEALLOCATE cursor_name

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 deptname[64];
        char HSno[64];
        char HSname[64];
        char HSsex[64];
        int         HSage;
        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*/
```

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);
                                              /*打印查询结果*/
     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*/
           EXEC SQL UPDATE Student                    /*嵌入式SQL*/
                    SET Sage = :NEWAGE
                    WHERE CURRENT OF SX ;
          }              /*对当前游标指向的学生年龄进行更新*/
}

    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*/
}
```

嵌 入 式 SQL

- 8.1.1 嵌入式SQL的处理过程
- 8.1.2 嵌入式SQL语句与主语言之间的通信
- 8.1.3 不使用游标的SQL语句
- 8.1.4 使用游标的SQL语句
- 8.1.5 动态SQL
- 8.1.6 小结

8.1.5 动态SQL

- 静态嵌入式SQL
  - 静态嵌入式SQL语句能够满足一般要求
  - 无法满足要到执行时才能够确定要提交的SQL语句
- 动态嵌入式SQL
  - 允许在程序运行过程中临时“组装”SQL语句
  - 支持动态组装SQL语句和动态参数两种形式

动态SQL简介（续）

- 一、使用SQL语句主变量
- 二、动态参数

一、使用SQL语句主变量

- SQL语句主变量：
  - 程序主变量包含的内容是SQL语句的内容，而不是原来保存数据的输入或输出变量
  - SQL语句主变量在程序执行期间可以设定不同的SQL语句，然后立即执行

使用SQL语句主变量（续）

[例9] 创建基本表TEST

```
EXEC SQL BEGIN DECLARE SECTION;
const char *stmt = "CREATE TABLE test(a int);";
                /* SQL语句主变量 */
EXEC SQL END DECLARE SECTION;
... ...
EXEC SQL EXECUTE IMMEDIATE :stmt;
                /* 执行语句 */
```

二、动态参数

- 动态参数
  - SQL语句中的可变元素
  - 使用参数符号(?)表示该位置的数据在运行时设定
- 和主变量的区别
  - 动态参数的输入不是编译时完成绑定
  - 而是通过 (prepare)语句准备主变量和执行(execute)时绑定数据或主变量来完成

动态参数（续）

- 使用动态参数的步骤：

1.声明SQL语句主变量。

2.准备SQL语句(PREPARE)。

EXEC SQL PREPARE <语句名> FROM <SQL语句主变量>;

动态参数（续）

- 使用动态参数的步骤（续）：

3.执行准备好的语句(EXECUTE)

EXEC SQL EXECUTE <语句名> [INTO <主变量表>] [USING < 主变量或常量>];

动态参数（续）

[例10]向TEST中插入元组。

```
EXEC SQL BEGIN DECLARE SECTION;
const char *stmt = "INSERT INTO test VALUES(?);";
                    /*声明SQL主变量 */
EXEC SQL END DECLARE SECTION;
... ...
EXEC SQL PREPARE mystmt FROM :stmt;   /* 准备语句 */
... ...
EXEC SQL EXECUTE mystmt USING 100; /* 执行语句 */
EXEC SQL EXECUTE mystmt USING 200; /* 执行语句 */
```

8.1 嵌入式SQL

8.1.1 嵌入式SQL的处理过程

8.1.2 嵌入式SQL语句与主语言之间的通信

8.1.3 不使用游标的SQL语句

8.1.4 使用游标的SQL语句

8.1.5 动态SQL

8.1.6 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

下课了。。

休息一会儿。。。