2024年春季学期

数据库系统概论

An Introduction to Database Systems

第八章 数据库编程

中国科学技术大学　大数据学院

黄振亚，huangzhy@ustc.edu.cn

回顾SQL

- SQL的优点
  - 1.综合统一
  - 2.高度非过程化
  - 3.面向集合的操作方式
  - 4.以同一种语法结构提供多种使用方式
  - 5.语言简洁，易学易用

- SQL的局限性
  - 查询班级中处于中位的学生的成绩？
  - 查询学生的绩点？

回顾SQL

- 突破SQL语言的局限性
  - 嵌入式SQL：高级程序语言中嵌入SQL
  - 过程化SQL：扩展SQL对于过程控制的能力
  - ODBC：数据库视作数据源

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（Interactive SQL）
    - 独立使用SQL语言进行数据库操作时的SQL使用方法
    - 缺点：只能用于数据库的操作，不能进行数据处理
  - 嵌入式(Embedded SQL)
    - 把SQL语言嵌入程序设计语言即宿主语言中
- 为什么要引入嵌入式SQL
  - SQL语言是非过程性语言 --- 集合性操作语言
  - 高级语言是过程性语言—数据处理
- 这两种方式细节上有差别，在程序设计的环境下，SQL语句要做某些必要的扩充

将SQL语言访问数据库的功能和宿主语言的数据处理功能相结合

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

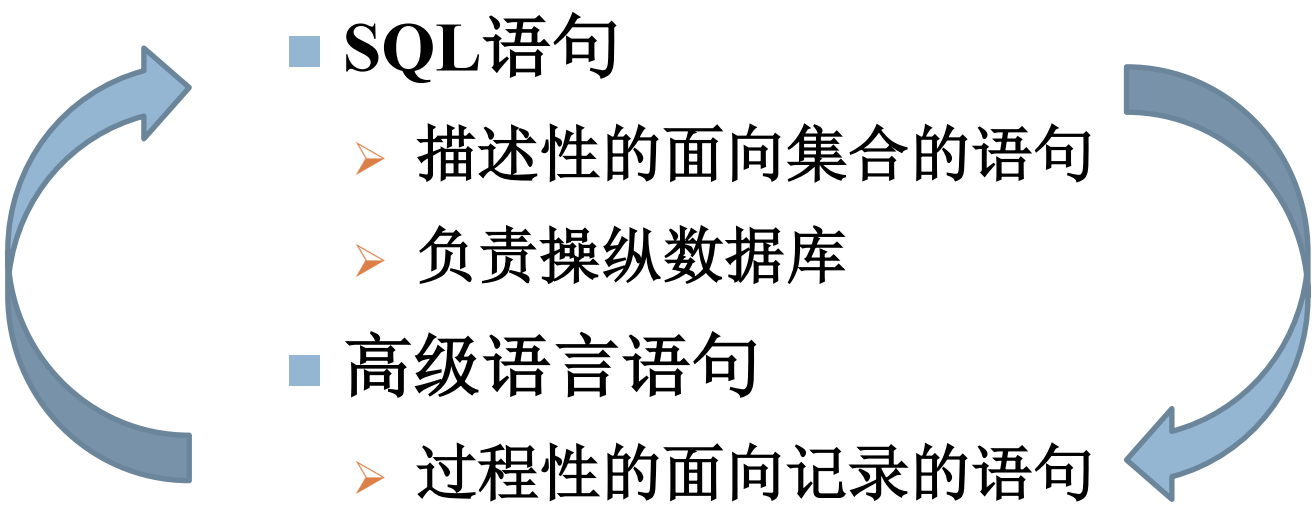
8.1.4 使用游标的SQL语句

8.1.5 动态SQL

8.1.6 小结

8.1.2 嵌入式SQL语句与主语言之间的通信

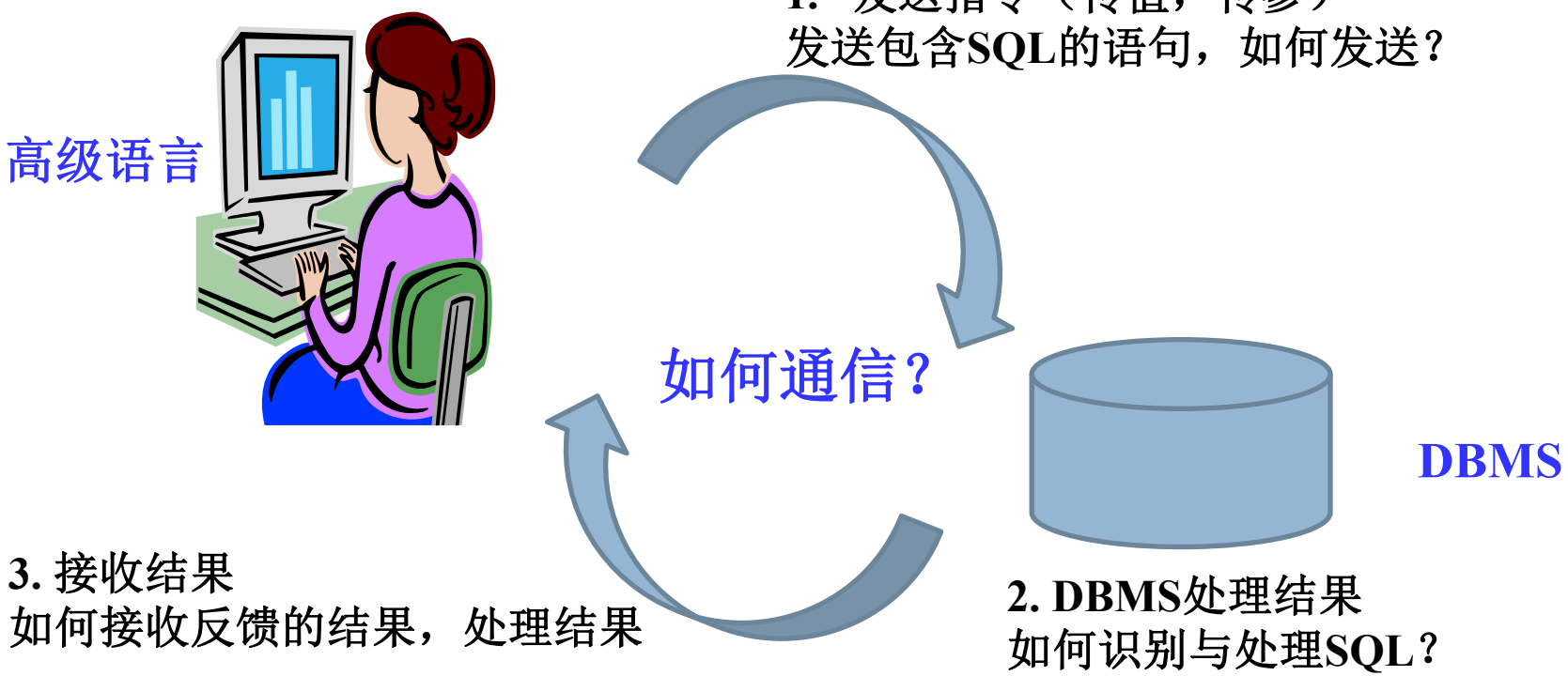
- 将SQL嵌入到高级语言中混合编程，程序中会含有两种不同计算模型的语句



如何通信？

嵌入式SQL

- 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

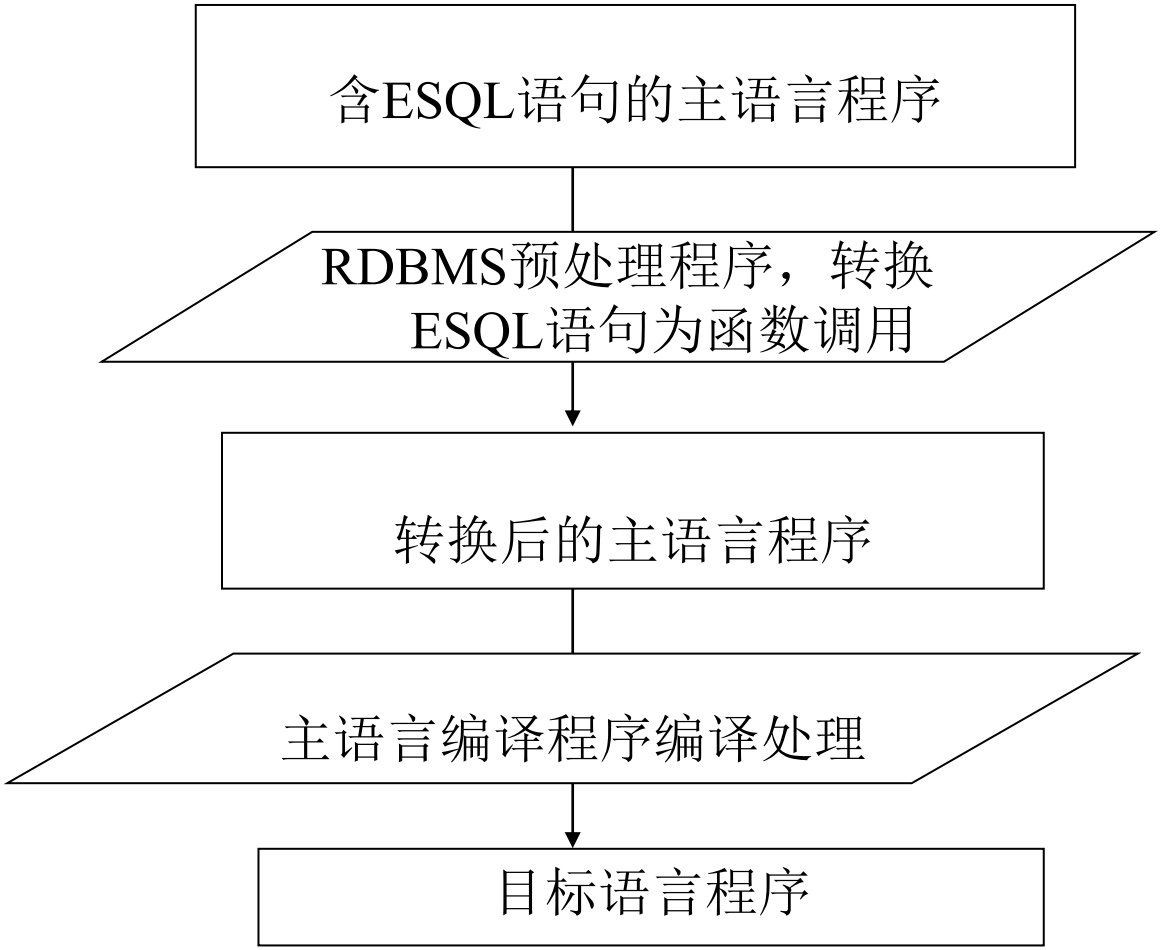


8.1.1 嵌入式SQL的处理过程

- 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、Python，称为宿主语言，简称主语言。
- 处理过程
  - 预编译方法（include 库）
- 为了区分SQL语句与主语言语句， 所有SQL语句必须加前缀
  - C语言中是EXEC SQL，以(;)结束:

    EXEC SQL <SQL语句>;
  - JAVA语言中: #SQL {<SQL语句>}

8.1.1嵌入式SQL的处理过程（续）



ESQL（嵌入式SQL）基本处理过程

一个例子


```
#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“SYSTEM/krms” */
EXEC SQL CONNECT TO TEST@localhost:54321 AS CONN1
USER "SYSTEM“  USING “manager";
```

2. 定义主变量和通信区

```
EXEC SQL    SELECT Sno,Sname,Ssex,Sage,Sdept
            INTO:Hsno,:Hname,:Hsex,:Hage,:Hdept
            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;
}
```

3. 执行SQL语句

4. 关闭数据库连接

1. 打开数据库，连接数据库

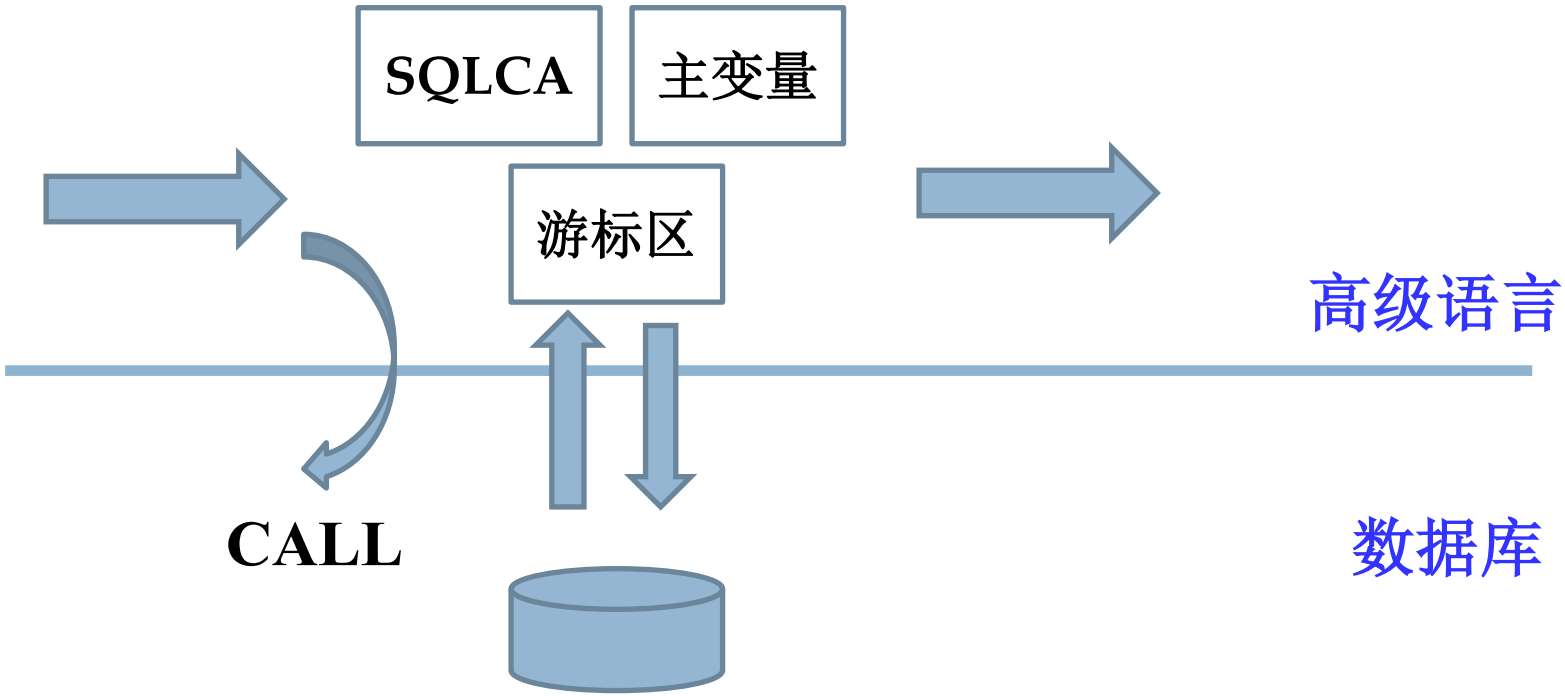
8.1 嵌入式SQL

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. 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，主要用SQL通信区实现（SQLCA）
    - DBMS提供
  - 2. 主语言向SQL语句提供参数，主要用主变量实现
    - DBMS：区分主变量和变量
    - 高级语言：不区分，都是变量
  - 3. 将SQL语句查询数据库的结果交主语言处理，主要用主变量和游标实现
    - 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问题

8.1.2 嵌入式SQL语句与主语言之间的通信

- 在数据库空间和高级语言空间之间的通信，需要额外代价



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

```
struct sqlca_t
{
  char    sqlcaid[8];
  long    sqlabc;
  long    sqlcode;
  struct
  {
    int     sqlerrml;
    char    sqlerrmc[SQLERRMC_LEN];
  }     sqlerrm;
  char    sqlerrp[8];
  long    sqlerrd[6];
  char    sqlwarn[8];
  char    sqlstate[5];
};
```

SQL通信区


```
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;          /*存放执行SQL语句后返回的代码*/
EXEC SQL INCLUDE sqlca;          /*定义SQL通信区*/
```

- **SQLCA使用方法：**
  - **定义SQLCA**
    - **EXEC SQL INCLUDE SQLCA**
  - **使用SQLCA**
    - **SQLCODE：SQLCA中存放每次执行SQL语句后返回代码的变量**
      - **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 语句后，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


```
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;        /*存放执行SQL语句后返回的代码*/
EXEC SQL INCLUDE sqlca;          /*定义SQL通信区*/
```

- 主变量（Host Variable）
  - 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
    - SQL语言中使用的主语言程序变量简称为主变量
- 主变量的类型
  - 输入主变量
    - 应用程序给其赋值，SQL语句引用
  - 输出主变量
    - SQL语句对其赋值或设置状态信息，返回给应用程序
  - 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）

- 指示变量：
  - 一个主变量可以附带一个指示变量（Indicator Variable）
  - 什么是指示变量
    - 一个整形的变量，“指示”所指主变量的值或条件（如，是否为空）
  - 指示变量的用途
    - “指示”所指主变量的值或条件
    - 指示输入主变量是否为空值
    - 检测输出变量是否为空值，值是否被截断

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

```
EXEC SQL BEGIN DECLARE SECTION
        ...
        ...                    （说明主变量和指示变量）
        ...
EXEC SQL END DECLARE SECTION
```

主变量（续）

- 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
  - 为了与数据库对象名（表名、视图名、列名等）区别，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

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

SELECT语句

[例8.3] 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Gradeid*/
                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。

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
  - 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：协调这两种不同的处理方式
- 游标
  - 系统为高级语言开设的一个数据缓冲区，存放SQL语句的执行结果
  - 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
  - 每个游标区都有一个名字：数据区指针

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：
    - 可以是常见的服务器标识串，如<dbname>@<hostname>:<port>
    - 可以是包含服务器标识的SQL串常量
    - 也可以是DEFAULT
  - connect-name是可选的连接名，必须是一个有效的标识符
    - 程序内只有一个连接时：可以不指定连接名
    - 程序内多个连接时：执行时都在该操作提交时所选择的连接上
  - 程序运行过程中可以修改当前（默认）连接：

EXEC SQL SET CONNECTION connection-name | DEFAULT;

- 关闭数据库连接

EXEC SQL DISCONNECT [connection];

8.1.2 嵌入式SQL语句与主语言之间的通信

四个关键步骤

- 1. 打开数据库，连接
- 2. 声明：定义必要的主变量和通信区
- 3. 用SQL访问数据库，并对返回结果进行处理
  - 增删改查语句（SQL）
  - 单个记录，记录集合
- 4. 关闭数据库，释放

8.1 嵌入式SQL

8.1.3 不用游标的SQL语句

- 不需要使用游标的SQL语句的种类
  - 说明性语句
  - 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
  - 非当前游标指向的位置---CURRENT OF SX ;

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8.2] 根据学生号查询学生信息。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;
        /*把要查询的学生的学号赋给为了主变量givensno*/
```

查询结果为单记录的SELECT语句（续）

注意：

(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)查询返回的记录中，可能某些列为空值NULL。为了表示NULL，在Into子句的主变量后面跟着指示变量。若查询结果为空，系统自动将相应指示变量置为负，不再向该主变量赋值（默认主变量为NULL）。

(3) 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

查询结果为单记录的SELECT语句（续）

[例8.3] 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。（主语言给定）

```
EXEC SQL SELECT Sno，Cno，Grade
                INTO :Hsno，:Hcno，:Hgrade :Gradeid
                                    /*指示变量Gradeid*/
                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。

(2)查询返回的记录中，可能某些列为空值NULL。为了表示NULL，在Into子句的主变量后面跟着指示变量。若查询结果为空，系统自动将相应指示变量置为负，不再向该主变量赋值（默认主变量为NULL）。

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8.4] 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] 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] 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8.5] 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;                    /*gradeid用作指示变量，赋为负值*/
EXEC SQL INSERT
            INTO SC(Sno，Cno，Grade)
            VALUES(:stdno，:couno，:gr :gradeid);
                                /*:stdno，:couno，:gr为主变量*/
```

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

- 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

- 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

学号 Sno	姓名 Sname	性别 Ssex	年龄 Sage	所在系 Sdept
201215121	李勇	男	20	CS
201215122	刘晨	女	19	IS
201215123	王敏	女	18	MA
201215125	张立	男	19	IS

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
  - 一条说明性语句，这时DBMS并不执行SELECT指定的查询操作。
  - 设定一个数据区

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：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
    - 高级语言可以进一步操作

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>]]...;

- 功能：指定方向推动游标指针，将缓冲区中的当前记录取出，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

DBMS具体实现方式不一样

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**
  - **UPDATE语句和DELETE语句**
    - 是面向集合的操作
    - 一次修改或删除所有满足条件的记录
  - 如果一次只想修改或删除其中某个记录呢？

CURRENT形式的UPDATE语句和DELETE语句(续)

- 如果只想修改或删除其中某个记录
  - 1.用带游标的SELECT语句查出所有满足条件的记录
  - 2.从中进一步找出要修改或删除的记录
  - 3.用CURRENT形式的UPDATE语句和DELETE语句修改或删除之。即UPDATE语句和DELETE语句中的要用子句：

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

```
例 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*/
```

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UNION或ORDER BY子句**
  - **该SELECT语句相当于定义了一个不可更新的视图**

五、程序实例

[例8.1] 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;          /*存放执行SQL语句后返回的代码*/
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*/
}
```