

数据管理基础

第8章 数据库编程

(8.1 嵌入式SQL)

智能软件与工程学院



□ 课程内容回顾

- 关系模型 & 关系代数
- 关系数据库系统 & SQL语言
 - 数据定义，数据操纵，数据库安全性，数据库完整性
- 关系数据理论 & 数据库设计
 - 关系规范化理论
 - 数据库设计：需求分析、概念设计、逻辑设计、物理设计
 - 数据库的实施与维护

□ 本章内容：数据库编程

- 在应用系统中如何编程访问数据库？
- SQL语言是关系数据库的标准语言，在应用程序中如何使用SQL语言来访问数据库？

□ SQL语言的三种使用方式

- 交互式SQL (Interactive SQL – **ISQL**)
- 嵌入式SQL (Embedded SQL – **ESQL**)
- 过程化SQL (Procedural Language SQL – **PL/SQL**)

	使用方式	应用场景
交互式SQL	命令行 / 批处理	可独立运行，一般供临时用户操作访问数据库用(即席查询, ad-hoc query)
嵌入式SQL	主语言 + ESQL	数据库应用开发
过程化SQL	SQL编程	<ul style="list-style-type: none">• 兼有SQL数据访问和高级程序设计语言的流程控制、简单数值处理功能；• 可在数据库服务器中独立运行；• 常用于编写存储过程、存储函数、触发器。

8.1 嵌入式SQL

8.2 过程化SQL

8.3 存储过程和函数

8.4 ODBC编程

8.5* OLE DB

8.6* JDBC编程

8.7 小结

❑ SQL语言提供了两种不同的使用方式

➤ 嵌入式SQL (ESQL) & 交互式SQL (ISQL)

❑ 为什么要引入ESQL?

- SQL语言是非过程性语言
- 事务处理应用需要高级语言

❑ 这两种使用方式细节上有差别，在程序设计的环境下，SQL语句要做某些必要的扩充

- 如何区分ESQL与用于应用开发的程序设计语言
 - 语句的定界符
 - 主变量 & SQL变量
- 数据交换
 - SQL通信区，游标(cursor)，select ... into ...
- 异常处理定义语句：WHENEVER
- 变量赋值，流程控制语句

为什么要引入嵌入式SQL?

□ SQL语言的目标

- 为最终用户(end-user)提供存取访问数据库的功能

□ SQL语言的不足

- 用户需要知道数据库的模式定义信息：表及其属性定义，视图定义，.....
- 用户需要掌握复杂的SQL语法
- 用户在实际使用过程中可能会存在各种误操作，特别是错误的update, delete等数据更新操作

□ 解决方案：提供嵌入式SQL

- 供应用开发人员(application programmers) 编写数据库应用程序
- 最终用户(end-user) 不再需要书写SQL语句，而是直接通过应用程序来存取访问数据库

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

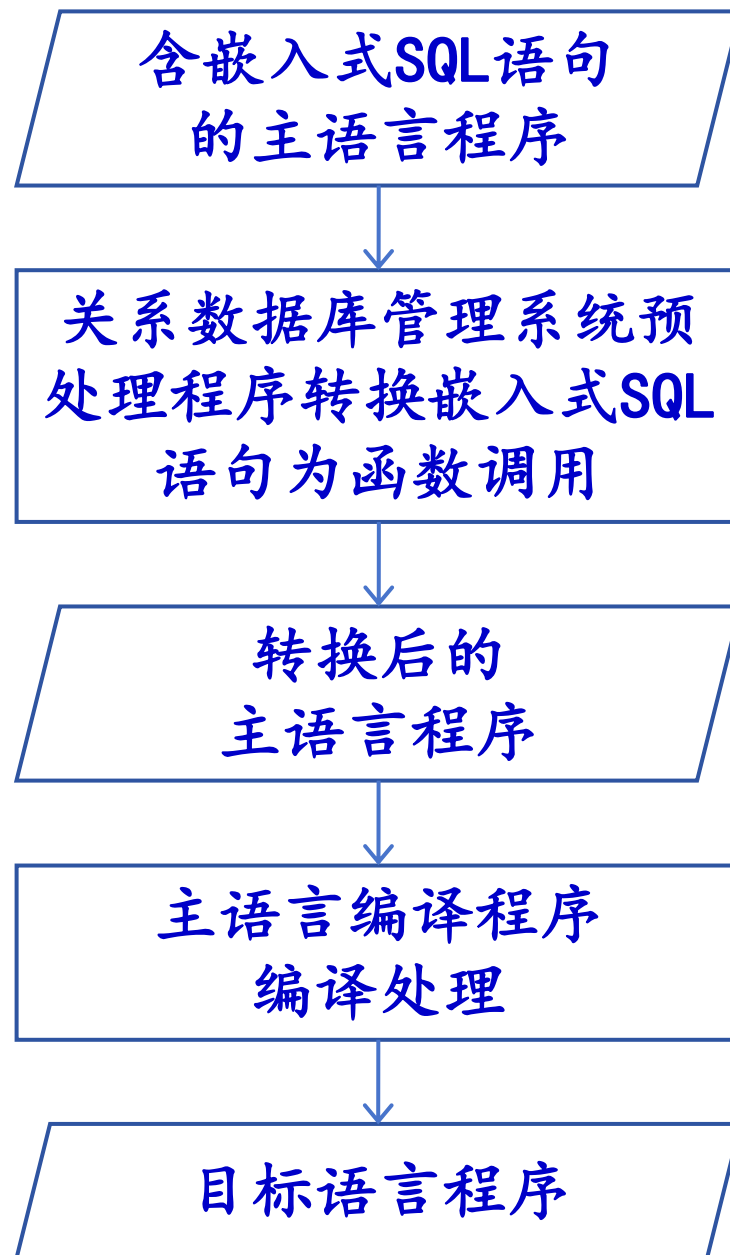
8.1.1 嵌入式SQL的处理过程

❑ 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（图8.1）

- 预编译方法



8.1 嵌入式SQL

❑ 在使用嵌入式SQL进行数据库应用开发时，需要解决下面四个问题：

- 主语言语句与嵌入式SQL语句的区别
- 主语言变量与SQL变量的区别
- 主语言程序与嵌入式SQL间的通讯
- SQL查询语句 与 主语言语句 的结果数据交换

❑ 数据交换模型



主语言语句与嵌入式SQL语句的区别

❑ (在C语言中) 嵌入式SQL语句

- 带有前缀 **EXEC SQL** 和后缀 **;**
- 使用 **into**子句 来获取结果元组值
(**SELECT ... INTO ...** 或 **FETCH ... INTO ...**)
- 用主变量 **:host_var** 保存查询结果元组中的属性值
通过前缀 **:** 来区分主变量和SQL语言中的表名或属性名 (也被称为SQL变量)

```
EXEC SQL select Sno, Sname, Sage  
into :hsno, :hsname, :hsage  
from Student  
where Sno = :givensno ;
```

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

8.1.2 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，程序中含会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，负责控制逻辑流程

它们之间应该如何通信？

❑ 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

② 主语言向**SQL**语句提供执行参数，主要用（输入）**主变量**实现

③ 将**SQL**语句查询数据库的结果交给主语言处理，主要用（输入）**主变量和游标**实现

1. 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，数据库系统反馈给应用程序的信息
 - 描述系统当前工作状态
 - 描述运行环境
- 这些信息将送到并保存在**SQL通信区**中
- 应用程序从**SQL通信区**中获取这些状态信息，据此决定接下来执行的语句

❑ SQLCA使用方法

➤ 定义SQLCA

- 用EXEC SQL INCLUDE SQLCA定义

➤ 使用SQLCA

- SQLCA中有一个用于存放每次执行SQL语句后返回当前SQL语句执行状态的代码变量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。

2. 主变量

□ 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
- 在SQL语句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

- 由应用程序对其赋值，在SQL语句引用

➤ 输出主变量

- 由SQL语句对其赋值或设置状态信息，返回给应用程序

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

输出主变量

输入主变量

□ 指示变量 (Indicator Variable)

- 是一个整型变量，是一种特殊的‘主变量’，用来“指示”相关主变量的值是否为‘空值’
- 一个主变量可以附带一个指示变量
- 指示变量的用途
 - 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

□ 指示变量的取值

0	A database value, not null, was assigned to the host variable.
> 0	A truncated database character string was assigned to the host variable.
-1	The database value is null, and the host variable value is not a meaningful value.

指示变量的应用示例

[例8.3] 指示变量**Gradeid**，用于指示查询返回的**Grade**属性值是否为空

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

[例] 在选课关系**SC**中，根据输入的课程号将该课程的成绩都置为空值
(在主变量与指示变量之间用 空格 或 保留字 **Indicator** 相分隔)

```
gra_ind = -1 ;
EXEC SQL UPDATE SC
          SET Grade = :Hgrade INDICATOR :gra_ind
          WHERE cno = :givencno;
```

主变量&指示变量的使用方法

❑ 在SQL语句中使用主变量和指示变量的方法

➤ 声明（定义）主变量和指示变量

BEGIN DECLARE SECTION

..... /* 声明主变量和指示变量，方法与普通主语言变量一样 */

END DECLARE SECTION

➤ 使用主变量

- 声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

➤ 使用指示变量

- 指示变量前也必须加冒号标志，且必须紧跟在所指主变量之后，用‘空格’或保留字 **Indicator** 相分隔

❑ 在SQL语句之外(主语言语句中)，可以直接使用主变量和指示变量，不必加冒号

主变量&指示变量的声明

- ❑ 为了使用主语言变量，必须首先在**DECLARE SECTION**部分声明这些变量，Why?

```
exec sql begin declare section;  
    char hsno[10], givensno[10], hsname[21];  
    int hsage;  
exec sql end declare section;  
  
.....  
EXEC SQL select Sno, Sname, Sage  
            into :hsno, :hsname, :hsage  
            from Student  
            where Sno = :givensno ;
```

❑ [例3.5] 建立“学生”表Student。

```
CREATE TABLE Student (  
    Sno CHAR(9) PRIMARY KEY,  
    Sname CHAR(20) UNIQUE,  
    Ssex CHAR(2),  
    Sage SMALLINT,  
    Sdept CHAR(20)  
);
```

- ❑ 主语言变量声明语句（**DECLARE SECTION**）作用
 - 在编译时就可以检查主语言变量与其对应属性的数据类型是否一致
 - 为接收从数据库返回的结果值而预先申请内存空间

主变量声明的作用

❑ 主变量声明的作用 (Declare Section)

- ① 在编译时对主语言变量与对应属性进行类型一致性检查
- ② 为接收从数据库返回的结果值而预先申请**足够大**的内存空间

➤ 对于数值类型的属性，常见的主变量类型

- **float cust_discnt;**
- **int cust_discnt;**

} 在数据交换时自动进行类型转换

➤ 对于字符类型的属性

- **in C:** with null terminal character
- **in DB:** no terminator, fixed length

主变量声明的示例

/ 'begin declare' 和 'end declare' 标志着主变量定义的开始与结束，只有在 declare section 中定义的主语言变量，才能被用在ESQL语句中，实现应用程序与数据库之间的数据交换。*/*

exec sql begin declare section; */* 主变量声明开始 */*

/ 字符类型的主变量，需要预先分配好足够大的内存空间（至少比对应的表中属性的最大长度+1，避免在返回查询结果时出现内存溢出错误。*/*

char Deptname[21];

char Hsno[10];

char Hsname[21];

char Hssex[3];

int HSage;

int NEWAGE;

exec sql end declare section; */* 主变量声明结束 */*

3. 游标

□ 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**：非过程化 vs 面向过程。
- **不同数据处理对象**：**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，可用来协调这两种不同的处理方式

□ 游标（Cursor）

- 游标是系统为用户开设的一个数据缓冲区，用于存放**SQL查询**的执行结果
- 每个游标都有一个名字 -- 游标名
- 当游标被打开后，用户可以用**SQL语句**逐一从游标中获取结果记录，并赋给主变量，交由主语言进一步处理

4. 建立和关闭数据库连接

(1) 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

- 可以是一个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，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

➤ 程序运行过程中可以修改当前连接

EXEC SQL SET CONNECTION connection-name | DEFAULT;

(2) 关闭数据库连接

EXEC SQL DISCONNECT [connection-name];

5. 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 DECLARE SECTION; /*主变量声明开始*/
```

```
char Deptname[21];
```

```
char Hsno[10];
```

```
char Hsname[21];
```

```
char Hssex[3];
```

```
int HSage;
```

```
int NEWAGE;
```

```
EXEC SQL END DECLARE SECTION; /*主变量声明结束*/
```

```
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SX*/
        SELECT Sno,Sname,Ssex,Sage /*游标SX对应的查询语句*/
        FROM Student
        WHERE SDept = :deptname;
    EXEC SQL OPEN SX;
    /*打开游标SX, 执行游标对应的查询, 并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更新语句*/
        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*/
}
```

程序实例 (续)

❑ [思考] 在例8.1的嵌入式SQL程序中，还存在什么BUG? (P248~249)

- 只在 **EXEC SQL FETCH SX INTO** 语句之后检查了状态码 **SQLCA.SQLCODE**
- 在其他嵌入式SQL语句之后，都没有检查SQL语句的执行状态码！

❑ 解决方案

- 在程序中添加关于异常处理(Condition Handling)的规则定义

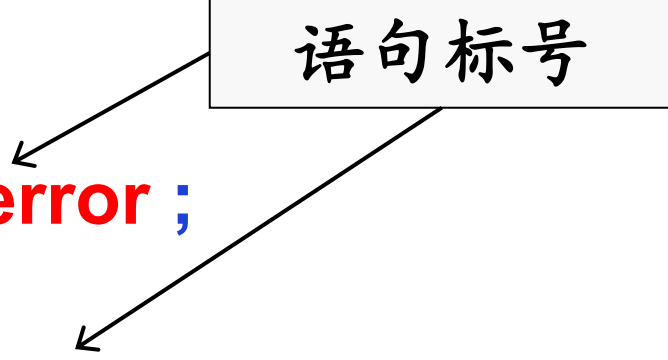
异常处理 (Condition Handling)

- ❑ **whenever** 语句用于定义异常处理办法
- ❑ 最常见的执行异常是: **sqlerror** 和 **not found**
- ❑ 最常用的异常处理办法是:
 - 用goto语句跳转到特定的语句标号处, 进行异常处理
 -

exec sql whenever sqlerror goto report_error ;

exec sql whenever not found goto notfound ;

语句标号



□ Whenever 语句

```
EXEC SQL WHENEVER condition action;
```

- 为SQL语句的执行设置一个‘异常捕捉’机制
- 预编译器在处理过程中，会根据WHENEVER语句的定义，在每一条嵌入式SQL语句之后添加下面的异常状态检查命令：

```
if ( condition ) { action }
```

```
EXEC SQL WHENEVER condition action;
```

❑ CONDITIONS

① **SQLERROR**

- 严重的SQL语句执行错误，出现这类异常系统将终止应用程序的执行。

② **NOT FOUND**

- SQL语句得到成功执行，但并没有访问到任何元组（Select, Fetch, Insert, Update, or Delete等数据操纵语句）
- 捕捉这类异常，通常用于结束循环（如FOR、WHILE等循环），或者改变程序的控制流程（如用于IF、SWITCH等语句）

③ **SQLWARNING**

- 非严重但值得注意的异常，通常不会影响到应用程序的执行
- 相关的warning信息被保存在SQL通讯区 **SQLCA** 中；

```
EXEC SQL WHENEVER condition action;
```

❑ ACTIONS

① CONTINUE

- 忽略相关的异常，不改变程序的控制流程
- 也可用于撤消之前关于condition的异常处理定义

② GOTO label

- 应用程序将跳转到标号label指定处，继续执行

③ STOP

- 立即终止应用程序的执行，同时回滚(rollback)当前事务并撤消与数据库的连接(disconnect)

④ DO function | BREAK | CONTINUE

- 可以用DO function调用一个 C语言函数，函数执行结束后，控制将返回触发该异常的SQL语句之后继续执行；也可以用BREAK或CONTINUE来改变触发该异常的SQL语句所在循环体的执行路径。

交互式访问数据库的程序段

```
exec sql whenever sqlerror goto report_error;  
exec sql whenever not found goto finish;
```

/ whenever 异常处理的定义 */*

```
while (prompt(cid_prompt, 1, cust_id, 4) >= 0)  
{
```

/ 用prompt函数获取用户输入的客
户编号，并保存在主变量cust_id中 */*

```
    exec sql select cname, discnt  
              into :cust_name, :cust_dscnt  
              from customers  
              where cid = :cust_id;
```

/ 通过into子句实现DBMS与应用程
序之间的数据交换(单条结果元组) */*

```
    exec sql commit work;
```

```
    printf("CUSTOMER'S NAME IS  %s AND DISCNT IS  %5.1f\n",  
          cust_name, cust_dscnt);  
    continue;
```

```
notfound:
```

```
    printf("Can't find customer %s, continuing\n", cust_id);  
}
```

程序实例（续）

❑ 使用嵌入式SQL编写的数据库应用程序，如果只考虑对数据库访问部分，其程序结构大致如下：

① **The Declare Section**

- 声明主变量、指示变量、游标

② **Condition Handling**

- 异常处理规则定义

③ **SQL Connect to Database**

- 建立数据库连接

④ **Main Body of Application Program**

- 用主语言编写的应用程序，包括应用界面和数据处理逻辑的实现
- 用ESQL编写的数据库访问语句

⑤ **SQL Disconnect**

- 撤消与数据库的连接

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 不用游标的SQL语句

8.1.4 使用游标的SQL语句

8.1.5 动态SQL

□ 不用游标的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**INSERT**、**DELETE**、**UPDATE**等增删改语句

1. 查询结果为单记录的SELECT语句

❑ 这类语句不需要使用游标，只需用**INTO**子句指定存放查询结果的主变量。

❑ [例8.2] 根据学生号码查询学生信息。

/* 首先，将要查询的学生的学号赋给主变量givensno，然后执行下述查询 */

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

/* 如果执行成功，结果元组被保存在主变量Ssno、Hname.....中 */

❑ 说明：

- **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
- 查询返回的记录中，可能某些列为空值**NULL**
- 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，关系数据库管理系统会在**SQLCA**中返回错误信息

1. 查询结果为单记录的SELECT语句 (续)

❑ 指示变量的使用

- ❑ [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;

IF (Gradeid < 0) {
    ..... /* 学生givensno在课程givencno上的成绩为空值，数据库系统不会在主
           变量Hgrade中写入任何值 */
} ELSE {
    ..... /* 学生givensno在课程givencno上的成绩不为空，且保存在主变量
           Hgrade中 */
}
```

2. 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。

/ 需修改成绩的学生的学号已赋给主变量 givensno */*

/ 新的成绩已赋给主变量 newgrade */*

EXEC SQL UPDATE SC

SET Grade = :newgrade

WHERE Sno = :givensno and Cno = '1' ;

2. 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。

/ 由于该学生刚选修课程，成绩应为空，所以要把成绩指示变量
gradeid赋值为 -1 */*

gradeid = -1; */* gradeid为指示变量 */*

EXEC SQL INSERT

INTO SC(Sno, Cno, Grade)

VALUES(:stdno, :couno, :gr :gradeid);

/ stdno、couno、gr为主变量 */*

- ❑ [思考] 不使用指示变量，如何插入一条成绩为空的选课记录？

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 不用游标的SQL语句

8.1.4 使用游标的SQL语句

8.1.5 动态SQL

❑ 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

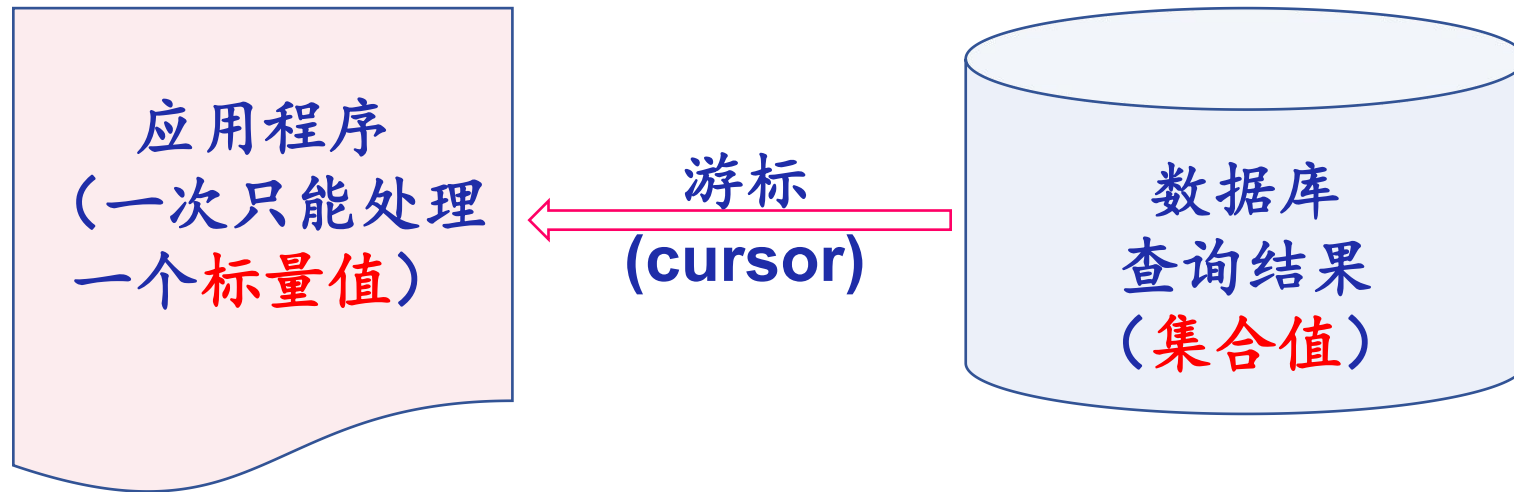
➤ CURRENT形式的DELETE语句

} UPDATE/DELETE当前游标
指向的元组

❑ 当一个查询只会返回单条结果元组，或者不确定该查询是否肯定返回单条结果元组时，也可以使用游标来完成SQL语句与主语言之间的数据交换。

1. 查询结果为多条记录的SELECT语句

- ❑ 在数据交换中，数据库SQL中的变量是集合型的而程序设计语言中的主变量则是标量型，因此数据库中SQL变量不能直接供程序设计语言使用，而需要有一种机制将SQL变量中的集合量逐个取出后送入应用程序主变量内供其使用，而提供此种机制就是游标（cursor）。



- ❑ 基于游标的数据交换特点: **One-Row-at-a-Time Principle**

1. 查询结果为多条记录的SELECT语句（续）

□ 游标（Cursor）操作

(1) 声明游标 **declare a cursor**

- 为某一映像语句(可能返回多个结果元组)的结果集合定义一个命名的游标

(2) 打开游标 **open the cursor**

- 执行相应的映像语句并打开获得的结果集，此时游标处于活动状态并指向结果集合的第一条记录的前面

(3) 推进游标指针并取当前记录 **fetch a row by the cursor**

- 将游标推向结果集合中的下一条记录，读出游标所指向记录的值并赋给对应的主语言变量（One-Row-at-a-Time Principle）

(4) 关闭游标 **close the cursor**

- 关闭所使用的游标，释放相关的系统资源

(1) 声明游标

```
EXEC SQL DECLARE cursor-name CURSOR FOR  
    subquery  
    [ ORDER BY ..... ]  
    [ FOR { READ ONLY |  
            UPDATE [ OF columnname, ..... ] } ] ;
```

- 是一条说明性语句，这时数据库管理系统并不执行SELECT语句；
- 如果一条查询语句的执行返回多条结果元组，那么必须使用游标来获取结果集合中的每一个结果元组；
- 仅当用户确信只可能返回单个结果元组的情况下才可以使用SELECT……INTO……形式的查询语句。

声明游标示例

define the cursor name

```
EXEC SQL DECLARE agent_dollars CURSOR FOR  
  select aid, sum(dollars)  
  from orders  
  where cid = :cust_id  
  group by aid ;
```

means multiple rows in result set

search by customer's id (stored in host variable cust_id)
when open the cursor agent_dollars

(2) 打开游标

- ❑ 打开游标实际上是执行相应的**SELECT**语句，把查询结果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的第一条记录

before open the cursor, you must place cno value of customer's id in the host variable cust_id using in the declare statement of cursor agent_dollars.

.....

EXEC SQL OPEN agent_dollars ;

.....

execute the select statement

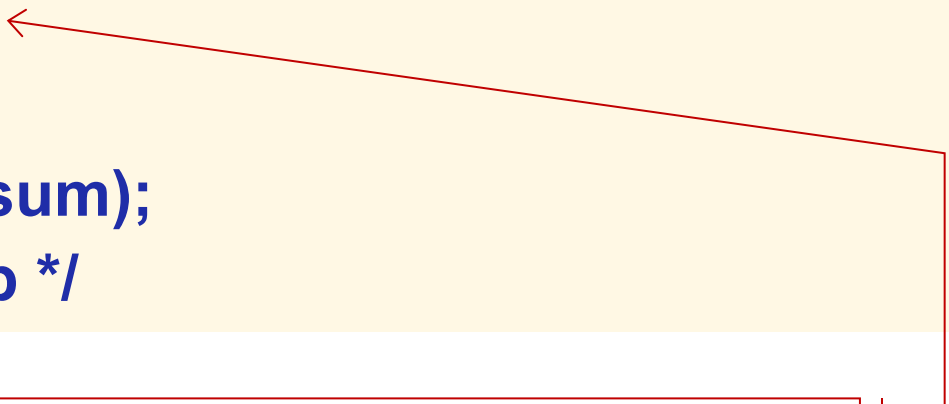
after open the cursor, the pointer of the cursor has been placed in the position before the first row in result set.

(3) 推进游标指针并取当前记录

- ❑ 推动游标指针至下一条记录，同时将缓冲区中当前游标指针指向的记录（即当前记录）取出来送至主变量供主语言进一步处理

```
while (TRUE) {                /* loop to fetch rows */
    exec sql fetch agent_dollars
        into :agent_id, :dollar_sum;

    printf("%s %11.2f\n", agent_id, dollar_sum);
}                             /* end fetch loop */
```



- ① move the pointer of cursor to the next row, then the next row is current row
- ② fetch the current row's value into host variables: agent's id to agent_id, summation of dollars to dollar_sum

结束游标循环

❑ 通过 whenever ... goto ... 定义循环结束方式

declare 'not found' event processing

```
exec sql whenever not found goto finish;
```

.....

```
while (TRUE) {  
    exec sql fetch ..... into .....;
```

.....

```
}
```

.....

```
finish:  exec sql close agent_dollars;
```

execute this statement after fetch loop when
'not found' event is occur

(4) 关闭游标

❑ 关闭游标，释放结果集占用的缓冲区及其他资源

```
EXEC SQL CLOSE agent_dollars ;
```

- ① close the cursor, and release the result set and other resource in DBMS
- ② after close the cursor, it can be opened again

说明：

- 游标一旦被声明(**declare**)，可以被重复使用(**open-fetch-close**)。每一次**open**一个游标，都将重新执行对应的**query**，并生成新的结果集。
- 一个游标结果集只能被遍历一次，其中的结果元组被**fetch**顺序是随机的（游标定义中无**order by**子句）。
- 应用程序可以通过‘游标状态变量’来了解一个游标的当前状态(是否处于打开状态、结果元组的个数、是否**fetch**到结果元组.....)

可滚动游标 (Scrollable Cursors)

- ❑ 如果希望能够重复**FETCH**游标结果集中的元组，可使用**可滚动游标**

```
EXEC SQL DECLARE cursor_name [ INSENSITIVE ] [ SCROLL ]  
CURSOR [ WITH HOLD ] FOR  
    subquery { UNION subquery }  
    [ ORDER BY ..... ]  
[ FOR READ ONLY |  
  FOR UPDATE OF columnname ..... ];
```

```
EXEC SQL FETCH  
    [ { NEXT | PRIOR | FIRST | LAST |  
      { ABSOLUTE | RELATIVE } value_spec } FROM ]  
cursor_name INTO .....;
```

2. CURRENT形式的UPDATE语句和DELETE语句

- ❑ 非**CURRENT**形式的**UPDATE**语句和**DELETE**语句，即常规的、不使用游标的**UPDATE**和**DELETE**操作，其特点是：
 - 面向集合的操作
 - 一次修改或删除所有满足条件的记录
- ❑ 在一个**可更新游标**上，可以使用**CURRENT**形式的**UPDATE**语句和**DELETE**语句，修改或删除当前游标记录
- ❑ **[思考]** 什么是可更新游标？

```
EXEC SQL DECLARE cursor-name CURSOR FOR
    subquery
    [ ORDER BY ..... ]
    [ FOR { READ ONLY |
            UPDATE [ OF columnname, ..... ] } ] ;
```

2. CURRENT形式的UPDATE语句和DELETE语句（续）

□ CURRENT形式的UPDATE语句和DELETE语句的用途

➤ 如果只想修改或删除其中某个记录

- 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或DELETE语句修改或删除之
 - 在UPDATE语句或DELETE语句中要用子句
WHERE CURRENT OF <游标名>
 - 表示修改或删除最近一次取出的记录，即游标指针当前指向的记录

□ 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

例8.1：CURRENT形式的UPDATE语句示例

```
EXEC SQL DECLARE SX CURSOR FOR /*定义游标SX*/
    SELECT Sno,Sname,Ssex,Sage /*游标SX对应的查询语句*/
    FROM Student
    WHERE SDept = :deptname;

.....
EXEC SQL OPEN SX; /*打开游标SX，执行游标对应的查询，并指向查询结果的第一行*/
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 */

    .....
    EXEC SQL UPDATE Student /* 嵌入式SQL更新语句 */
        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*/
```

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 不用游标的SQL语句

8.1.4 使用游标的SQL语句

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
- 但无法满足：到执行时才能够确定要提交的SQL语句、查询的条件

❑ 动态嵌入式SQL

- 在嵌入式SQL编程中，很多时候编程人员无法确定到底应该做什么工作，所使用的SQL语句也不能预先确定，需要根据程序的实际运行情况来决定，也就是根据实际情况来生成并调用SQL语句。这样的SQL语句被称为**动态SQL**。

❑ 动态SQL语句的可变性

- SQL语句正文动态可变
- 变量个数动态可变
- 语句类型动态可变
- SQL语句引用对象动态可变

静态SQL & 动态SQL

	描述	优点	缺点
静态SQL	在应用开发阶段就能够确定下来的SQL语句。	在预编译时，DBMS将对SQL语句进行分析和执行计划优化，可确保数据库访问的正确性。	只能根据缺省参数值进行优化，其访问路径并非是最优路径。
动态SQL	在运行时动态生成的SQL语句。	可以根据运行时数据库的当前状态选择最优访问路径。	动态地进行SQL语句的语法分析和访问路径选择，需要额外的执行开销。

□ 在什么情况下需要使用动态SQL？

- 应用程序需要在执行过程中生成SQL语句；
- SQL语句用到的数据库对象在预编译时不存在；
- 希望SQL语句的执行能够根据执行时的数据库系统内部的统计信息来采用最优的访问策略。

1. 使用**SQL**语句主变量
2. 动态参数
3. 执行准备好的语句 (**EXECUTE**)

1. 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

❑ [例8.6] 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，内容是创建表的SQL语句 */
```

```
EXEC SQL END DECLARE SECTION;
```

```
.....
```

```
EXEC SQL EXECUTE IMMEDIATE :stmt; /*执行动态SQL语句*/
```

2. 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语句**EXECUTE**绑定数据或主变量来完成

❑ 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>;

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>];

□ [例8.7] 向TEST中插入元组。

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

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 嵌入式SQL总结

8.1.6 嵌入式SQL (总结)

1. 主语言语句 与 SQL语句 的区别?

- 对嵌入在主语言中的SQL语句必须加前缀和后缀
- 当主语言是C语言时, 前缀是EXEC SQL, 后缀是END_EXEC 或 分号 ‘;’

2. 主语言程序与ESQL间的通讯

- 向主语言传递SQL语句的执行状态: SQL通讯区 -- SQLCA
- 主语言向SQL语句提供参数: 主变量 (host variable)
- 将SQL语句的查询结果交换给主语言处理: 游标 (cursor) + 主变量

3. 主变量 与 SQL变量的区别?

- 可以在嵌入式SQL语句中使用的主语言变量被称为‘主变量’; SQL语句中的表名或属性名被称为‘SQL变量’, SQL变量只能在嵌入式SQL语句中使用。
- 在嵌入式SQL语句中, 为了区分‘主变量’和‘SQL变量’, 需要在‘主变量’标识符的前面加上一个前缀‘:’, 以便与表名或属性名等‘SQL变量’区别开来。
- 可以通过‘主变量’在嵌入式SQL语句与主语言语句之间交换数据: 可以通过主变量获取查询结果值, 并用于主语言语句中; 也可以将保存在主语言变量中的值用于SQL语句的执行。

4. SQL查询语句 与 主语言语句 的结果数据交换?

- 单条结果元组的数据交换: SELECT ... INTO ... 或 游标
- 多条结果元组的数据交换: 游标

Some Common ESQL Statement

❑ Type Correspondences

Basic SQL type	ORACLE type	DB2 UDB type	C datatype
char(n)	char(n)	char(n)	char arr[n+1]
varchar(n)	varchar(n)	varchar(n)	char array[n+1]
smallint	smallint	smallint	short int
integer, int	integer, int, number(10)	integer, int	int
real	real	real	float
double precision, float	double precision, number, float	double precision, double, float	double

Some Common ESQL Statement

❑ Basic Embedded SQL Select Form (Single-Row Select)

EXEC SQL

SELECT [ALL|DISTINCT] expression,
INTO host-variable [indicator-variable],
FROM tableref [corr-name],
[WHERE search-condition] ;

❑ Embedded SQL Declare Cursor Syntax

EXEC SQL

DECLARE cursor_name CURSOR FOR
subquery
[ORDER BY]
[FOR { READ ONLY |
UPDATE [OF columnname,] }] ;

❑ Embedded Basic SQL Delete Syntax

EXEC SQL

```
DELETE FROM tablename [ corr_name ]  
[ WHERE search_condition |  
  WHERE CURRENT OF cursor_name ]
```

❑ Embedded Basic SQL Update Syntax

EXEC SQL

```
UPDATE tablename [ corr_name ]  
SET columnname = expr, .....  
[ WHERE search_condition |  
  WHERE CURRENT OF cursor_name ]
```

Some Common ESQL Statement

❑ Embedded Basic SQL Insert Syntax

EXEC SQL

INSERT INTO tablename [(column_nme, ...)]

VALUES (expr,) | subquery ;

❑ The Other ESQL Statement

EXEC SQL CREATE TABLE ;

EXEC SQL DROP TABLE ;

EXEC SQL COMMIT WORK ;

EXEC SQL ROLLBACK WORK ;

EXEC SQL CONNECT ;

EXEC SQL DISCONNECT ;

复习思考题

1. 使用嵌入式SQL语言编写数据库应用程序，与数据库访问有关的程序构成是什么？
2. 什么是主变量？什么是指示变量？请简述它们的各自作用。
3. 嵌入式SQL与主语言语句之间的信息交换方式有哪些？
4. 什么是游标(cursor)？请简述游标的使用流程。
5. 在使用嵌入式SQL语句访问数据库的过程中，可能会产生哪几种执行异常？

习言道 | 我们都是奋斗者