第4章 嵌入式SQL语言

第4章 嵌入式SQL语言

4.1 嵌入式SQL语言简述

--交互式SQL语言、嵌入式SQL语言与高级语言

--本章的目标

4.2 嵌入式SQL语言的基本处理技巧

4.3 嵌入式程序中SQL语句的基本操作方式

4.4 动态SQL简介

4.1 嵌入式SQL语言简述

--交互式SQL语言

➢ 交互式SQL

交互式SQL是直接执行SQL语句，一般DBMS都供联机交互工具，可以从前端应用程序，如Oracle的SQL*Plus，直接与SQL服务器上的数据库进行通信。只要把查询输入到应用程序窗口，再执SQL语句，就可以获取查询结果。通过这种方式可以迅速检查数据、验证连接和观察数据库对象。

SQL语句由DBMS来进行解释。

4.1 嵌入式SQL语言简述

--嵌入式SQL语言

➢ 嵌入式SQL

在这种方式中，SQL语句被嵌入到高级语言（宿主语言）中，使应用程序充分利用SQL访问数据库的能力、宿主语言的过程处理能力。

这种方式般需要预编译，将嵌入的SQL语句转换为宿主语言编译器能处理的语句。

4.1 嵌入式SQL语言简述

--交互式SQL语言

- 交互式SQL语言有很多优点
  - 记录集合操作
  - 非过程性操作：指出要做什么，而不需指出怎样做
  - 一条语句就可实现复杂查询的结果
- 然而，交互式SQL本身也有很多局限……

4.1 嵌入式SQL语言简述

--交互式SQL语言的局限(续)

- 现实中有些特别复杂的检索结果难以用一条交互式SQL语句完成，此时需要结合高级语言中经常出现的顺序、分支和循环结构来帮助处理
  - 例如：依据不同条件执行不同的检索操作等

```
If some-condition Then
    SQL-Query1
Else
    SQL-Query2
End If
```

4.1 嵌入式SQL语言简述

--交互式SQL语言的局限(续)

- 现实中有些特别复杂的检索结果难以用一条交互式SQL语句完成，此时需要结合高级语言中经常出现的顺序、分支和循环结构来帮助处理
  - 再如：控制检索操作执行的顺序

```
Do While some-condition
    SQL-Query
End Do
```

4.1 嵌入式SQL语言简述

--交互式SQL语言的局限(续)

- 再如：有时需要在SQL语句检索结果之上再进行处理

```
SQL-Query1
For Every-Record-By-SQL-Query1 Do
    Process the Record
Next

SQL-Query2
If Record-By-SQL-Query2 Satisfy some-condition Then
    Process the Record (condition true)
Else
    Process the Record (condition false)
End If
```

4.1 嵌入式SQL语言简述

--嵌入式SQL语言

- 因此，高级语言+SQL语言
  - 既继承高级语言的过程控制性
  - 又结合SQL语言的复杂结果集操作的非过程性
  - 同时又为数据库操作者提供安全可靠的操作方式：通过应用程序进行操作
- 嵌入式SQL语言
  - 将SQL语言嵌入到某一种高级语言中使用
  - 这种高级语言，如C/C++，Java等，又称宿主语言(HostLanguage)
  - 嵌入在宿主语言中的SQL与前面介绍的交互式SQL有一些不同的操作方式

4.1 嵌入式SQL语言简述

--嵌入式SQL语言与交互式SQL语言

- 交互式SQL语言

  select Sname, Sage from Student where Sname=‘张三’ ;

- 嵌入式SQL语言
  - 以宿主语言C语言为例

    exec sql select Sname, Sage
    into :vSname, :vSage from Student where Sname=‘张三’;
  - 典型特点

    --exec sql引导SQL语句：提供给C编译器，以便对SQL语句预编译成C编译器可识别的语句

    --增加一into子句：该子句用于指出接收SQL语句检索结果的程序变量

    --由冒号引导的程序变量，如: ‘:vSname’, ‘:vSage’
  - 嵌入式SQL还有很多特点，后面将一一介绍。

4.1 嵌入式SQL语言简述

--本章目标

- 理解嵌入式SQL语言的操作方式
- 理解嵌入式SQL语句与宿主语言语句之间的变量交互方式
- 理解宿主语言如何判断SQL语句执行的成功与否：错误捕获处理
- 理解单记录结果与多记录结果(游标方式)处理方式
- 理解动态SQL的概念和应用

第4章 嵌入式SQL语言

4.1 嵌入式SQL语言简述

4.2 嵌入式SQL语言基本处理技巧

--变量的声明与使用

--程序与数据库的连接与断开

--SQL结果的提交与撤消

--游标(Cursor)的基本使用方法(单行检索结果与多行检索结果的处理差异)

--错误捕获与处理

4.3 嵌入式程序中SQL语句的基本操作方式

4.4 动态SQL简介

4.2 嵌入式SQL语言基本处理技巧

--变量的声明与使用

- 在嵌入式SQL语句中可以出现宿主语言语句所使用的变量

```
exec sql select Sname, Sage into :vSname, :vSage from Student
where Sname='张三' ;
```

- 这些变量需要特殊的声明，如下所示：

```
exec sql begin declare section;
        char vSname[10], specName[10]="张三";
        int vSage;
exec sql end declare section;
```

4.2 嵌入式SQL语言基本处理技巧

--变量的声明与使用(续)

- 变量声明和赋值中，要注意宿主程序的字符串变量长度应比字符型字段的长度多1个，因宿主程序的字符串尾部多一个终止符为‘\0‘，而程序中用双引号来描述。(注意宿主程序变量类型与数据库字段类型之间有些是有差异的,有些DBMS可支持自动转换，有些不能)

4.2 嵌入式SQL语言基本处理技巧

--变量的声明与使用(续)

- 声明的变量，可以在宿主程序中赋值，然后传递给SQL语句的where等子句中，以使SQL语句能够按照指定的要求(可变化的)进行检索。

```
exec sql begin declare section;
        char vSname[10], specName[10]="张三";
        int vSage;
exec sql end declare section;
exec sql select Sname, Sage into :vSname, :vSage from Student
where Sname = :specName ;
```

4.2 嵌入式SQL语言基本处理技巧

--变量的声明与使用(续)

- 相应的交互式SQL语句

```
exec sql select Sname, Sage into :vSname, :vSage from Student
where Sname = '张三' ;
```

- 嵌入式比交互式SQL语句灵活了一些：只需改一下变量值，SQL语句便可反复使用，以检索出不同结果。也初步显示了程序式SQL语句的优越之处

4.2 嵌入式SQL语言基本处理技巧

--程序与数据库的连接与断开

- 在嵌入式SQL程序执行之前，首先要与数据库进行连接。不同DBMS，具体连接语句的语法略有差异
- SQL标准中建议的连接语法为：

```
exec sql connect to target-server as connect-name user user-name;
```

或

```
exec sql connect to default;
```

- Oracle中数据库连接：

```
exec sql connect :user_name identified by :user_pwd;
```

- DB2 UDB中数据库连接：

```
exec sql connect to mydb user :user_name using :user_pwd;
```

4.2 嵌入式SQL语言基本处理技巧

--程序与数据库的连接与断开(续)

- 在嵌入式SQL程序执行之后，需要与数据库断开连接
- SQL标准中建议的断开连接的语法为：

```
exec sql disconnect connect-name;
    或
exec sql disconnect current;
```

- Oracle中断开数据库连接：

```
exec sql commit release;
    或
exec sql rollback release;
```

- DB2 UDB中断开数据库连接：

```
exec sql connect reset;
```

4.2 嵌入式SQL语言基本处理技巧

--SQL结果的提交与撤消

- SQL语句在程序执行过程中，必须有提交和撤消语句才能确认其操作结果
- SQL结果的提交：

```
exec sql commit work;
```

- SQL结果的撤消：

```
exec sql rollback work;
```

- 为此，很多DBMS都设计了捆绑提交/撤消与断开连接在一起的语句，以保证在断开连接之前使用户确认提交或撤消先前的工作，例如Oracle中：

```
exec sql commit release; 或
exec sql rollback release;
```

4.2 嵌入式SQL语言基本处理技巧

--典型嵌入式SQL程序的例子

```
#include <stdio.h>
#include “prompt.h”
exec sql include sqlca;
char cid_prompt[ ] = “Please enter customer id: ”;
int main()
{ exec sql begin declare section;
      char cust_id[5], cust_name[14]; float cust_discnt;
      char user_name[20],user_pwd[20];
   exec sql end declare section;
   exec sql whenever sqlerror goto report_error;
   exec sql whenever not found goto notfound;
   strcpy(user_name, “poneilsql”);
   strcpy(user_pwd, “XXXX”);
   exec sql connect :user_name identified by :user_pwd;
```

4.2 嵌入式SQL语言基本处理技巧

--典型嵌入式SQL程序的例子

```
#include <stdio.h>
#include "prompt
exec sql include sqlca;
char cid_prompt[ ] = "Please enter customer id: ";
int main()
{ exec sql begin declare section;
      char cust_id[5], cust_name[14]; float cust_discnt;
      char user_name[20],user_pwd[20];
   exec sql end declare section;
   exec sql whenever sqlerror goto report_error;
   exec sql whenever not found goto notfound;
   strcpy(user_name, "poneilsql");
   strcpy(user_pwd, "XXXX");
   exec sql connect :user_name identified by :user_pwd;
```

SQLCA: SQL Communication Area / 后面介绍

The Declare Section

SQL错误捕获语句/ 后面介绍

SQL Connect

4.2 嵌入式SQL语言基本处理技巧

--典型嵌入式SQL程序的例子(续)

```
while((prompt(cid_prompt,1,cust_id,4)) >=0) {
    exec sql select cname,discnt into :cust_name, :cust_discnt
                from customers where cid=:cust_id;
    exec sql commit work;
    printf("Customer's name is %s and discount
            is %5.1f\n",cust_name,cust_discnt);
    continue;
    notfound: printf("Can't find customer %s,
                        continuing\n",cust_id); }
  exec sql commit release;     return 0;
  report_error:   print_dberror();
  exec sql rollback release;
  return 1;
}
```

4.2 嵌入式SQL语言基本处理技巧

--典型嵌入式SQL程序的例子(续)

```
while((prompt(cid_prompt,1,cust_id,4)) >=0) {
    exec sql select cname,discnt into :cust_name, :cust_discnt
                from customers where cid=:cust_id;
    exec sql commit work;
    printf("Customer's name is %s and discount
            is %5.1f\n",cust_name,cust_discnt);
    continue;
    notfound: printf("Can't find customer %s,
                continuing\n",cust_id); }
exec sql commit release;
report_error:   print_dberror();
exec sql rollback release;
return 1;
}
```

SQL Commit Work

SQL Commit Work and Disconnect

SQL Rollback Work and Disconnect

4.2 嵌入式SQL语言基本处理技巧

--单行结果处理与多行结果处理的差异

Into子句与游标(Cursor)

- 单行检索结果的处理，可以直接使用select…into…
- 嵌入式select语句中引入into子句，只能保留和处理单行结果

```
exec sql select Sno, Sname, Sclass
          into :vSno, :vSname, :vSclass
          from Student
          where Sname=‘张三’ ;
```

4.2 嵌入式SQL语言基本处理技巧

--单行结果处理与多行结果处理的差异

Into子句与游标(Cursor)(续)

- 多行检索结果的处理则需使用游标(Cursor):
  - 游标是指向某检索记录集的指针
  - 通过这个指针的移动，每次读一行，处理一行，再读一行…，直至处理完毕

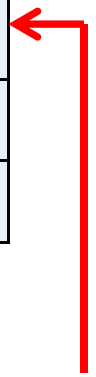
Student

S#	Sname	Sclass
11101	张三大	101
11102	李四大	101
11103	王五大	101
11104	赵六大	101
11105	冯七大	101
11106	张三小	102
11107	李四小	102
11108	王五小	102
11109	赵六小	102
11110	冯七小	102

Select
(记录集)

Result

S#	Sname	Sclass
11101	张三大	101
11102	李四大	101
11103	王五大	101
11104	赵六大	101
11105	冯七大	101



Cursor_name

4.2 嵌入式SQL语言基本处理技巧

--单行结果处理与多行结果处理的差异

Into子句与游标(Cursor)(续)

- 多行检索结果的处理则需使用游标(Cursor):
  - 读一行操作是通过Fetch…into语句实现的：每一次Fetch，都是先向下移动指针，然后再读取
  - 记录集有结束标识EOF，用来标记前面和后面已没有记录了

Student

S#	Sname	Sclass
11101	张三大	101
11102	李四大	101
11103	王五大	101
11104	赵六大	101
11105	冯七大	101
11106	张三小	102
11107	李四小	102
11108	王五小	102
11109	赵六小	102
11110	冯七小	102

Result

S#	Sname	Sclass
11101	张三大	101
11102	李四大	101
11103	王五大	101
11104	赵六大	101
11105	冯七大	101

EOF

Cursor_name

Fetch…into(一次一行)

11103	王五大	101

4.2 嵌入式SQL语言基本处理技巧

--游标(Cursor)的使用方法

- 游标(Cursor)的使用需要先定义、再打开(执行)、接着一条接一条处理，最后再关闭。

```
exec sql declare cur_student cursor for
  select Sno, Sname, Sclass from Student where Sclass='035101' ;
exec sql open cur_student;
exec sql fetch cur_student into :vSno, :vSname, :vSclass;
……
exec sql close cur_student;
```

- 游标可以定义一次，多次打开(多次执行)，多次关闭

4.2 嵌入式SQL语言基本处理技巧

--典型嵌入式SQL程序的例子

```
#define TRUE 1
#include <stdio.h>
#include “prompt.h”
exec sql include sqlca;
exec sql begin declare section;
        char cust_id[5], agent_id[14];  double dollar_sum;
exec sql end declare section;
int main()
{ char cid_prompt[ ]=“Please enter customer ID:”;
  exec sql declare agent_dollars cursor for select aid,sum(dollars)
        from orders where cid = :cust_id group by aid;
  exec sql whenever sqlerror goto report_error;
  exec sql connect to testdb;
  exec sql whenever not found goto finish;
```

4.2 嵌入式SQL语言基本处理技巧

--典型嵌入式SQL程序的例子(续)

```
while((prompt(cid_prompt,1,cust_id,4)) >=0) {
    exec sql open agent_dollars;
    while(TRUE) {
        exec sql fetch agent_dollars into :agent_id, :dollar_sum;
        printf("%s %11.2f\n",agent_id, dollar_sum);}
  finish: exec sql close agent_dollars;
          exec sql commit work;
          exec sql disconnect current;
          return 0;
report_error: print_dberror();
              exec sql rollback;
              exec sql disconnect current;
              return 1;
}
```

4.2 嵌入式SQL语言基本处理技巧

--状态捕获及其处理

- 状态，是嵌入SQL语句的执行状态，尤其指一些出错状态；有时程序需要知道这些状态并对这些状态进行处理
- 嵌入式SQL程序中，状态捕获及处理有三部分构成
  - 设置SQL通信区：一般在嵌入式SQL程序的开始处便设置

    exec sql include sqlca;
  - 设置状态捕获语句：在嵌入式SQL程序的任何位置都可设置；可多次设置；但有作用域

    exec sql whenever sqlerror goto report_error;
  - 状态处理语句：某一段程序以应对SQL操作的某种状态

    report_error: exec sql rollback;

4.2 嵌入式SQL语言基本处理技巧

--状态捕获及其处理(续)

- SQL通信区：SQLCA
  - SQLCA是一个已被声明过的具C结构形式的内存信息区，其中的成员变量用来记录SQL语句执行的状态，便于宿主程序读取与处理
  - SQLCA是DBMS(执行SQL语句)与宿主程序之间交流的桥梁之一

4.2 嵌入式SQL语言基本处理技巧

--状态捕获及其处理(续)

- 状态捕获语句

  exec sql whenever condition action;

- Whenever语句的作用是设置一个“条件陷阱”，该条语句会对其后面的所有由Exec SQL语句所引起的对数据库系统的调用自动检查它是否满足条件(由condition指出)。
  - SQLERROR：检测是否有SQL语句出错。其具体意义依赖于特定的DBMS
  - NOT FOUND：执行某一SQL语句后，没有相应的结果记录出现
  - SQLWARNING：不是错误，但应引起注意的条件

4.2 嵌入式SQL语言基本处理技巧

--状态捕获及其处理(续)

- 如果满足condition, 则要采取一些动作(由action指出)
  - CONTINUE：忽略条件或错误，继续执行
  - GOTO 标号：转移到标号所指示的语句，去进行相应的处理
  - STOP：终止程序运行、撤消当前的工作、断开数据库的连接
  - DO函数或CALL函数：调用宿主程序的函数进行处理，函数返回后从引发该condition的ExecSQL语句之后的语句继续进行

4.2 嵌入式SQL语言基本处理技巧

--状态捕获及其处理(续)

- 状态捕获语句Whenever的作用范围是其后的所有Exec SQL语句，一直到程序中出现另一条相同条件的Whenever语句为止，后面的将覆盖前面的。

```
int main()
{  exec sql whenever sqlerror stop;
   ……
   goto s1
   ……
   exec sql whenever sqlerror continue;
   s1: exec sql update agents
                    set percent = percent + 1;
   ……
};
```

- ✓ S1标号指示的语句受第二个Whenever语句约束。
- ✓ 注意：作用域是语句在程序中的位置，而不是控制流程(因是预编译程序处理条件陷阱)

4.2 嵌入式SQL语言基本处理技巧

--典型DBMS系统记录状态信息的三种方法

➢ 状态记录：sqlcode、sqlca.sqlcode、sqlstate

- sqlcode: 典型DBMS都提供一个sqlcode变量来记录其执行sql语句的状态：

  sqlcode== 0, successful call;

  sqlcode < 0, error, e.g., from connect, database does not exist;

  sqlcode >0, warning, e.g., no rows retrieved from fetch，

- 但不同DBMS定义的sqlcode值所代表的状态意义可能是不同的，需要查阅相关的DBMS资料来获取其含义。

4.2 嵌入式SQL语言基本处理技巧

--典型DBMS系统记录状态信息的三种方法

➢ 状态记录：sqlcode、sqlca.sqlcode、sqlstate

- sqlca.sqlcode: 支持sqlca的产品一般要在sqlca中填写sqlcode来记录上述信息; 除此而外，sqlca还有其他状态信息的记录

  补充知识：
  - sqlca是一个存储区域，将统计和错误从应用程序传递到数据库服务器再传回应用程序的每个数据库请求会使用它；
  - sqlca 用作应用程序到数据库的通信链接的句柄。它会被传递到需要与数据库服务器进行通信的所有数据库库函数中；
  - 它会在所有嵌入式SQL语句上被隐式传递。
- sqlstate: 有些DBMS提供的记录状态信息的变量是sqlstate或sqlca.sqlstate

4.2 嵌入式SQL语言基本处理技巧

--典型DBMS系统记录状态信息的三种方法

- 当我们不需明确知道错误类型，而只需知道发生错误与否，则我们只要使用前述的状态捕获语句

  exec sql whenever condition action;

  即可，而无需关心状态记录变量(隐式状态处理)

- 但我们程序中如要自行处理不同状态信息时，则需要知道状态：

  sqlcode、sqlca.sqlcode、sqlstate

  同时还需知道正确的操作方法(显式状态处理)

4.2 嵌入式SQL语言基本处理技巧

--程序自己进行错误信息的处理

- 不正确的显式状态处理示例

```
exec sql begin declar section;
            char sqlstate[6];
exec sql end declare section;
exec sql whenever sqlerror goto handle_error;
……
exec sql create table custs
                    (cid char(4) not null, cname varchar(13), … … );
if (strcmp(sqlstate, “82100”)==0)
        <处理82100错误的程序>
……
```

- 上述的if 语句是不能被执行的，因为一旦create table发生错误, 则执行handle_error标号后的语句

4.2 嵌入式SQL语言基本处理技巧

--程序自己进行错误信息的处理

- 正确的显式状态处理示例

```
exec sql begin declar section;
            char sqlstate[6];
exec sql end declare section;
exec sql whenever sqlerror goto handle_error;
……
exec sql whenever sqlerror continue;
exec sql create table custs
                    (cid char(4) not null, cname varchar(13), … … );
if (strcmp(SQLSTATE, “82100”)==0)
        <处理82100错误的程序>
……
```

- 上述的if 语句是能被执行的，因为create table发生错误时是继续向下执行的

第4章 嵌入式SQL语言

4.1 嵌入式SQL语言简述

4.2 嵌入式SQL语言的基本处理技巧

4.3 嵌入式程序中SQL语句的基本操作方式

--Select语句的基本操作

--cursor相关语句的基本操作

--Insert、Delete、Update语句的基本操作

--典型的过程性程序示例

4.4 动态SQL简介

4.3 嵌入式程序中SQL语句的基本操作方式

--Select语句的基本操作

- 一种是检索单行结果，
  将结果传送到宿主程序的变量中

```
exec sql select [all | distinct] expression [, expression…]
            into host-variable , [host-variable, …]
            from tableref [corr_name] [ , tableref [corr_name] …]
            where search_condition;
```

例如：

```
exec sql select Sname, Sage      into :vSname, :vSage
            from Student         where Sname = :specName ;
```

- 另一种是检索多行结果，
  则要通过游标来使用select语句

4.3 嵌入式程序中SQL语句的基本操作方式

--Cursor相关语句的基本操作

- Cursor的定义：declare cursor

```
exec sql declare cursor_name cursor for  subquery
    [order by result_column [asc | desc][, result_column …]
    [for [ read only | update [of columnname [, columnname…]]]];
```

例如：

```
exec sql declare cur_student cursor for
    select Sno, Sname, Sclass from Student
    where Sclass= :vClass order by Sno for read only ;
```

- Cursor的打开和关闭： open cursor / close cursor

```
exec sql open cursor_name;
exec sql close cursor_name;
```

4.3 嵌入式程序中SQL语句的基本操作方式

--Cursor相关语句的基本操作

➢ 数据读取：Fetch

```
exec sql fetch cursor_name
        into host-variable , [host-variable, …];
```

例如：

```
exec sql declare cur_student cursor for
    select Sno, Sname, Sclass from Student
    where Sclass= :vClass
    order by Sno for read only ;
exec sql open cur_student;
  ••• •••
exec sql fetch cur_student into :vSno, :vSname, :vSage
  ••• •••
exec sql close cur_student;
```

4.3 嵌入式程序中SQL语句的基本操作方式

--ODBC支持的可滚动Cursor

- 标准的游标始终是自开始向结束方向移动的，每fetch一次，向结束方向移动一次；
- 一条记录只能被访问一次；再次访问该记录只能关闭游标后重新打开

4.3 嵌入式程序中SQL语句的基本操作方式

--ODBC支持的可滚动Cursor

```
exec sql declare cur_student cursor for
        select Sno, Sname, Sclass from Student
        where Sclass= :vClass
        order by Sno for read only ;
exec sql open cur_student;
…
exec sql fetch cur_student into :vSno, :vSname, :vSage
…
exec sql close cur_student;
```

4.3 嵌入式程序中SQL语句的基本操作方式

--ODBC支持的可滚动Cursor

- 可滚动游标是允许游标指针在记录集之间灵活移动、使每条记录可以反复被访问的一种游标。
- 许多实际的DBMS并不支持可滚动游标，但通过ODBC可以使用该功能

- 开放数据库互连（Open Database Connectivity，ODBC）是微软公司开放服务结构中有关数据库的一个组成部分。它建立了一组规范，并提供了一组对数据库访问的标准API（应用程序编程接口）。
- API利用SQL来完成其大部分任务。ODBC本身也提供了对SQL语言的支持，用户可以直接将SQL语句送给ODBC。

4.3 嵌入式程序中SQL语句的基本操作方式

--ODBC支持的可滚动Cursor

- 可滚动游标是允许游标指针在记录集之间灵活移动、使每条记录可以反复被访问的一种游标。

exec sql declare cursor_name [insensitive] **[scroll]** cursor

[with hold] for subquery

[order by result_column [asc | desc][, result_column …]

[for read only | for update of columnname [,columnname]…];

4.3 嵌入式程序中SQL语句的基本操作方式

--ODBC支持的可滚动Cursor

exec sql fetch

[next | prior | first | last | [absolute | relative] value_spec]

from cursor_name into host-variable [, host-variable …];

- next:向结束方向移动一条；
- prior:向开始方向移动一条；
- first:回到第一条；
- last:移动到最后一条；
- absolut value_spec:定向检索指定位置的行，value_spec由1至当前记录集最大值；
- relative value_spec:相对当前记录向前或向后移动，value_spec为正数向结束方向移动，为负数向开始方向移动。

4.3 嵌入式程序中SQL语句的基本操作方式

--ODBC支持的可滚动Cursor

- 可滚动游标移动时需判断是否到结束位置，或到起始位置
  - 可通过判断是否到EOF位置(最后一条记录的后面)，或BOF位置(起始记录的前面)
  - 如果不需区分，可通过whenever not found语句设置来检测

Student

S#	Sname	Sclass
11101	张三大	101
11102	李四大	101
11103	王五大	101
11104	赵六大	101
11105	冯七大	101
11106	张三小	102
11107	李四小	102
11108	王五小	102
11109	赵六小	102
11110	冯七小	102

Select
(记录集)

Result

S#	Sname	Sclass
11101	张三大	101
11102	李四大	101
11103	王五大	101
11104	赵六大	101
11105	冯七大	101

BOF

EOF

Cursor_name

Fetch…into(一次一行)

11103	王五大	101

4.3 嵌入式程序中SQL语句的基本操作方式

--Delete语句的基本操作

- 一种是查找删除(与交互式delete语句相同)，
- 一种是定位删除

exec sql delete from tablename [corr_name]
where search_condition | where current of cursor_name;

例如查找删除：

exec sql delete from customers c where c.city = ‘harbin’ and
not exists (select * from orders o where o.cid = c.cid);

4.3 嵌入式程序中SQL语句的基本操作方式

--Delete语句的基本操作

- 一种是查找删除(与交互式delete语句相同)，
- 一种是定位删除

例如定位删除：

```
exec sql declare delcust cursor for
        select cid from customers c where c.city =‘harbin’ and
                not exists ( select * from orders o where o.cid = c.cid)
                for update of cid;
exec sql open delcust
While (true) {
      exec sql fetch delcust into :cust_id;
      exec sql delete from customers where current of delcust ; }
```

4.3 嵌入式程序中SQL语句的基本操作方式

--Update语句的基本操作

- 一种是查找更新(与交互式Update语句相同)，
- 一种是定位更新

exec sql update tablename [corr_name]
set columnname = expr [, columnname = expr …]
[where search_condition] | where current of cursor_name;

例如查找更新：

exec sql update student s set sclass = ‘035102’
where s.sclass = ‘034101’

4.3 嵌入式程序中SQL语句的基本操作方式

--Update语句的基本操作

- 一种是查找更新(与交互式Update语句相同)，
- 一种是定位更新

例如定位更新：

```
exec sql declare stud cursor for
        select * from student s where s.sclass =‘034101’
        for update of sclass;
exec sql open stud
While (true) {
      exec sql fetch stud into :vSno, :vSname, :vSclass;
      exec sql update student set sclass = ‘035102’
      where current of stud ; }
```

4.3 嵌入式程序中SQL语句的基本操作方式

--Insert语句的基本操作

- 只有一种类型的插入语句

exec sql insert into tablename [(columnname [, columnname, …])]
[values (expr [, expr , …]) | subqurey] ;

例如插入语句：

exec sql insert into student (sno, sname, sclass)
values (‘03510128’, ‘张三’ , ‘035101’) ;

再例如插入语句：

exec sql insert into masterstudent (sno, sname, sclass)
select sno, sname, sclass
from student;

第4章嵌入式SQL语言

4.1 嵌入式SQL语言简述

4.2 嵌入式SQL语言的基本处理技巧

4.3 嵌入式程序中SQL语句的基本操作方式

4.4 动态SQL简介

--动态SQL的概念和作用

--动态SQL的两种执行方式

--更为复杂的动态SQL

4.4 动态SQL简介

--动态SQL的概念和作用

- 动态SQL是相对于静态SQL而言的
- 静态SQL示例：

```
exec sql select Sno, Sname, Sclass
          into :vSno, :vSname, :vSclass
          from Student
          where Sname=‘张三’ ;
```

或

```
exec sql declare cur_student cursor for
          select Sno, Sname, Sclass from Student
          where Sclass= :vClass
          order by Sno for read only ;
```

4.4 动态SQL简介

--动态SQL的概念和作用

- 静态SQL特点：SQL语句在程序中已经按要求写好，只需要把一些参数通过变量(程序中不带冒号，而嵌入式SQL语句中带冒号的变量)传送给SQL语句即可。
- 动态SQL示例：

```
#include <stdio.h>
exec sql include sqlca;
exec sql begin declare section;
      char user_name[ ] = “Scott”;
      char user_pwd[ ] =“tiger”;
      char sqltext[ ] = “ delete from customers where cid = \‘c006\’ ” ;
exec sql end declare section;
```

4.4 动态SQL简介

--动态SQL的概念和作用

➢ 动态SQL示例：（续）

```
int main()
{
  exec sql whenever sqlerror goto report_error;
  exec sql connect :user_name identified by :user_pwd;
  exec sql execute immediate :sqltext;
  exec sql commit release; return 0;
  report_error: print_dberror(); exec sql rollback release; return 1;
};
```

➢ 动态SQL特点：SQL语句可以在程序中动态构造，就像构造字符串一样；如上例sqltext字符串可书写任意SQL语句，只要合法即可。

4.4 动态SQL简介

--动态SQL的概念和作用

➢ 动态SQL示例：（续）

```
#include <stdio.h>
exec sql include sqlca;
exec sql begin declare section;
      char user_name[ ] = “Scott”;
      char user_pwd[ ] =“tiger”;
      char sqltext[ ] = “ delete from customers where cid = \‘c006\’ ” ;
exec sql end declare section;
int main()
{  exec sql whenever sqlerror goto report_error;
    exec sql connect :user_name identified by :user_pwd;
    exec sql execute immediate :sqltext;
    exec sql commit release; return 0;
    report_error: print_dberror(); exec sql rollback release; return 1;
};
```

4.4 动态SQL简介

--动态SQL的概念和作用

- 动态构造SQL语句是应用程序员必须掌握的重要手段
  - 例如：编写一个由用户确定检索条件的应用程序

    请输入 姓名张三_

    年龄在_____和_____之间

    班级035103_

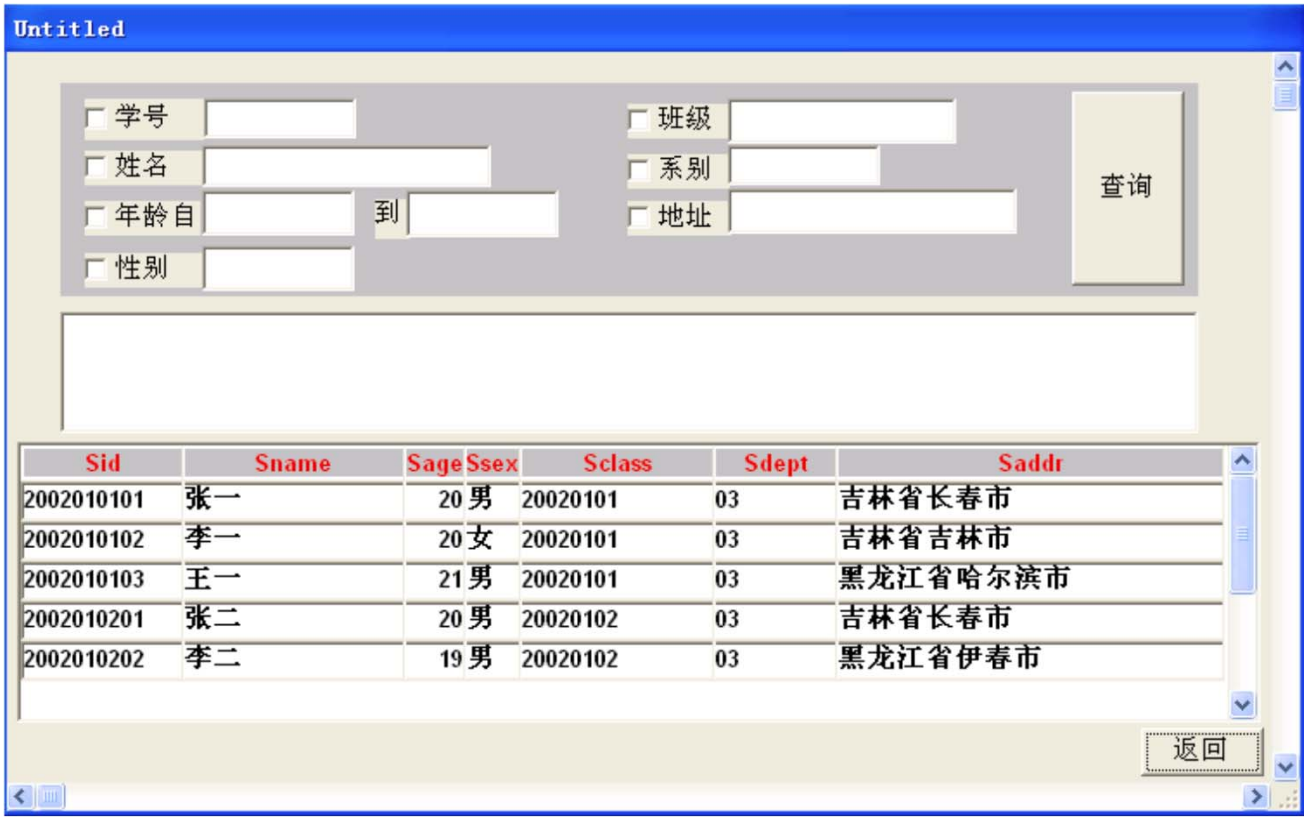
    ……

- 如何实现以上应用程序呢？若用静态方法，则需将上述所有可能条件组合的SQL语句都写出来，则是不现实的；
- 此时需要使用动态SQL语句。

4.4 动态SQL简介

--动态SQL的概念和作用

➢ 示例:



```
string str_temp
int   firstflag = 0
str_temp = ""

// dw_2.setfilter("")
Str_temp = " select * from student where "
if (cbx_id.checked = true and len(trim(sle_id.text))>0) then
   str_temp = str_temp + "(sid like '" + trim(sle_id.text)+ "')"
   firstflag = 1
end if
if (cbx_name.checked = true and len(trim(sle_name.text))>0) then
   if (firstflag = 1) then
      str_temp = str_temp + " and  (sname like '" + trim(sle_name.text)+ "')"
   else
      str_temp = str_temp + " ( sname like '" + trim(sle_name.text)+ "')"
      firstflag = 1
   end if
end if
```

```
if (cbx_sex.checked = true and len(trim(sle_sex.text))>0) then
   if (firstflag = 1) then
      str_temp = str_temp + " and ( ssex = '" + trim(sle_sex.text)+ "' )"
   else
      str_temp = str_temp + " (ssex = '" + trim(sle_sex.text)+ "')"
      firstflag = 1
   end if
end if
if (cbx_class.checked = true and len(trim(sle_class.text))>0) then
   if (firstflag = 1) then
      str_temp = str_temp + " and (sclass like '" + trim(sle_class.text)+ "')"
   else
      str_temp = str_temp + " ( sclass like '" + trim(sle_class.text)+ "')"
      firstflag = 1
   end if
end if
```

```
if (cbx_dept.checked = true and len(trim(sle_dept.text))>0) then
   if (firstflag = 1) then
      str_temp = str_temp + " and (sdept = '" + trim(sle_dept.text)+ "') "
   else
      str_temp = str_temp + " ( sdept like '" + trim(sle_dept.text)+ "')"
      firstflag = 1
   end if
end if
if (cbx_addr.checked = true and len(trim(sle_addr.text))>0) then
   if (firstflag = 1) then
      str_temp = str_temp + " and (saddr like '" + trim(sle_addr.text)+ "') "
   else
      str_temp = str_temp + " (saddr like '" + trim(sle_addr.text)+ "') "
      firstflag = 1
   end if
end if
```

```
if (cbx_agest.checked = true and len(trim(sle_agest.text))>0 ) then
    if (firstflag = 1) then
    str_temp = str_temp + " and ( sage >= " + trim(sle_agest.text) +")"
    if (len(trim(sle_ageed.text))>0 and integer(trim(sle_ageed.text))> Integer(trim(sle_agest.text))) then
        str_temp = str_temp + " and ( sage <= " + trim(sle_ageed.text) +")"
    end if
  else
    str_temp = str_temp + " (sage >= " + trim(sle_agest.text) +")"
    if ( len(trim(sle_ageed.text))>0 and integer(trim(sle_ageed.text))>integer(trim(sle_agest.text))) then
      str_temp = str_temp + " and (sage <= " + trim(sle_ageed.text)+")"
    end if
      firstflag = 1
  end if
end if
//sle_sqltext.text    = str_temp
sqltext = str_temp
exec   sql    execute   immediate   :sqltext
exec sql commit   release
```

4.4 动态SQL简介

--动态SQL的概念和作用

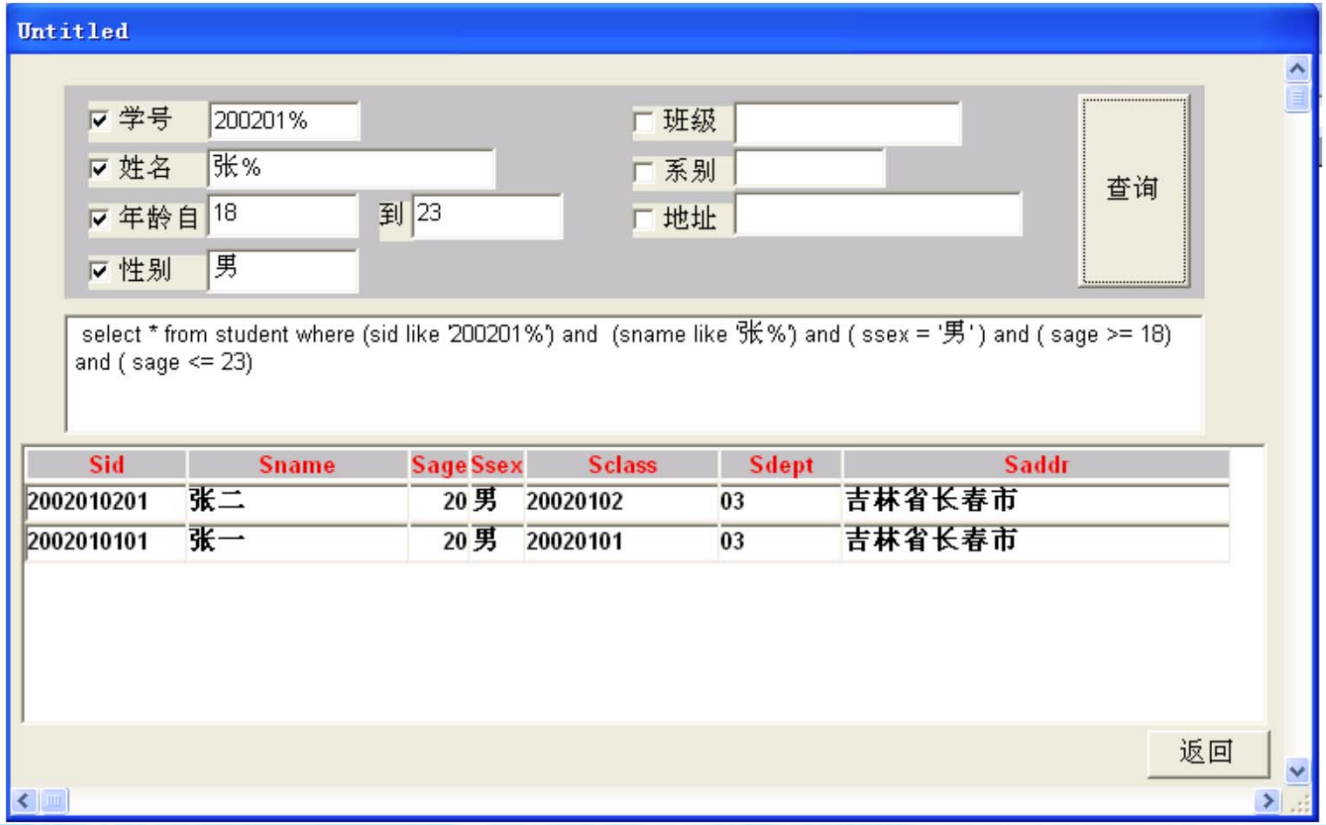
➢ 示例（局部放大）

```
Str_temp = " select * from student where "
if (cbx_id.checked = true and len(trim(sle_id.text))>0) then
    str_temp = str_temp + "(sid like '" + trim(sle_id.text)+ "')"
    firstflag = 1
end if
if (cbx_name.checked = true and len(trim(sle_name.text))>0) then
    if (firstflag = 1) then
        str_temp = str_temp + " and  (sname like '" + trim(sle_name.text)+ "')"
    else
        str_temp = str_temp + " ( sname like '" + trim(sle_name.text)+ "')"
        firstflag = 1
    end if
end if
```

4.4 动态SQL简介

--动态SQL的概念和作用

- 示例结果1:



4.4 动态SQL简介

--动态SQL的概念和作用

➢ 示例结果2:

4.4 动态SQL简介

--动态SQL示例

- 从Customer表中删除满足条件的行

Delete customer rows with ALL of the following properties:

Customer name is Mbale Jamson___
Customer is in city _____
Customer discount is in range from __ to ___
…(and so on)

4.4 动态SQL简介

--动态SQL示例

```
#include  <stdio.h>
#include "prompt.h"
char  Vcname[];
char  Vcity[];
double range_from, range_to;
int Cname_chose, City_chose, Discnt_chose;
Cname_chose = City_chose = Discnt_chose = 0;
int     sql_sign = 0;
char  continue_sign[];

exec  sql  include sqlca;
exec  sql  begin declare section;
        char user_name[20],user_pwd[20];
        char sqltext[]="delete from customers where ";
exec sql end declare section;
```

程序变量申明

------SQLCA: SQL Communication Area

-----The Declare Section

4.4 动态SQL简介

--动态SQL示例

```
int main()
{
    exec sql whenever sqlerror goto report_error;        ------SQL错误捕获语句
    strcpy(user_name, “poneilsql”);
    strcpy(user_pwd, “XXXX”);
    exec sql connect  :user_name  identified              ------SQL Connect
                by  :user_pwd;

  while(1) {
      memset(Vcname, ‘\0’, 20);
      memset(Vcity, ‘\0’, 20);
      if (GetCname(Vcname))                    ------获取Cname值
            Cname_chose = 1;
      if (GetCity(Vcity))                      ------获取City值
            City_chose = 1;
      if (GetDiscntRange(&range_from, &range_to))   ------获取Discnt区间值
            Discnt_chose = 1;
```

4.4 动态SQL简介

--动态SQL示例

```
if(Cname_chose){
      sql_sign = 1;
      strcat(sqltext, “Cname = \’”);
      strcat(sqltext, Vcname);
      strcat(sqltext, “\’”);
 }

if(City_chose){
   sql_sign = 1;
   if(Cname_chose)
       strcat(sqltext, “ and City = \’”);
   else
        strcat(sqltext, “ City = \’”);
   strcat(sqltext, Vcity);
   strcat(sqltext, “\’”);
}
```

-----如果选择了Cname，构造sqltext

-----如果选择了City，构造sqltext

4.4 动态SQL简介

--动态SQL示例

```
if(Discnt_chose){
    sql_sign =1;
    if(Cname_chose=0 and City_chose = 0)
        strcat(sqltext, " discnt >");
    else
        strcat(sqltext, " and (discnt >");
    strcat(sqltext, dtoa(range_from));
    strcat(sqltext, " and discnt<");
    strcat(sqltext, dtoa(range_to));
    strcat(sqltext, ")");
}

if(sql_sign){
    exec sql execute immediate :sqltext;
    exec sql commit work;
}
```

------如果选择了Discnt区间值，构造sqltext

------SQL Commit Work

4.4 动态SQL简介

--动态SQL示例

```
    scanf("continue (y/n) %1s", continue_sign)
    if(continue_sign = "n"){
        exec  sql   commit release;        ------SQL Commit Work and  Disconnect
        returen 0;
    }
  }     ------while 结束

report_error:
  print_dberror();
  exec sql rollback release;        ------SQL  Rollback  Work and  Disconnect
  return 1;
}       ------main 结束
```

4.4 动态SQL简介

--动态SQL的两种执行方式

- 如SQL语句已经被构造在host-variable字符串变量中，则：
  - 立即执行语句：运行时编译并执行

```
exec sql execute immediate :host-variable;
```

  - Prepare-Execute-Using语句：prepare语句先编译，编译后的sql语句允许动态参数，execute语句执行，用using语句将动态参数值传送给编译好的sql语句

```
exec sql prepare sql_temp from :host-variable;
……
exec sql execute sql_temp using :cond-variable
```

4.4 动态SQL简介

--动态SQL的更为复杂的方式

- 例如：编写一个由用户确定检索条件的应用程序

  请输入 姓名张三

  年龄在_____和_____之间

  班级035103

  ……

- 在以上示例中，我们已知被检索的Table名，以及Table中各列名。

  动态性体现在检索条件的可构造方面。

4.4 动态SQL简介

--动态SQL的更为复杂的方式

- 还有一类更复杂的查询：

  例如，如果给定一个表的集合，具体检索哪个表，以及检索该表中的哪些列等等

  都是在程序运行过程中由用户确定的，

  检索条件也是用户临时设置的，

  这时如何书写动态SQL呢？

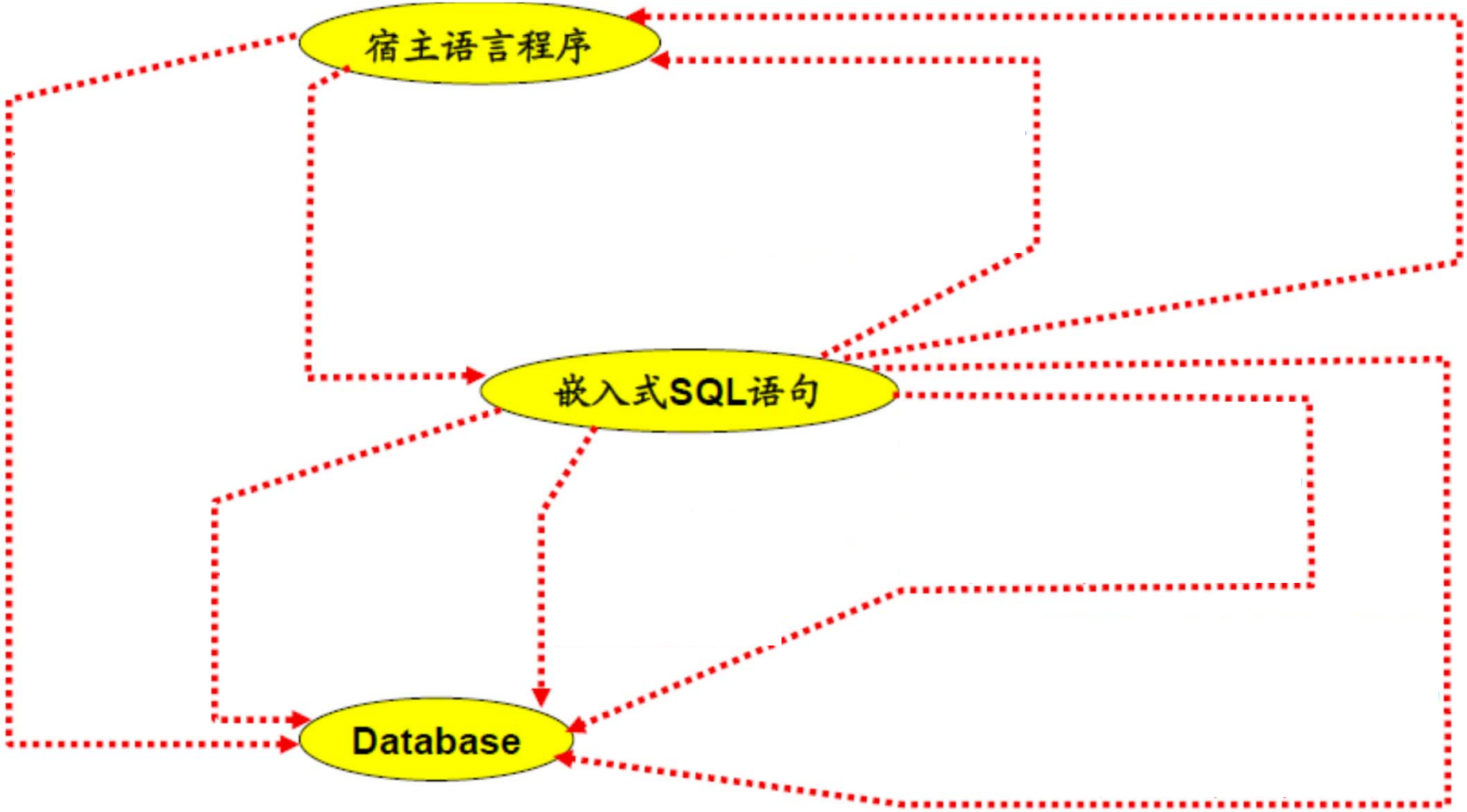
- 这种类型的查询需要涉及DBMS关系模式的定义以及将该定义引入程序中的方法(使程序获知)。

4.4 动态SQL简介

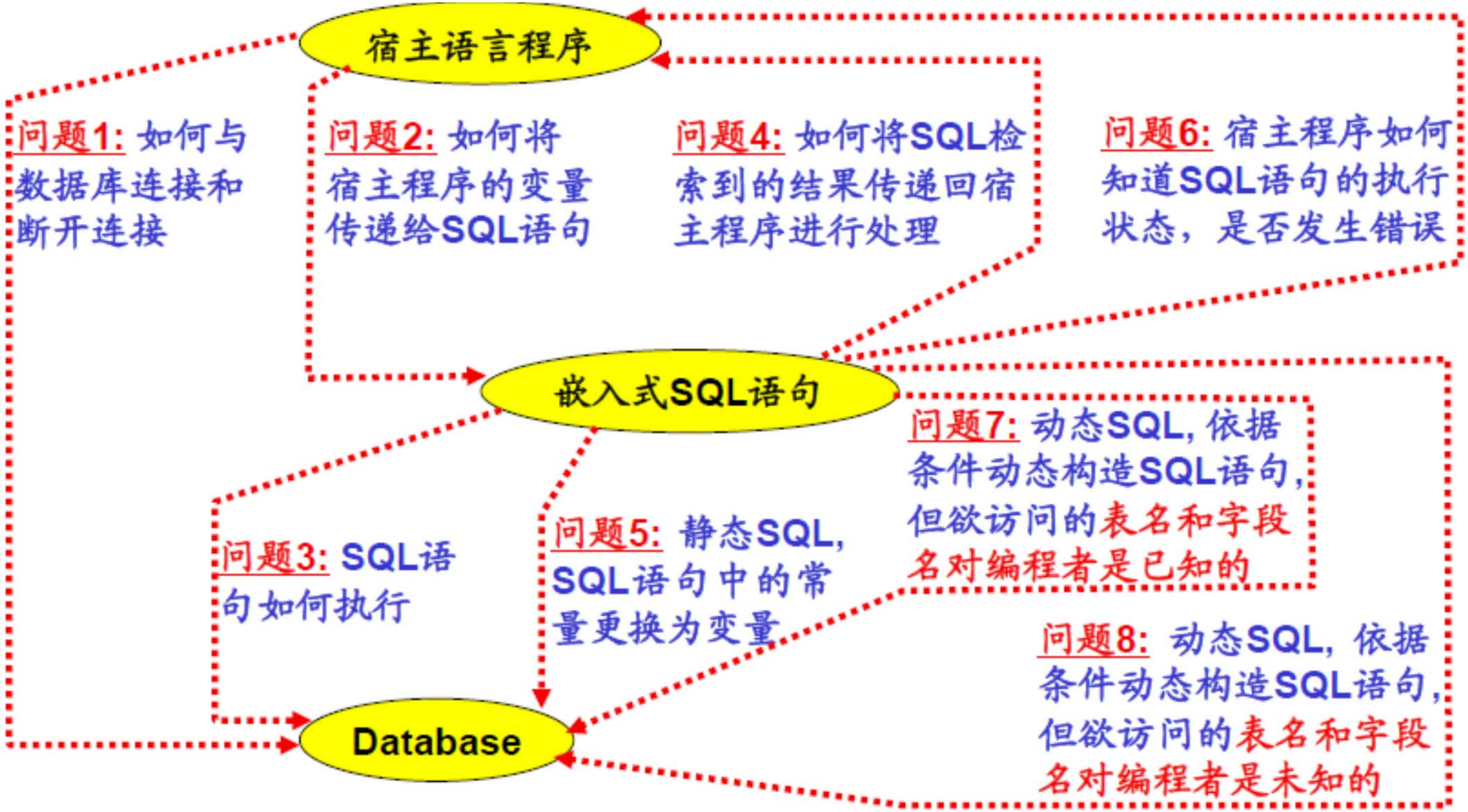
--动态SQL的更为复杂的方式

- 获知关系模式信息可以使用SQLDA(SQL描述符区域)。
- SQLDA是一个内存的数据结构，内可装载关系模式的定义信息，如列的数目，每一列的名字和类型等等。
- 通过读取SQLDA信息可以进行更为复杂的动态SQL的处理。
- 不同DBMS提供的SQLDA格式并不是一致的，教材中提供了Oracle等典型DBMS获取SQLDA及分析SQLDA结构较为详细的内容，可参见教材。

嵌入式SQL语言的有关问题一览



嵌入式SQL语言的有关问题一览



下一章的学习内容

第5章数据库设计

理解如何对应用系统进行需求分析和抽象，设计出正确的数据库模式

----数据库设计的基本概念

---- E-R图/IDEF1X图及其数据库设计

----数据库设计正确性分析

----函数依赖与关系范式

----数据库设计方法和设计过程