

数据库系统之二

--数据库语言



第9讲 嵌入式SQL语言之基本技巧

战德臣

哈尔滨工业大学 教授·博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent
Computing for Enterprises & Services,
Harbin Institute of Technology

本讲学习什么？



基本内容

1. 嵌入式SQL语言概述
2. 变量声明与数据库连接
3. 数据集与游标
4. 可滚动游标与数据库的增删改
5. 状态捕获及错误处理机制

重点与难点

- 数据库语言嵌入到高级语言中使用需要解决的问题—过程及其思维
- 怎样在高级语言中处理数据集—游标的使用技巧
- 错误捕获机制—设置错误陷阱与SQLCA的作用与使用
- 事务的概念—保证数据正确性的机制

嵌入式SQL语言概述

(嵌入式SQL与交互式SQL和高级语言的关系)

战德臣

哈尔滨工业大学 教授·博士生导师
黑龙江省教学名师
教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent
Computing for Enterprises & Services,
Harbin Institute of Technology

➤ 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... ..

```
Select  S#, Avg(Score) From  SC
Where  S# in
        ( Select  S# From  SC
          Where  Score < 60
          Group by  S# Having Count(*)>2 )
Group by S# ;
```

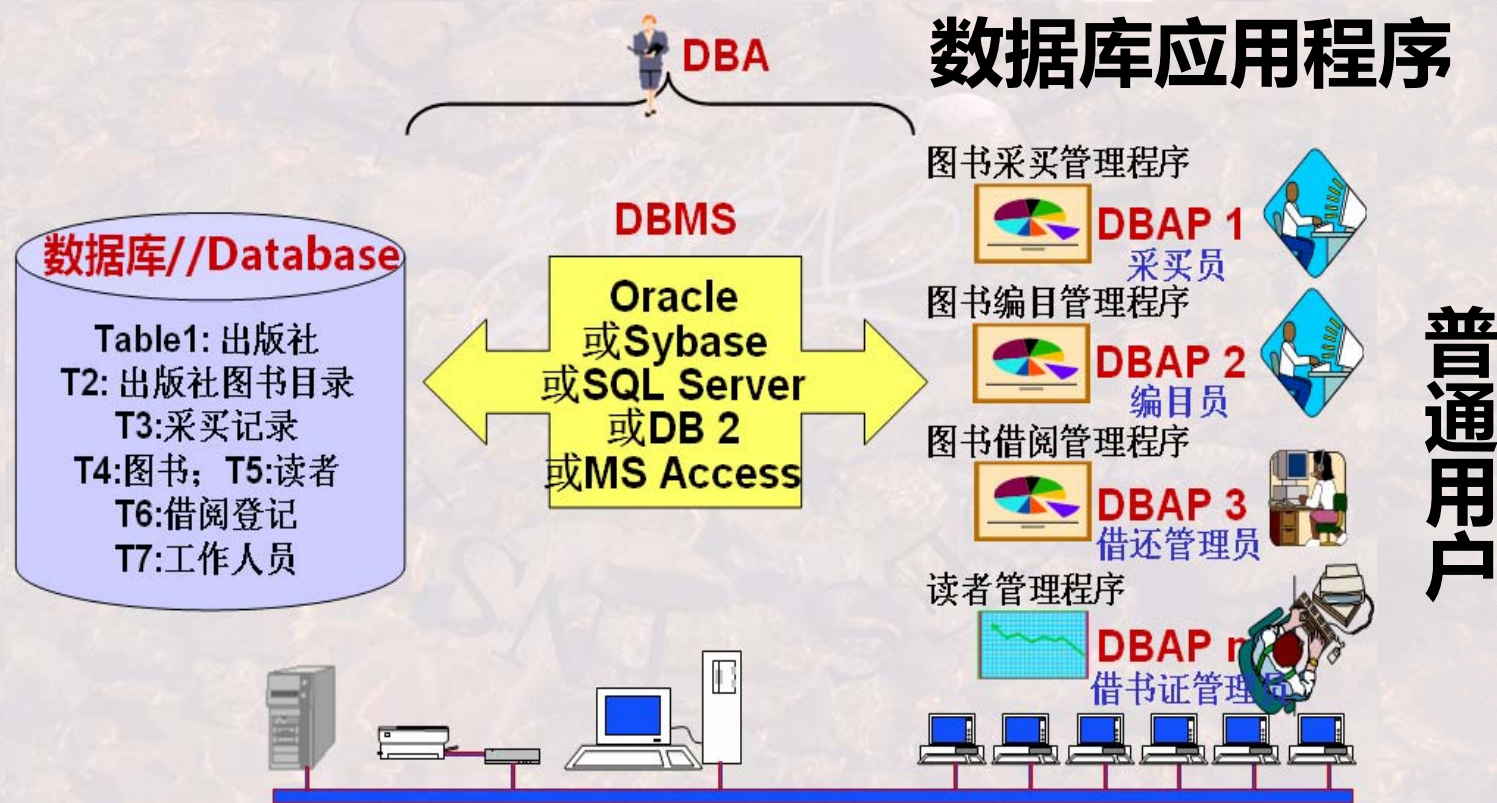
有什么局限性呢...

嵌入式SQL语言概述

(1) 交互式SQL语言的局限

从使用者角度

➤ 专业人员可熟练写出SQL语句，但大部分的普通用户...



从SQL本身角度

- 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

- 再如：循环地进行检索操作

Do While some-condition

SQL-Query

End Do

一条SQL
语句不能
包打天下

多条SQL语
句怎样联合完
成检索呢

SQL+
高级语言

➤再如：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

一条SQL
语句不能
包打天下

多条SQL语
句怎样联合完
成检索呢

SQL+
高级语言

嵌入式SQL语言概述

(2) 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，PowerBuilder等，又称**宿主语言(Host Language)**
- ❑ 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

示例：交互式SQL语言

```
select Sname, Sage from Student where Sname='张三';
```

示例：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

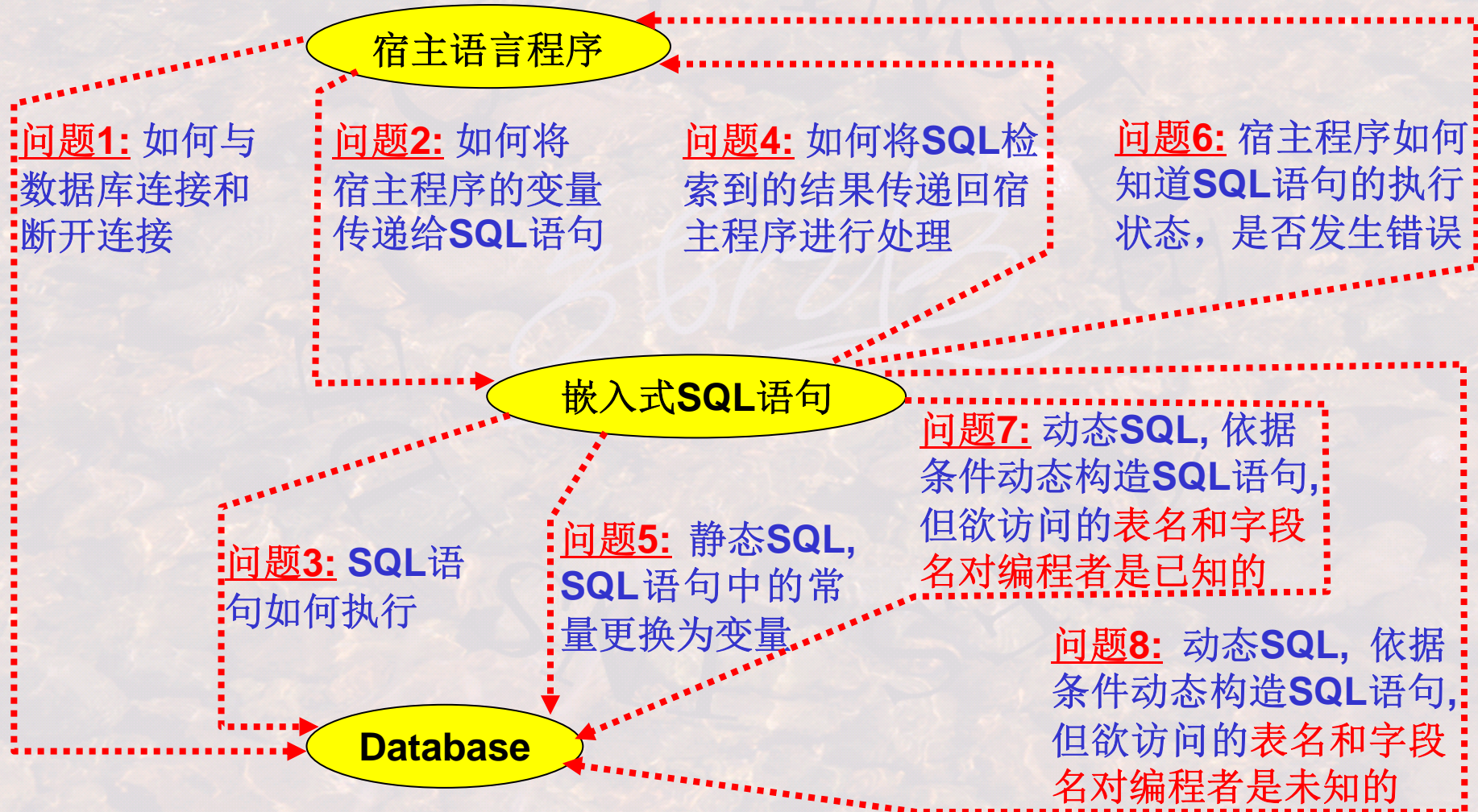
---- 增加一 **into**子句: 该子句用于指出接收SQL语句检索结果的程序变量

---- 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。

交互式SQL
与嵌入式
SQL的差别?

高级语言(语句)→嵌入式SQL语句→DBMS←→DB



学习目标

- 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
- 理解事务的概念和应用
- 能够结合C/PowerBuilder/Java等(同学自学), 熟练地编写数据库应用程序

变量声明与数据库连接

(高级语言如何与数据库连接)

战德臣

哈尔滨工业大学 教授·博士生导师
黑龙江省教学名师
教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent
Computing for Enterprises & Services,
Harbin Institute of Technology

高级语言(语句)→嵌入式SQL语句→DBMS←→DB



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= :specName;
```

- 这些变量需要特殊的声明：

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

- 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。
- 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。

嵌入式SQL的
变量为什么要
加冒号？

嵌入式SQL的
变量为什么要
声明？

变量声明与数据库连接

(2)变量的声明与使用



- 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;
```

```
//用户可在此处基于键盘输入给specName赋值
```

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

- 比较相应的交互式SQL语句：

```
select Sname, Sage 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。

体验到嵌入式
SQL的可变化
性了吗？

程序与数据库的连接和断开

- 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**;

➤ 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中断开连接：

exec sql commit release;

或

exec sql rollback release;

➤ DB2 UDB中断开连接：

exec sql connect reset;

exec sql disconnect current;

SQL执行的提交与撤销

- SQL语句在执行过程中，必须有**提交**和**撤销**语句才能确认其操作结果
- SQL执行的**提交**：
`exec sql commit work;`
- SQL执行的**撤销**：
`exec sql rollback work;`
- 为此，很多DBMS都设计了捆绑提交/撤销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销先前的工作，例如Oracle中：
`exec sql commit release;`
或
`exec sql rollback release;`

为什么需要
提交和撤销?

事务：(从应用程序员角度)是一个存取或改变数据库内容的程序的一次执行，或者说一条或多条SQL语句的一次执行被看作一个事务

➤事务一般是由应用程序员提出，因此有开始和结束, 结束前需要提交或撤消。

Begin Transaction

```
exec sql ...
```

```
...
```

```
exec sql ...
```

```
exec sql commit work | exec sql rollback work
```

End Transaction

➤在嵌入式SQL程序中，任何一条数据库操纵语句(如exec sql select等)都会引发一个新事务的开始，只要该程序当前没有正在处理的事务。而事务的结束是需要应用程序员通过commit或rollback确认的。因此Begin Transaction 和End Transaction两行语句是不需要的。

事务：(从微观角度，或者从DBMS角度)是数据库管理系统提供的控制数据操作的一种手段，通过这一手段，应用程序员将一系列的数据库操作组合在一起作为一个整体进行操作和控制，以便数据库管理系统能够提供一致性状态转换的保证。

示例：“银行转帐”事务T：从帐户A过户5000RMB到帐户B上

T: **read(A);**
 A := A - 5000;
 write(A);
 read(B);
 B := B + 5000;
 write(B);

注：**read(X)**是从数据库传送数据项X到事务的工作区中；**write(X)**是从事务的工作区中将数据项X写回数据库。

事务的特性: ACID

- **原子性Atomicity** : DBMS能够保证事务的一组更新操作是原子不可分的, 即对DB而言, 要么全做, 要么全不做
 - **一致性Consistency**: DBMS保证事务的操作状态是正确的, 符合一致性的操作规则, 它是进一步由隔离性来保证的
 - **隔离性Isolation**: DBMS保证并发执行的多个事务之间互相不受影响。例如两个事务T1和T2, 即使并发执行, 也相当于或者先执行了T1,再执行T2;或者先执行了T2, 再执行T1。
 - **持久性Durability**: DBMS保证已提交事务的影响是持久的, 被撤销事务的影响是可恢复的。
- 换句话说: 具有**ACID**特性的若干数据库基本操作的组合体被称为事务。

事务处理是
DBMS的核心
技术哟

变量声明与数据库连接

(6)示例



示例

```
#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;
```

-----SQLCA: SQL Communication Area / 后面介绍

-----The Declare Section

-----SQL错误捕获语句 / 后面介绍

-----SQL Connect

变量声明与数据库连接

(6)示例



```
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; -----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; -----SQL Commit Work and Disconnect  
    return 0;  
report_error:  
    print_dberror();  
    exec sql rollback release; -----SQL Rollback Work and Disconnect  
    return 1;  
}
```


数据集与游标

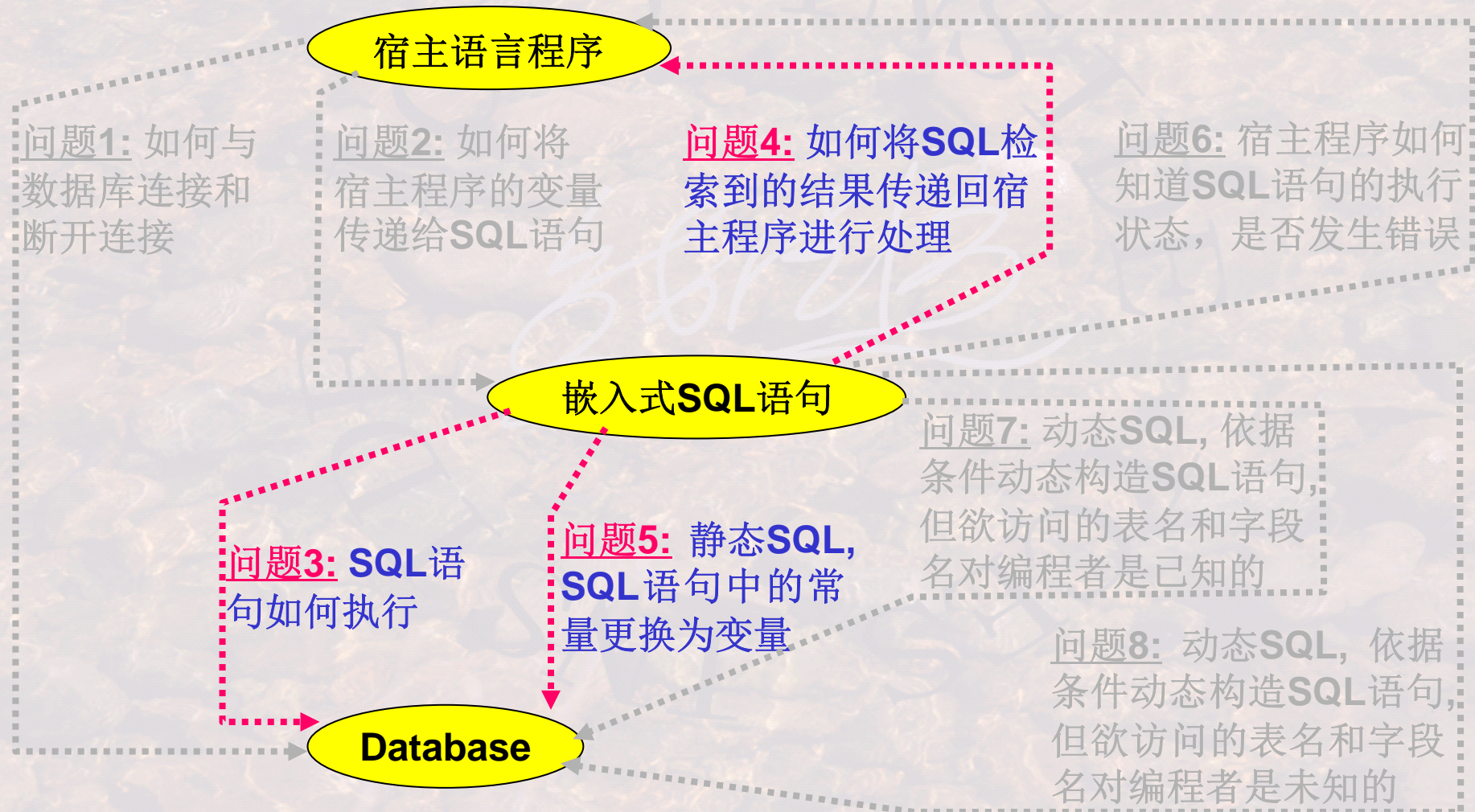
(高级语言如何读取单行与多行数据)

战德臣

哈尔滨工业大学 教授·博士生导师
黑龙江省教学名师
教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent
Computing for Enterprises & Services,
Harbin Institute of Technology

高级语言(语句)→嵌入式SQL语句→DBMS↔DB



单行结果处理与多行结果处理的差异(Into子句与游标(Cursor))

■检索单行结果，可将结果直接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;
```

示例

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


■检索多行结果，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		
Sno	Sname	Sclass
2003510101	张三	035101
2003510102	李四	035101
2003510103	王五	035101
2003510104	李六	035101
2003510105	张四	035101
2003510106	张五	035101
2003510107	张小三	035101
2003510108	张小四	035101
2003510109	李小三	035101
2003510110	李小四	035101
2003520201	周三	035202
2003520202	赵四	035202
2003520203	赵五	035202
2003520204	赵六	035202
2003520205	钱四	035202
2003520206	强五	035202
2003520207	梁小三	035202
2003520208	梁小四	035202
2003520209	王小三	035202
2003520210	王小四	035202

DataBase

Select
(记录集)



Result(Record set) by select statement		
Sno	Sname	Sclass
2003510101	张三	035101
2003510102	李四	035101
2003510103	王五	035101
2003510104	李六	035101
2003510105	张四	035101
2003510106	张五	035101
2003510107	张小三	035101
2003510108	张小四	035101
2003510109	李小三	035101
2003510110	李小四	035101

Cursor_name

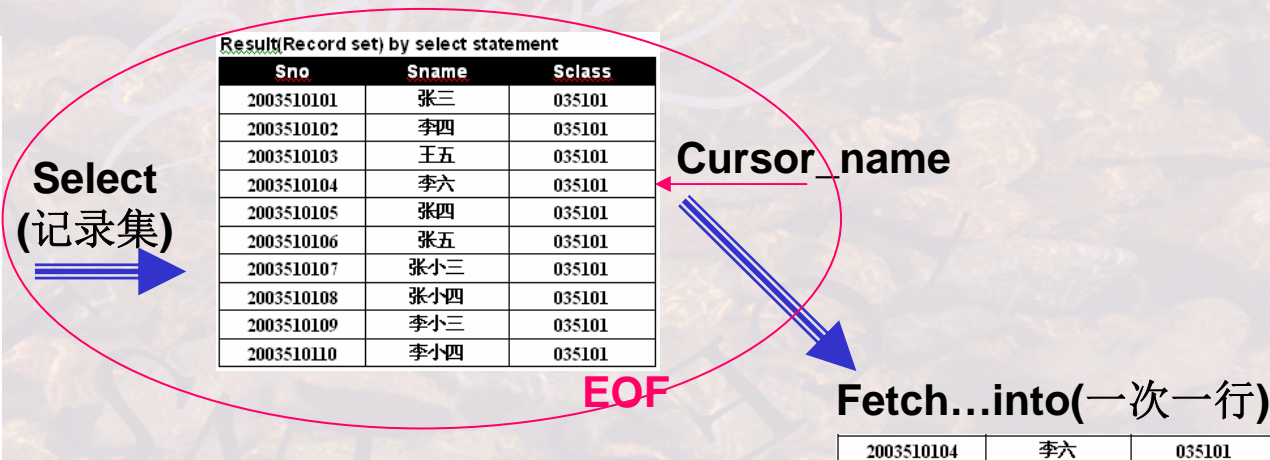


■检索多行结果，则需使用游标(Cursor)

- 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, 用来标记后面已没有记录了

Student		
Sno	Sname	Sclass
2003510101	张三	035101
2003510102	李四	035101
2003510103	王五	035101
2003510104	李六	035101
2003510105	张四	035101
2003510106	张五	035101
2003510107	张小三	035101
2003510108	张小四	035101
2003510109	李小三	035101
2003510110	李小四	035101
2003520201	周三	035202
2003520202	赵四	035202
2003520203	赵五	035202
2003520204	赵六	035202
2003520205	钱四	035202
2003520206	强五	035202
2003520207	梁小三	035202
2003520208	梁小四	035202
2003520209	王小三	035202
2003520210	王小四	035202

DataBase



游标(Cursor)的使用

➤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

数据集与游标

(4)示例

示例

```
#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;
```

Orders				
cid	aid	product	...	dollars
C1	A1	P1		100.00
C1	A1	P2		200.00
C1	A1	P3		400.00
C1	A2	P2		100.00
C1	A2	P3		200.00
C1	A2	P1		300.00
C1	A3	P3		100.00
C1	A3	P1		200.00
C1	A3	P2		300.00
C1	A3	P1		400.00
C2	A1	P1		100.00
C2	A1	P2		200.00
C2	A1	P3		400.00
C2	A2	P2		100.00
C2	A2	P3		200.00
C2	A2	P1		300.00
C2	A2	P2		400.00
C2	A3	P3		100.00
C2	A3	P2		300.00
C2	A3	P1		400.00

agent_dollars with cid=' c1'

Aid	dollars
A1	700.00
A2	600.00
A3	1000.00

agent_dollars with cid=' c2'

aid	dollars
A1	700.00
A2	1000.00
A3	800.00


```

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;
}

```

Orders				
cid	aid	product	...	dollars
C1	A1	P1		100.00
C1	A1	P2		200.00
C1	A1	P3		400.00
C1	A2	P2		100.00
C1	A2	P3		200.00
C1	A2	P1		300.00
C1	A3	P3		100.00
C1	A3	P1		200.00
C1	A3	P2		300.00
C1	A3	P1		400.00
C2	A1	P1		100.00
C2	A1	P2		200.00
C2	A1	P3		400.00
C2	A2	P2		100.00
C2	A2	P3		200.00
C2	A2	P1		300.00
C2	A2	P2		400.00
C2	A3	P3		100.00
C2	A3	P2		300.00
C2	A3	P1		400.00

agent_dollars with cid=' c1'

Aid	dollars
A1	700.00
A2	600.00
A3	1000.00

agent_dollars with cid=' c2'

aid	dollars
A1	700.00
A2	1000.00
A3	800.00

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...]]]]];
```

示例

```
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;
```


数据集与游标

(5)游标的使用方法



Cursor的数据读取 : Fetch

```
EXEC SQL FETCH cursor_name  
      INTO host-variable , [host-variable, ...];
```

示例

```
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;
```


可滚动游标及数据库的增删改

战德臣

哈尔滨工业大学 教授.博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on **I**ntelligent
Computing for **E**nterprises & **S**ervices,
Harbin **I**nstitute of **T**echnology

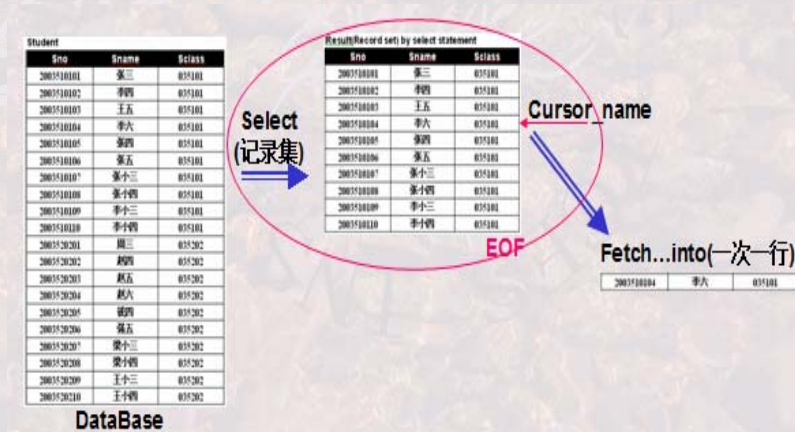
可滚动游标及数据库的增删改

(1)可滚动游标的概念

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

```
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;
```



- ODBC(Open DataBase Connectivity)是一种跨DBMS的DB操作平台，它在应用程序与实际的DBMS之间提供了一种通用接口
- 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

可滚动游标及数据库的增删改

(2)可滚动游标的定义和使用



➤可滚动游标是可使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 ]...];
```

```
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移动到最后一行；ABSOLUTE 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

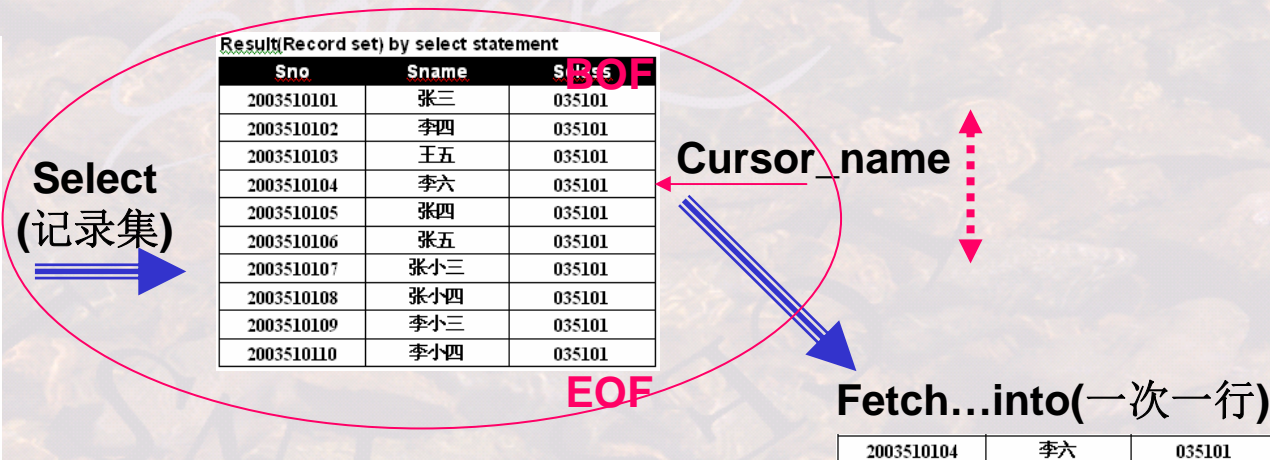
可滚动游标及数据库的增删改

(2)可滚动游标的定义和使用

- 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		
Sno	Sname	Sclass
2003510101	张三	035101
2003510102	李四	035101
2003510103	王五	035101
2003510104	李六	035101
2003510105	张四	035101
2003510106	张五	035101
2003510107	张小三	035101
2003510108	张小四	035101
2003510109	李小三	035101
2003510110	李小四	035101
2003520201	周三	035202
2003520202	赵四	035202
2003520203	赵五	035202
2003520204	赵六	035202
2003520205	钱四	035202
2003520206	强五	035202
2003520207	梁小三	035202
2003520208	梁小四	035202
2003520209	王小三	035202
2003520210	王小四	035202

DataBase



数据库记录的删除

➤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;
```

示例：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);
```

示例：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 ; }
```


数据库记录的更新

➤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;
```

示例：查找更新

```
exec sql update student s set sclass = '035102'
      where s.sclass = '034101'
```

示例：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 ; }
```


数据库记录的插入

➤ 只有一种类型的插入语句

```
EXEC SQL INSERT INTO tablename [ (columnname [,  
columnname, ...] )]  
[ VALUES (expr [ , expr , ...] ) | subquery ] ;
```

示例：插入语句

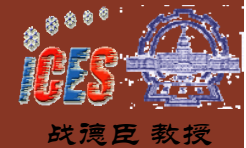
```
exec sql insert into student ( sno, sname, sclass)  
values ('03510128', '张三', '035101') ;
```

示例：插入语句

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


可滚动游标及数据库的增删改

(4)示例



示例：宿主语言与SQL结合的过程性控制 求数据库中某一行位于中值的那一行

```
#include <stdio.h>
#include "prompt.h"
exec sql include sqlca;
char custprompt[ ]="Please enter a customer ID: "

int main()
{
    exec sql begin declare section;
        char cid[5],user_name[20], user_pwd[10];
        double dollars; int ocount;
    exec sql end declare section;
    exec sql declare dollars_cursor cursor for
        select dollars from orders where cid = :cid and dollars is not null
        order by dollars;

    int i;
```

Orders				
cid	aid	product	dollars
C1	A1	P1		100.00
C1	A2	P2		100.00
C1	A3	P3		100.00
C1	A3	P1		200.00
C1	A1	P2		200.00
C1	A2	P3		200.00
C1	A3	P2		300.00
C1	A2	P1		300.00
C1	A1	P3		400.00
C1	A3	P1		400.00
C2	A1	P1		100.00
C2	A3	P3		100.00
C2	A2	P2		100.00
C2	A2	P3		200.00
C2	A1	P2		200.00
C2	A2	P1		300.00
C2	A3	P2		300.00
C2	A3	P1		400.00
C2	A2	P2		400.00
C2	A1	P3		400.00

可滚动游标及数据库的增删改

(4)示例

```
exec sql whenever sqlerror goto report_error;
strcpy(user_name, "poneilsql");
strcpy(user_pwd, "xxxx");
exec sql connect :user_name identified by :user_pwd;
While ( prompt(custprompt, 1, cid ,4)>=0 {
    exec sql select count(dollars) into :ocount from orders where cid = :cid ;
    if (ocount ==0 ){
        printf("No record reviewed for cid value %s\n", cid);
        continue; }
    exec sql open dollars_cursor;
    for (i =0; i< (ocount+1)/2; i++)
        exec sql fetch dollars_cursor into :dollars;
    exec sql close dollars_cursor;
    exec sql commit work;
    printf("Median dollar amount =%f\n", dollars); }
...
}
```

Orders				
cid	aid	product	dollars
C1	A1	P1		100.00
C1	A2	P2		100.00
C1	A3	P3		100.00
C1	A3	P1		200.00
C1	A1	P2	↔	200.00
C1	A2	P3		200.00
C1	A3	P2		300.00
C1	A2	P1		300.00
C1	A1	P3		400.00
C1	A3	P1		400.00
C2	A1	P1		100.00
C2	A3	P3		100.00
C2	A2	P2		100.00
C2	A2	P3		200.00
C2	A1	P2		200.00
C2	A2	P1		300.00
C2	A3	P2		300.00
C2	A3	P1		400.00
C2	A2	P2		400.00
C2	A1	P3		400.00

状态捕获及错误处理机制

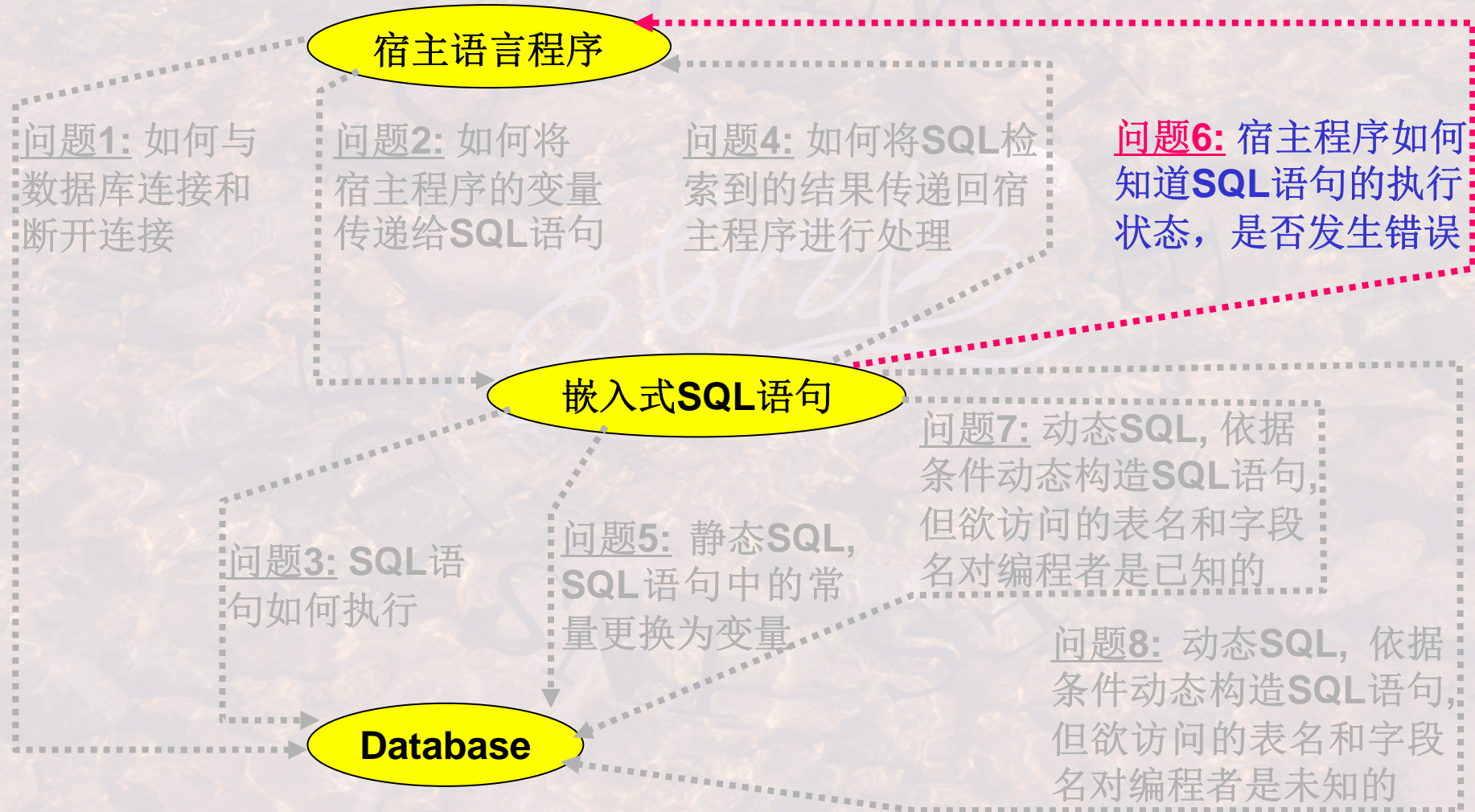
(高级语言如何识别数据库连接及数据读写的正确性)

战德臣

哈尔滨工业大学 教授·博士生导师
黑龙江省教学名师
教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent
Computing for Enterprises & Services,
Harbin Institute of Technology

高级语言(语句)→嵌入式SQL语句→DBMS←→DB



状态捕获及其处理

- 状态，是嵌入式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;

SQL通信区: SQLCA

- **SQLCA**是一个已被声明过的具**C**语言的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

```
struct sqlca
{
    char sqlcaid[8];
    long sqlabc;
    long sqlcode;
    struct
    {
        int sqlerrml;
        char sqlerrmc[SQLERRMC_LEN];
    } sqlerrm;
    char sqlerrp[8];
    long sqlerrd[6];
    char sqlext[8];
};
```

可查阅相关手册解读相关数据项的含义

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

➤ 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**Exec SQL**语句之后的语句继续进行

状态捕获及错误处理机制

(2)基本机制



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;
    ....
}
```

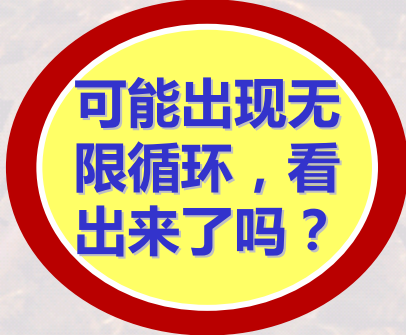
这才是关键点，要特别注意

- 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)

状态捕获语句Whenever的使用容易引发无限循环

```
int main()
{
    exec sql whenever sqlerror goto handle_error;
    exec sql create table customers(cid char(4) not null,
                                   cname varchar(13), ... .. );

    ... ..
handle_error:
    exec sql drop customers;
    exec sql disconnect;
    fprintf(stderr,"could not create customers table\n");
    return -1;
}
```



可能出现无限循环，看出来了吗？

状态捕获语句Whenever的使用容易引发无限循环

```
int main()
{
    exec sql whenever sqlerror goto handle_error;
    exec sql create table customers(cid char(4) not null,
                                   cname varchar(13), ... .. );
    ... ..
handle_error:
    exec sql whenever sqlerror continue;
    /*控制是否无限循环：无，则可能；有，则不会
    exec sql drop customers;
    exec sql disconnect;
    fprintf(stderr,"could not create customers table\n");
    return -1;
}
```

**一定要加这
条语句哟！**

典型DBMS系统记录状态信息的三种方法

状态记录

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

✓ **sqlcode == 0**, successful call;

✓ **sqlcode < 0**, error, e.g., from connect, database does not exist, -16;

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

❑ **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

❑ **sqlstate**: 有些DBMS提供的记录状态信息的变量是**sqlstate**或**sqlca.sqlstate**

➤ 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, 则需要知道以上信息, 但也需知道正确的操作方法(显式状态处理)

程序自身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>  
... ..
```

注意语句位置
不同可能出现
问题哟

➤ 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.

➤ 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>  
... ..
```

为什么这是
正确的呢？

➤ 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。

回顾本讲学了什么？

战德臣

哈尔滨工业大学 教授·博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on **I**ntelligent
Computing for **E**nterprises & **S**ervices,
Harbin **I**nstitute of **T**echnology

回顾本讲学习了什么？

