数据库系统

第四章 数据库编程（课本第八章）

回顾

- 标准SQL
  - 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
  - 高度非过程化的优点也同时造成了它的一个弱点：缺少流程控制能力，难以实现应用业务中的逻辑控制

回顾

- 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:控制检索操作执行的顺序

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

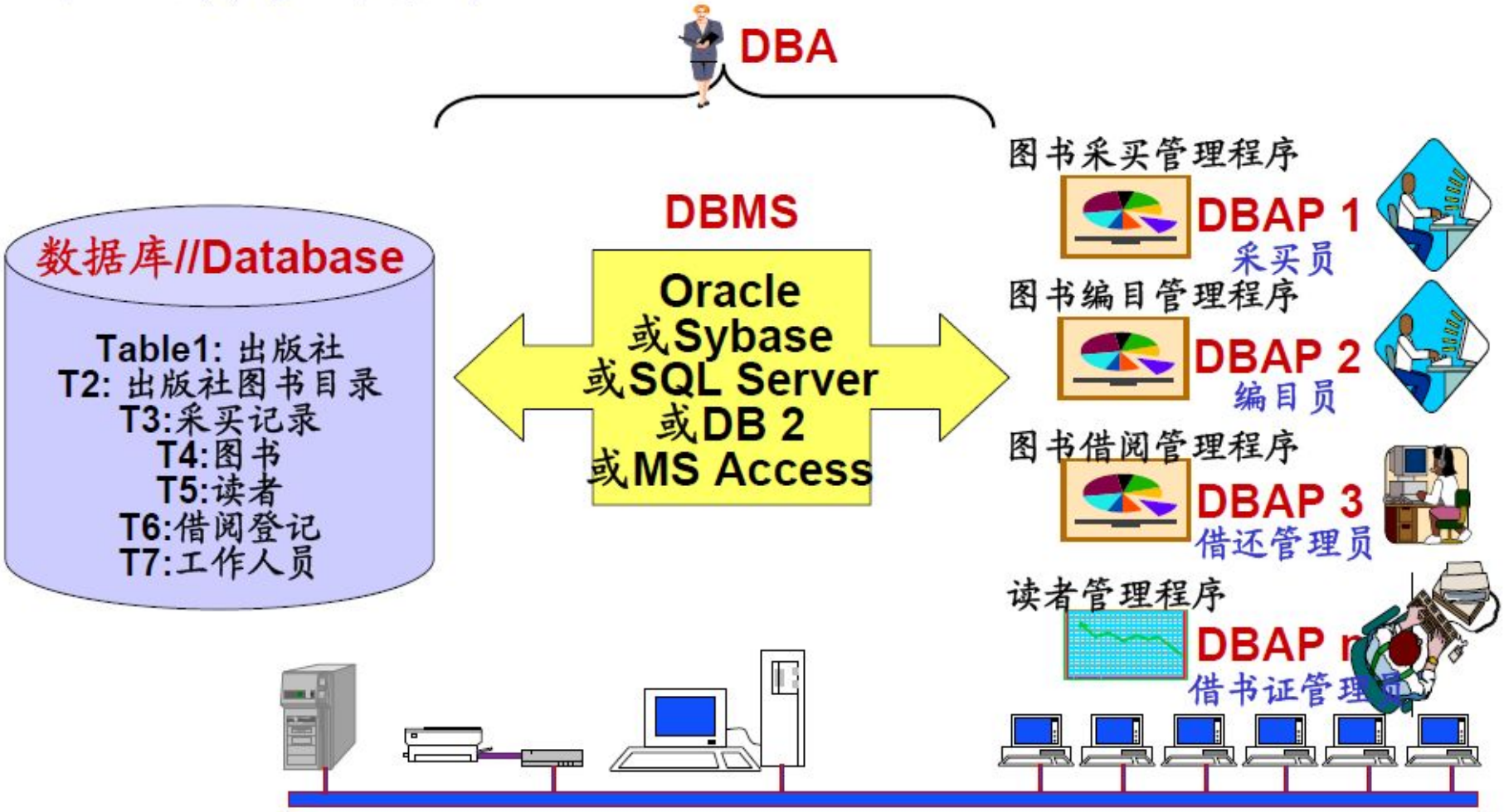
回顾

- 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
```

例如：图书管理数据库


DBA
DBMS
Oracle
或Sybase
或SQL Server
或DB 2
或MS Access
数据库//Database
Table1: 出版社
T2: 出版社图书目录
T3:采买记录
T4:图书
T5:读者
T6:借阅登记
T7:工作人员
图书采买管理程序
DBAP 1
采买员
图书编目管理程序
DBAP 2
编目员
图书借阅管理程序
DBAP 3
借还管理员
读者管理程序
DBAP n
借书证管理员

回顾

- *SQL编程技术*能够有效克服SQL语言实现复杂应用方面的不足，提供应用系统和RDBMS之间的互操作性。

第4章 数据库编程

- 4.1 嵌入式SQL
- 4.2 ODBC编程
- 4.3 存储过程

第4章 数据库编程

- *4.1 嵌入式SQL*
- 4.2 ODBC编程
- 4.3 存储过程

4.1嵌入式SQL

- 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，称为*宿主语言(Host Language)*，简称*主语言*
- 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

- **交互式SQL语言**

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

4.1嵌入式SQL

- 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
    - 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’

4.1嵌入式SQL

- 主要内容
  - 嵌入式SQL的处理过程
  - 嵌入式SQL语句与主语言之间的通信
  - 不使用游标的SQL语句
  - 使用游标的SQL语句
  - 动态SQL

4.1嵌入式SQL-处理过程

- **预编译方法**

ESQL基本处理过程

4.1嵌入式SQL-处理过程

- 为了区分SQL语句与主语言语句，所有SQL语句必须加前缀EXEC SQL，以(;)结束:

EXEC SQL <SQL语句>;

4.1嵌入式SQL

DBMS	内嵌SQL支持的宿主语言
DB2	Assembler, Basic, Cobol, Fortran, Java, PL/I
Informix	C, Cobol
SQL Server	C
Oracle	C, Cobol, Fortran, Pascal, PL/I
Sybase	C, Cobol

4.1嵌入式SQL-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
    - 负责操纵数据库
    - 描述性的面向集合的语句
  - 高级语言语句
    - 负责控制程序流程
    - 过程性的面向记录的语句

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- 数据库工作单元与源程序工作单元之间的通信
  - SQL通信区
    - 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
  - 主变量
    - 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

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- 数据库工作单元与源程序工作单元之间的通信
  - 游标
    - 解决集合性操作语言与过程性操作语言的不匹配

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- SQL通信区
  - SQLCA： SQL Communication Area
    - *SQLCA是一个数据结构*

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

```
struct sqlca
{
    unsigned char  sqlcaid[8] ;     /* the string "SQLCA"   */
    long           sqlcabc ;        /* length of SQLCA, in bytes */
    long           sqlcode ;        /* SQL status code */
    short          sqlerrml ;       /* length of sqlerrmc array data */
    unsigned char  sqlerrmc[70] ; /* names of objects causing
                                     error   */
    unsigned char  sqlerrp[8] ;  /* diagnostic information   */
    long           sqlerrd[6] ;  /* various counts and error code */
    unsigned char  sqlwar[8] ;   /* warning flag arry   */
    unsigned char  sqlext[8] ;   /* extension to sqlwarn array   */
}
```

内嵌SQL执行的状态代码

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- **SQL通信区**
  - **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**

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- SQL通信区
  - SQLCA使用方法
    - 定义SQLCA
      - 用*EXEC SQL INCLUDE SQLCA*定义

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- **SQL通信区**
  - **SQLCA使用方法**
    - **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

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- ✓sqlcode=0 ： SQL语句成功完成，且没有警告.
- ✓sqlcode>0 ： SQL语句成功完成，但带有警告.
- ✓sqlcode<0 ： SQL语句执行失败，执行过程中产生严重错误.

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- 主变量
  - 在SQL语句中使用的主语言程序变量简称为主变量（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

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- 主变量
  - 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

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- DBMS的数据类型和主语言数据类型转换

SQL数据类型(SQL2标准)	C语言
smallint	short
integer	int/long
real/money	float
double	double
char(n)	char x[n+1]
varchar(n)	char x[n+1]

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- **主变量**
  - **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**
      - 整型变量
      - 用来“指示”所指主变量的值或条件

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- **主变量**
  - **指示变量**
    - **指示变量的用途**
      - 输入主变量可以利用指示变量赋空值
      - 输出主变量可以利用指示变量检测出是否空值，值是否被截断

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- **主变量**
  - **指示变量**

✓指示器x_ind=0, 宿主变量x包含一个有效值.

✓指示器x_ind<0, 宿主变量x没有实际值, 即为Null值.

✓指示器x_ind>0, 宿主变量x包含一个有效值, 但该值经过四舍五入或截断等处理.

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- 主变量
  - 指示变量
    - 指示变量不能用在查询条件（如where子句）中，在查询条件中应使用显示的IS NULL测试。

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- **主变量**
  - **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

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- 主变量
  - 在SQL语句中使用主变量和指示变量的方法

    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

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- 主变量
  - 在SQL语句中使用主变量和指示变量的方法

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

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- 主变量
  - 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

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- 游标（cursor）
  - 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

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- 游标（cursor）
  - 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

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- 游标（cursor）
  - 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

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- 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*];

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- 建立和关闭数据库连接
  - 关闭数据库连接

EXEC SQL DISCONNECT [*connection*];

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

- [例]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*/
```

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

```
EXEC SQL INCLUDE sqlca;                    /*定义SQL通信区*/
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  LENOVO-96A870.TEST  USER sa.sa;
     /*连接数据库TEST*/
```

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

```
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*/
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*/
```

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

```
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');
```

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

```
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*/
}
```

4.1嵌入式SQL-嵌入式SQL语句与主语言之间的通信

```
    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*/

     return 0;

}
```

4.1嵌入式SQL-不使用游标的SQL语句

- **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形式的增删改语句*

4.1嵌入式SQL-不使用游标的SQL语句

- **查询结果为单记录的SELECT语句**
  - **[例] 根据学生号码查询学生信息。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；

4.1嵌入式SQL-不使用游标的SQL语句

- 查询结果为单记录的SELECT语句

(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。

(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

4.1嵌入式SQL-不使用游标的SQL语句

- 查询结果为单记录的SELECT语句
  - [例] 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 ;
```

4.1嵌入式SQL-不使用游标的SQL语句

- 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

4.1嵌入式SQL-不使用游标的SQL语句

- 非CURRENT形式的增删改语句
  - **[例] 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*/
                and cno='1'
```

4.1嵌入式SQL-不使用游标的SQL语句

- 非CURRENT形式的增删改语句
  - [例] 将CS系全体学生年龄置NULL值

```
Sageid=-1;
EXEC SQL UPDATE Student
        SET Sage=:Raise :Sageid
        WHERE Dno= 'CS';
```

等价于:

```
EXEC SQL UPDATE Student
        SET Sage=NULL
        WHERE Dno= 'CS';
```

4.1嵌入式SQL-不使用游标的SQL语句

- **非CURRENT形式的增删改语句**
  - **[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)；
```

4.1嵌入式SQL-不使用游标的SQL语句

- 非CURRENT形式的增删改语句
  - [例7] 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；          /*用作指示变量，赋为负值*/
EXEC SQL INSERT
    INTO SC(Sno，Cno，Grade)
    VALUES(:stdno，:couno，:gr :gradeid)；
```

4.1嵌入式SQL-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

4.1嵌入式SQL-使用游标的SQL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

4.1嵌入式SQL-使用游标的SQL语句

- 说明游标
  - 语句格式

    EXEC SQL *DECLARE* <游标名> CURSOR
    FOR <SELECT语句>;
  - 功能
    - 是一条说明性语句，这时DBMS并不执行SELECT指定的查询操作。

4.1嵌入式SQL-使用游标的SQL语句

- 打开游标
  - 语句格式

    EXEC SQL *OPEN* <游标名>;
  - 功能
    - 打开游标实际上是执行相应的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前

4.1嵌入式SQL-使用游标的SQL语句

- 推进游标
  - 语句格式

    EXEC SQL *FETCH* <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来送至主变量供主语言进一步处理

4.1嵌入式SQL-使用游标的SQL语句

- 关闭游标
  - 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

4.1嵌入式SQL-使用游标的SQL语句

- CURRENT形式的UPDATE语句和DELETE语句
  - 面向集合的操作
  - 一次修改或删除所有满足条件的记录

4.1嵌入式SQL-使用游标的SQL语句

- CURRENT形式的UPDATE语句和DELETE语句
  - 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语句和DELETE语句修改或删除之
    - UPDATE语句和DELETE语句中的子句：

      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

4.1嵌入式SQL-使用游标的SQL语句

- 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

4.1嵌入式SQL-动态SQL

- 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

4.1嵌入式SQL-动态SQL

- 使用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

4.1嵌入式SQL-动态SQL

[例] 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 */
```

4.1嵌入式SQL-动态SQL

- 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

4.1嵌入式SQL-动态SQL

- **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>];

4.1嵌入式SQL-动态SQL

[例]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 */
```

第4章 数据库编程

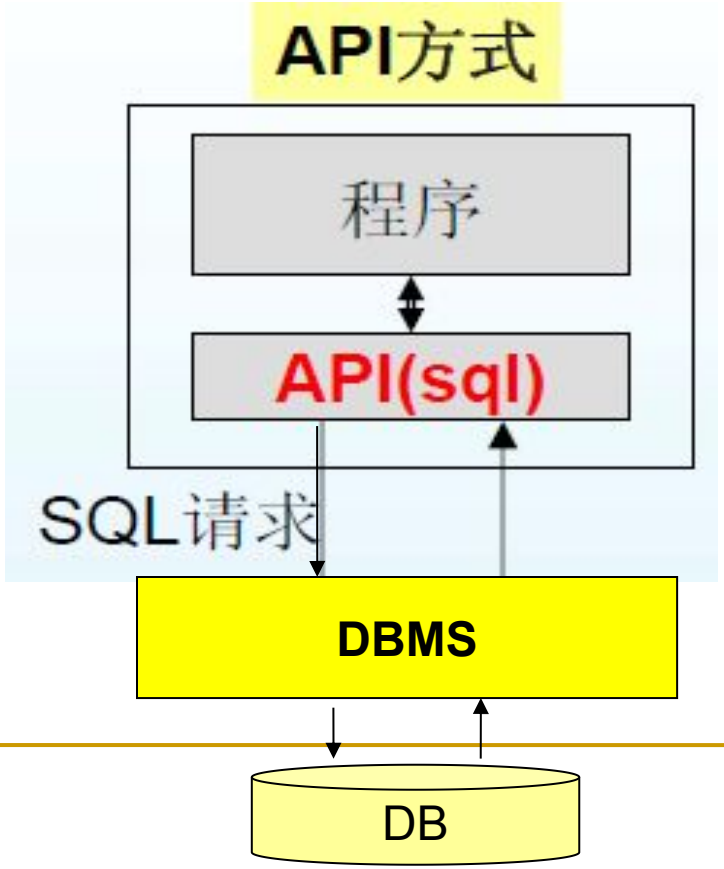
- 4.1 嵌入式SQL
- *4.2 ODBC编程*
- 4.3 存储过程

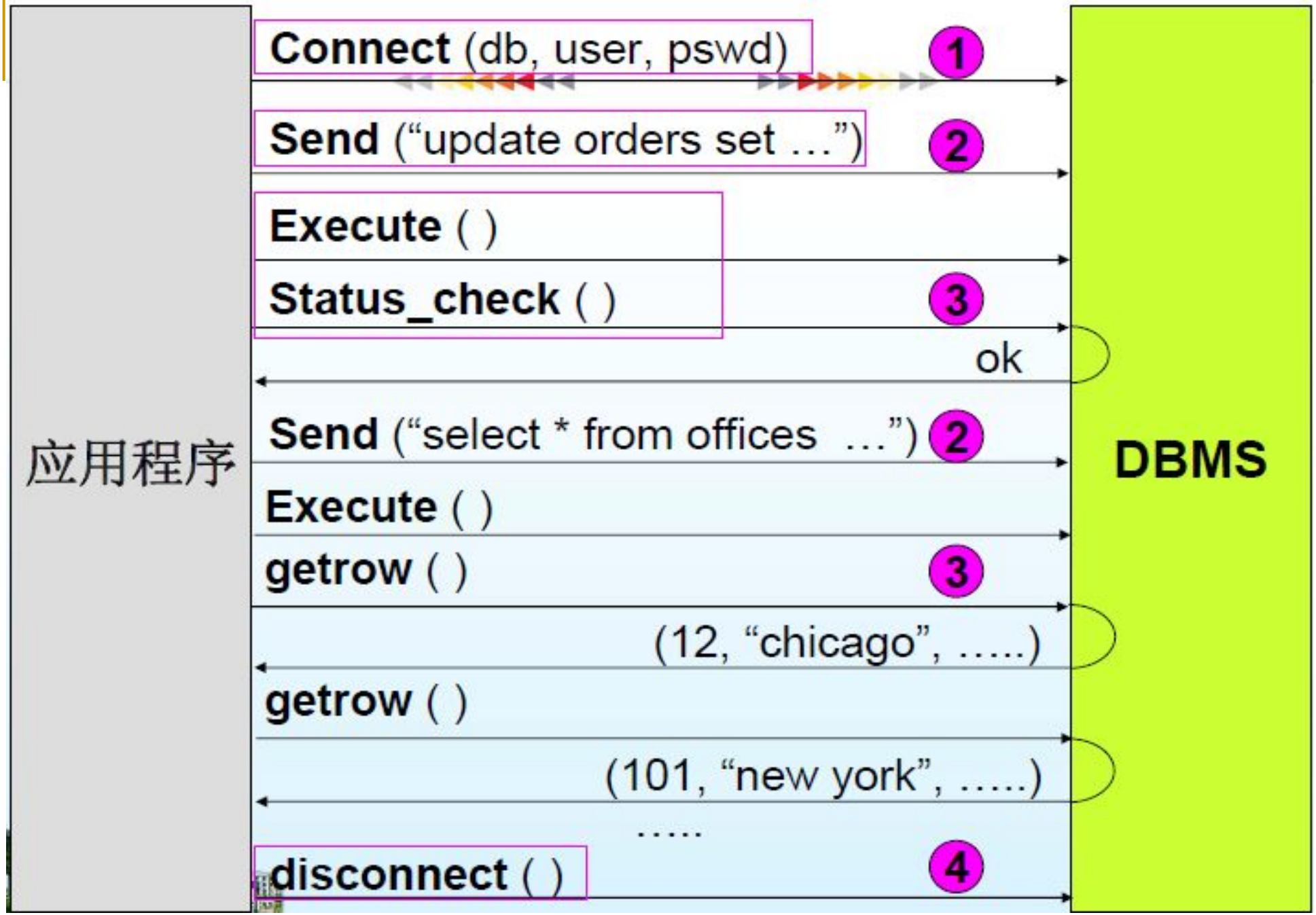
4.2ODBC编程

- 由于不同的数据库管理系统的存在，在某个RDBMS下编写的应用程序就不能在另一个RDBMS下运行
- 许多应用程序需要共享多个部门的数据资源，访问不同的RDBMS
- 数据库开放互连？

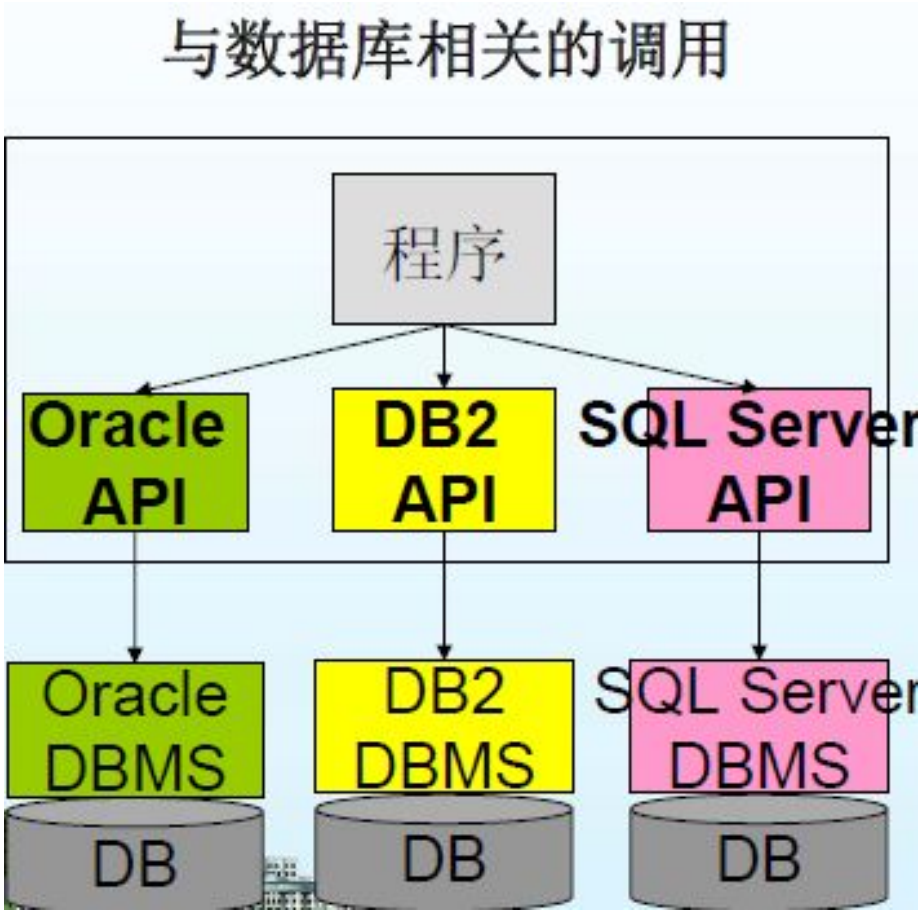
4.2ODBC编程

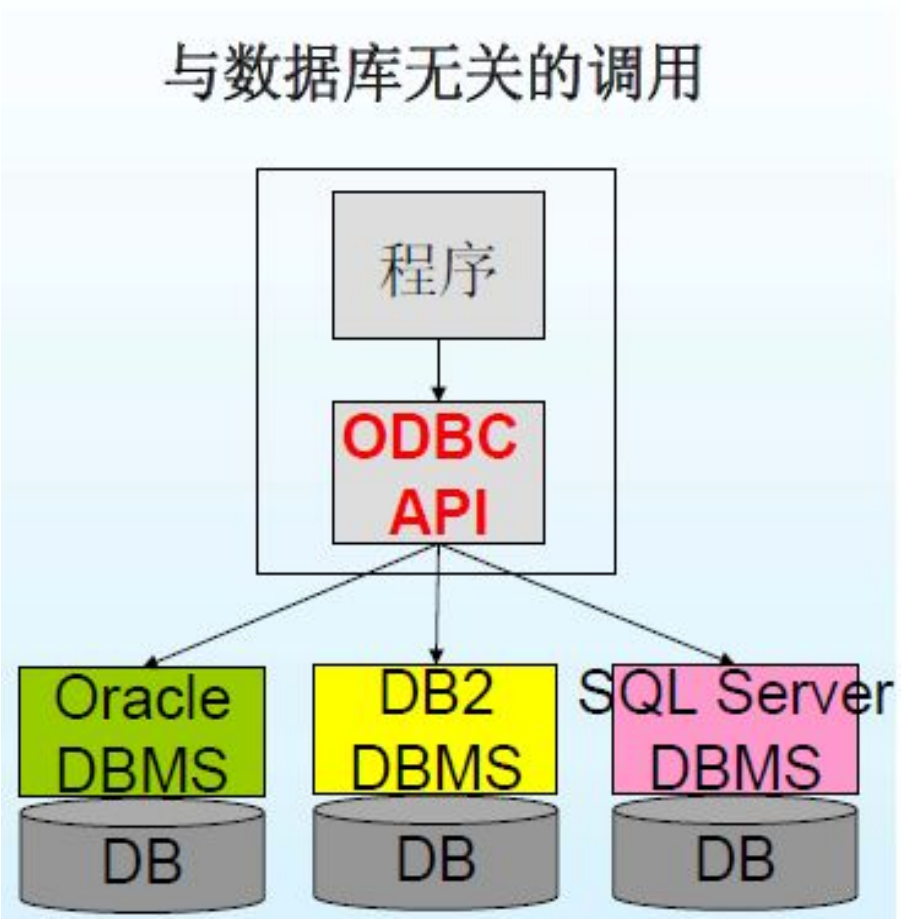
- API的使用：API是一种主程序和DBMS之间通信的协议，由DBMS提供并被主程序支持的调用（函数或类）




应用程序
Connect (db, user, pswd)
1
Send ("update orders set …")
2
Execute ( )
Status_check ( )
3
ok
Send ("select * from offices …")
2
Execute ( )
getrow ( )
3
(12, "chicago", …..)
getrow ( )
(101, "new york", …..)
…..
disconnect ( )
4
DBMS

4.2ODBC编程


与数据库相关的调用
程序
Oracle API
DB2 API
SQL Server API
Oracle DBMS
DB2 DBMS
SQL Server DBMS
DB
DB
DB


与数据库无关的调用
程序
ODBC API
Oracle DBMS
DB2 DBMS
SQL Server DBMS
DB
DB
DB

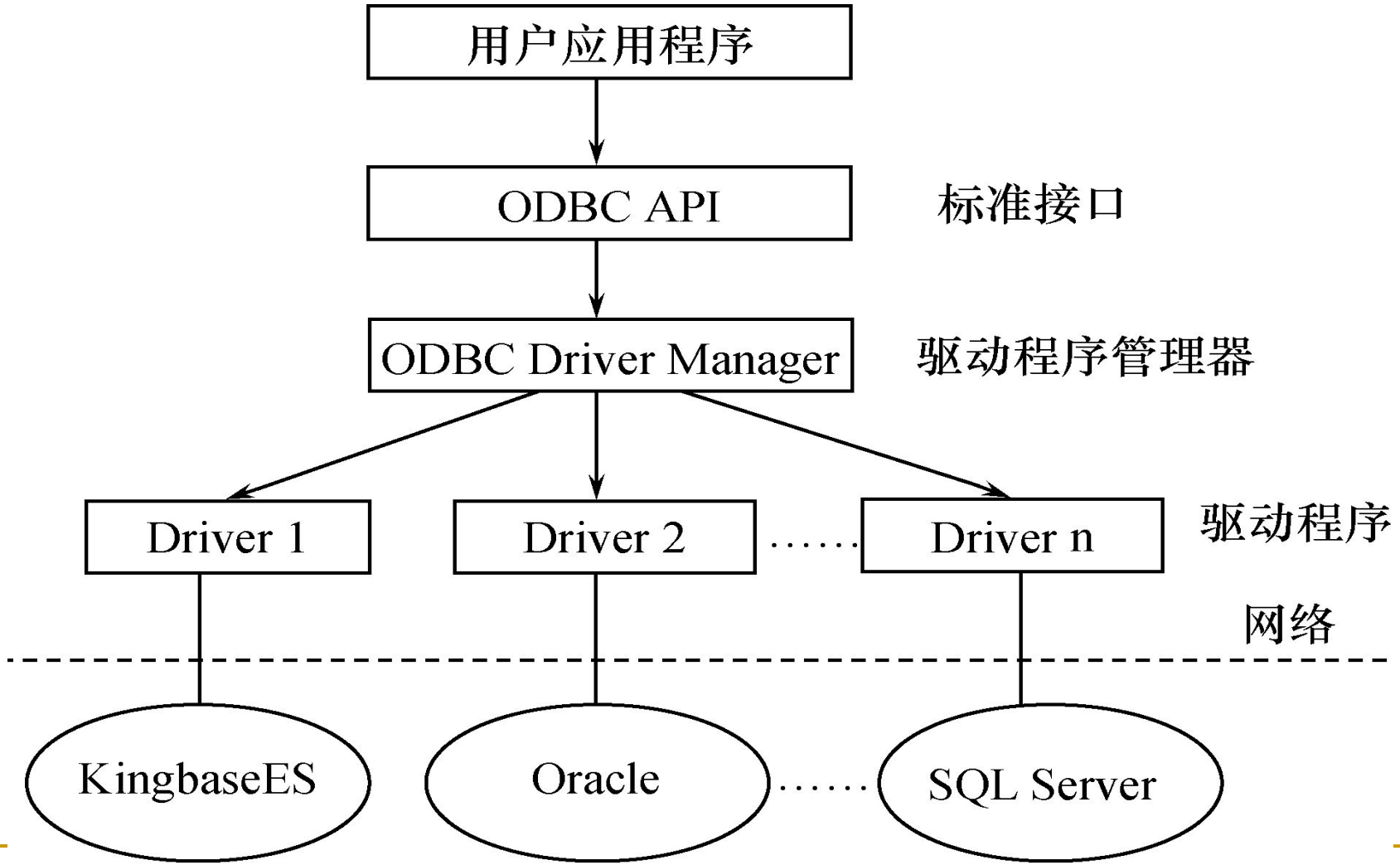
4.2 ODBC编程

- ODBC
  - 是微软公司开放服务体系(Windows Open Services Architecture，WOSA)中有关数据库的一个组成部分
  - 提供了一组访问数据库的标准API

4.2ODBC编程

- ODBC优点：
  - 移植性好
  - 能同时访问不同的数据库
  - 共享多个数据资源

4.2ODBC编程


用户应用程序
ODBC API
标准接口
ODBC Driver Manager
驱动程序管理器
Driver 1
Driver 2
Driver n
驱动程序
网络
KingbaseES
Oracle
SQL Server

4.2ODBC编程

- ODBC API主要函数

资源和连接管理	SQLAllocHandle() SQLFreeHandle() SQLConnect() SQLDisconnect() …	为环境连接,描述符或者语句分配资源 释放先前分配的资源 建立一个数据库连接 终止一个数据库连接
语句执行	SQLExecDirect() SQLPrePare() SQLExecute() …	直接执行一条SQL语句 准备一条SQL语句,供以后执行 执行一条先前准备好的SQL语句
事务管理	SQLEndTran() SQLCancel() …	终止一个SQL事务 撤消一条SQL语句的执行

4.2ODBC编程

- ODBC API主要函数

参数处理	SQLBindParam() SQLParamData() SQLPutData() …	把程序位置绑定到一个参数值 处理延迟参数值 提供延迟参数值,或者字符串值的一部分
查询结果处理	SQLSetCursorName() SQLGetCursorName() SQLFetch() SQLCloseCursor() …	设定游标名 取游标名 取一行查询结果 关闭游标
查询结果描述	SQLNumResultsCols() SQLdescribeCol() …	确定查询结果的字段数目 描述查询的结果字段

4.2ODBC编程

错误处理	SQLError() ...	获得错误信息

返回值	含义
0	成功完成语句
1	成功完成,带有报警
100	没有发现数据(当取查询结果时)
99	需要数据(没有需要的动态参数)
-1	SQL语句执行期间错误
-2	错误-在调用中提供了非法句柄

4.2ODBC编程

- API的种类

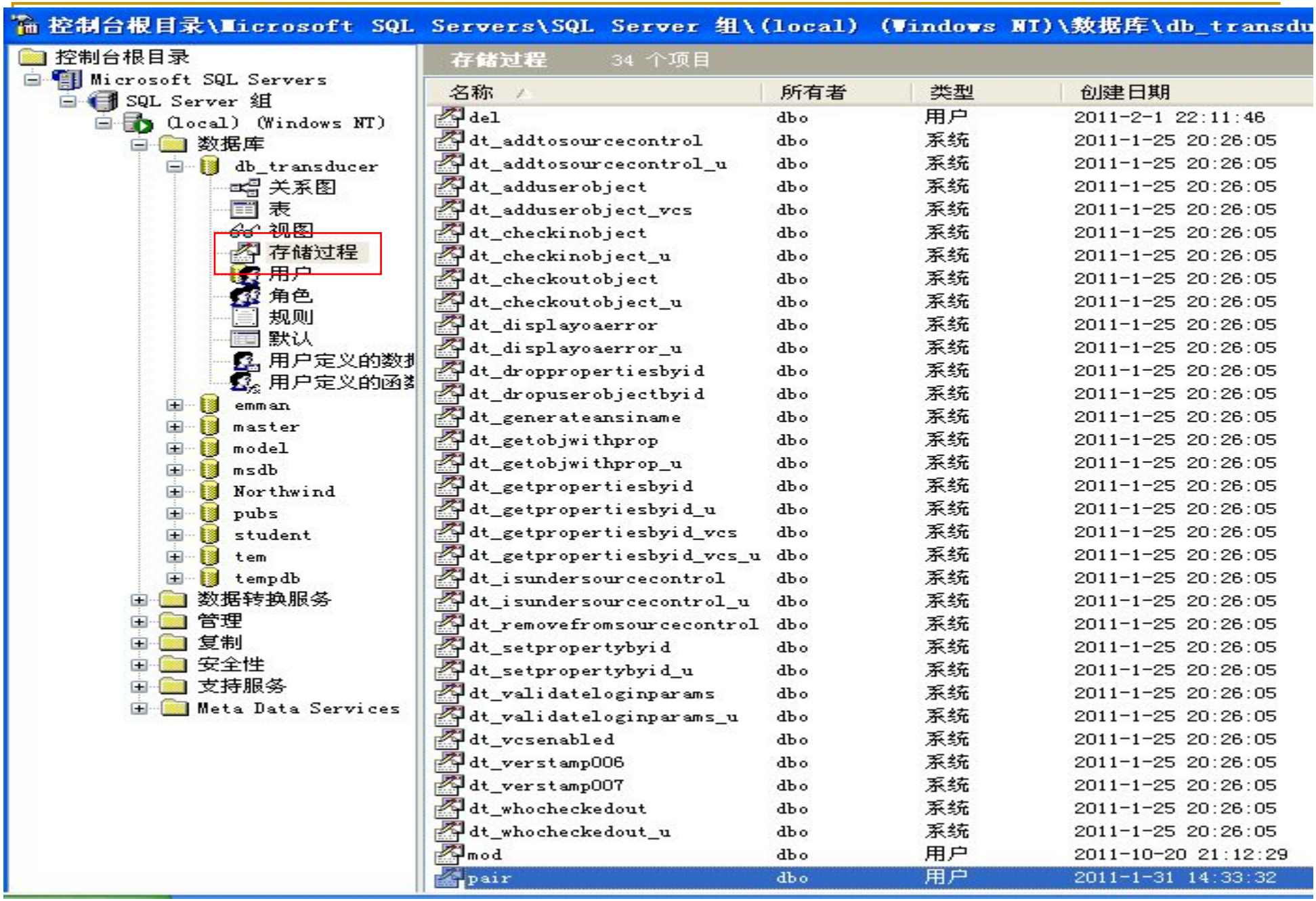
DBMS	API
DB2	ODBC, JDBC
Informix	ODBC, JDBC
SQL Server	ODBC, JDBC, DB Library (dblib)
Oracle	ODBC, JDBC, Oracle Call Interface (OCI)
Sybase	ODBC, JDBC, DB Library (dblib)

第4章 数据库编程

- 4.1 嵌入式SQL
- 4.2 ODBC编程
- *4.3 存储过程*

4.3 存储过程

- 是一组为了完成特定功能的SQL语句集
- 经编译和优化后存储在数据库服务器中
- 建立存储过程可以指定使用的程序设计语言
- PL/SQL（Procedural Language/SQL）是编写存储过程的一种过程语言
  - SQL的扩展
  - 增加了过程化语句功能


控制台根目录\Microsoft SQL Servers\SQL Server 组\(local) (Windows NT)\数据库\db_transdu
控制台根目录
Microsoft SQL Servers
SQL Server 组
(local) (Windows NT)
数据库
db_transducer
关系图
表
视图
存储过程
用户
角色
规则
默认
用户定义的数据
用户定义的函数
emman
master
model
msdb
Northwind
pubs
student
tem
tempdb
数据转换服务
管理
复制
安全性
支持服务
Meta Data Services
存储过程 34 个项目
名称 所有者 类型 创建日期
del dbo 用户 2011-2-1 22:11:46
dt_addtosourcecontrol dbo 系统 2011-1-25 20:26:05
dt_addtosourcecontrol_u dbo 系统 2011-1-25 20:26:05
dt_adduserobject dbo 系统 2011-1-25 20:26:05
dt_adduserobject_vcs dbo 系统 2011-1-25 20:26:05
dt_checkinobject dbo 系统 2011-1-25 20:26:05
dt_checkinobject_u dbo 系统 2011-1-25 20:26:05
dt_checkoutobject dbo 系统 2011-1-25 20:26:05
dt_checkoutobject_u dbo 系统 2011-1-25 20:26:05
dt_displayoaerror dbo 系统 2011-1-25 20:26:05
dt_displayoaerror_u dbo 系统 2011-1-25 20:26:05
dt_droppropertiesbyid dbo 系统 2011-1-25 20:26:05
dt_dropuserobjectbyid dbo 系统 2011-1-25 20:26:05
dt_generateansiname dbo 系统 2011-1-25 20:26:05
dt_getobjwithprop dbo 系统 2011-1-25 20:26:05
dt_getobjwithprop_u dbo 系统 2011-1-25 20:26:05
dt_getpropertiesbyid dbo 系统 2011-1-25 20:26:05
dt_getpropertiesbyid_u dbo 系统 2011-1-25 20:26:05
dt_getpropertiesbyid_vcs dbo 系统 2011-1-25 20:26:05
dt_getpropertiesbyid_vcs_u dbo 系统 2011-1-25 20:26:05
dt_isundersourcecontrol dbo 系统 2011-1-25 20:26:05
dt_isundersourcecontrol_u dbo 系统 2011-1-25 20:26:05
dt_removefromsourcecontrol dbo 系统 2011-1-25 20:26:05
dt_setpropertybyid dbo 系统 2011-1-25 20:26:05
dt_setpropertybyid_u dbo 系统 2011-1-25 20:26:05
dt_validateloginparams dbo 系统 2011-1-25 20:26:05
dt_validateloginparams_u dbo 系统 2011-1-25 20:26:05
dt_vcsenabled dbo 系统 2011-1-25 20:26:05
dt_verstamp006 dbo 系统 2011-1-25 20:26:05
dt_verstamp007 dbo 系统 2011-1-25 20:26:05
dt_whocheckedout dbo 系统 2011-1-25 20:26:05
dt_whocheckedout_u dbo 系统 2011-1-25 20:26:05
mod dbo 用户 2011-10-20 21:12:29
pair dbo 用户 2011-1-31 14:33:32

存储过程属性 — pair

常规

名称(N): pair

权限(P)...

所有者: dbo

创建日期: 2011-1-31 14:33:32

文本(T):

```
CREATE procedure pair @operator_user varchar(10), @pairstd INT, @status_value INT OUT

as
  declare @batcha varchar(20)
  declare @ida varchar(20)
  declare @batchb varchar(20)
  declare @idb  varchar(20)
  declare @cold_recv  decimal(24,3)
  declare @cold_emit  decimal(24,3)
  declare @hot_recv  decimal(24,3)
  declare @hot_emit  decimal(24,3)


DECLARE @pair_cur CURSOR
set  @pair_cur = cursor scroll for
select pair_cold_emit pair_cold_recv pair_hot_emit pair_hot_recv from pair_std
OPEN @pair_cur
FETCH NEXT  FROM @pair_cur into @cold_recv, @cold_emit, @hot_recv, @hot_emit
if @@FETCH_STATUS != 0
  BEGIN
  select @status_value=4
  return
  END
CLOSE @pair_cur
```

1, 12/82

检查语法(C)

确定 取消 应用 帮助

4.3 存储过程-PL/SQL

- PL/SQL基本结构是块
  - 块之间可以互相嵌套
  - 每个块完成一个逻辑操作
- PL/SQL块的基本结构

  定义部分

  执行部分

4.3 存储过程-PL/SQL

- 定义部分

  DECLARE

  ------变量、常量、游标、异常等

  - 定义的变量、常量等只能在该基本块中使用
  - 当基本块执行结束时，定义就不再存在

4.3 存储过程-PL/SQL

- 执行部分

  BEGIN

  ------SQL语句、PL/SQL的流程控制语句

  EXCEPTION

  ------异常处理部分

  END;

4.3 存储过程

- 创建存储过程

  CREATE Procedure 过程名（[参数1，参数2，...]）

  AS

  <PL/SQL块>;

4.3 存储过程

[例11]　利用存储过程来实现下面的应用: 从一个账户转指定数额的款项到另一个账户中。

```
CREATE PROCEDURE TRANSFER(inAccount INT , outAccount INT ,
 amount FLOAT)
AS DECLARE
        totalDeposit FLOAT;
BEGIN                          /* 检查转出账户的余额 */
      SELECT total INTO totalDeposit
      FROM ACCOUNT WHERE ACCOUNTNUM=outAccount;
      IF totalDeposit IS NULL THEN   /* 账户不存在或账户中没有存款 */
         ROLLBACK;
         RETURN;
      END IF;
```

4.3 存储过程

```
IF totalDeposit < amount THEN    /* 账户账户存款不足 */
        ROLLBACK;
        RETURN;
   END IF;
   UPDATE account SET total=total-amount
   WHERE ACCOUNTNUM=outAccount;
                                  /* 修改转出账户，减去转出额 */
  UPDATE account SET total=total + amount WHERE
   ACCOUNTNUM=inAccount;
                                  /* 修改转入账户，增加转出额 */
   COMMIT;                        /* 提交转账事务 */
END;
```

4.3 存储过程

- 执行存储过程

  CALL/PERFORM Procedure 过程名([参数1，参数2，...])；

- 删除存储过程

DROP PROCEDURE 过程名（）；

4.3 存储过程

- 存储过程的优点
  - 经编译和优化后存储在数据库服务器中，运行效率高
  - 降低客户机和服务器之间的通信量
  - 有利于集中控制，方便维护

小结

- 嵌入式SQL
  - 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

- 嵌入式SQL
  - 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
  - 动态SQL

小结

- 存储过程
  - SQL语句集，采用PL/SQL语言编写的程序片段
  - 经编译和优化后存储在数据库服务器中，运行效率高
  - 降低客户机和服务器之间的通信量
  - 有利于集中控制，方便维护

小结

- ODBC
  - ODBC目的：为了提高应用系统与数据库平台的独立性，使得应用系统的移植变得容易
  - ODBC优点：
    - 使得应用系统的开发与数据库平台的选择、数据库设计等工作并行进行
    - 方便移植
    - 大大缩短整个系统的开发时间