第八章 数据库编程

思考：SQL语言有什么缺陷？

本章内容
应用系统如何对数据库进行操作

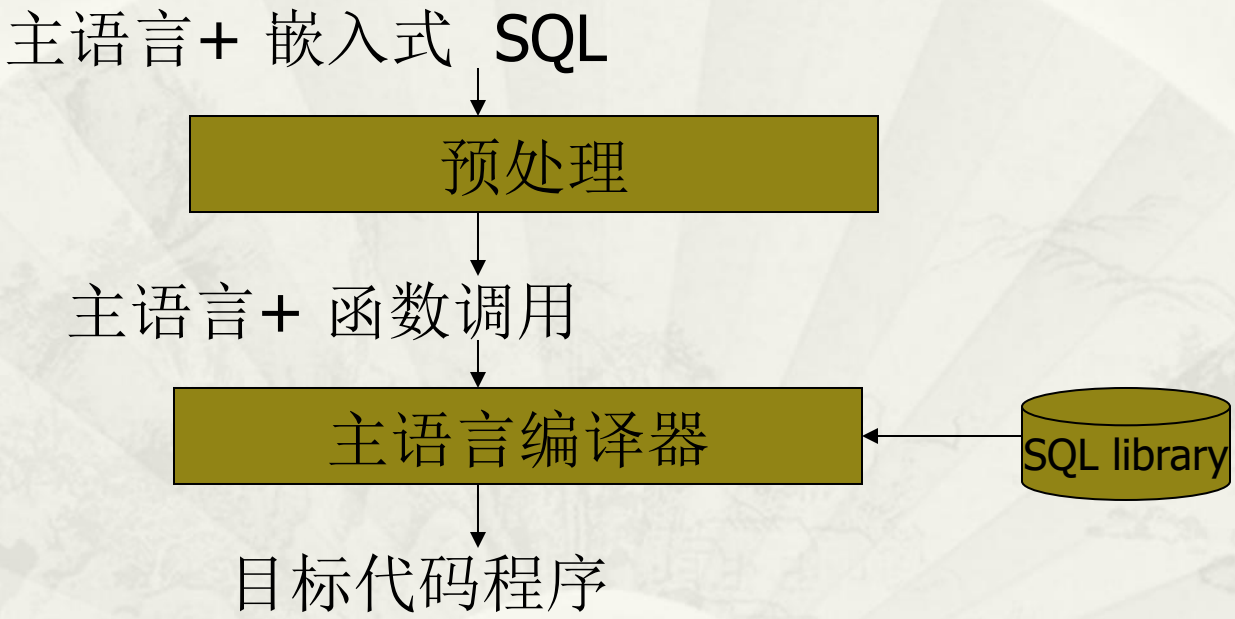
- **嵌入式SQL**
- **存贮过程**
- SQL/API（Application Programming Interface）一组函数和程序 SQL/CLI: “Call-Level Interface”从宿主语言主程序中调用一个SQL DBMS库，而SQL语句是这个调用的参数。
  - ODBC
  - JDBC

SQL概述——嵌入式SQL

- 将SQL语句嵌入到高级语言（宿主语言）
- 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
- 需要预编译，将嵌入的SQL语句转化为宿主语言编译器能处理的语句

嵌入式SQL —— 系统实现

* 实现框图：



嵌入式SQL

- 将SQL访问数据库的能力，与宿主语言的过程化处理的能力进行综合
- 将SQL语句嵌入宿主语言中

产生的问题：

1）宿主语言如何识别SQL语句？

2）SQL语句如何识别宿主语言变量？

3）如何进行数据交换？

4）集合操作如何转换为记录操作？

嵌入式SQL——形式

- SQL语句加上前缀，区别于宿主语言的语句
  - Exec Sql <sql语句>;
  - ex.： Exec sql Delete From Student;
- 预编译
  - 将具有前缀的语句，转换成宿主语言的调用语句
  - 由宿主语言的编译器生成目标码

嵌入式SQL——SQLCA

- SQLCA——SQL Communication Area
- sql通讯区域
  - 数据库工作单元←→宿主语言工作单元之间的通讯区域
  - 宿主语言→DB：请求、参数
  - DB→宿主语言：执行的状态信息、出错信息
  - DB→宿主语言：（查询）执行的结果

嵌入式SQL——SQLCA

- Sqlca.sqlcode (Sybase 数据库):
  - error code
    - 0: success
    - < 0 : fail
    - 100: row not found
- Sqlca.sqlstate:
  - 标准（ISO）返回信息码

    00000: no error

    02000: could not be found

嵌入式SQL——变量

SQL语句如何识别宿主语言的变量？

* 主变量（共享变量）
  * 宿主语言语句
  * 嵌入的SQL语句
  * SQL语句中主变量前加冒号：作为标志
* 宿主变量
  * 仅用于宿主语言的语句中

嵌入式SQL——变量

- 主变量的定义

  Exec Sql Begin Declare Setction;

  …

  Exec Sql End Declare Setction;

嵌入式SQL——游标(Cursor)

- 数据库操作的结果分为单记录和多记录
- 单记录：变量 select…into…
- 多记录：游标（游标的定义，打开和关闭）

- 游标的作用：将多行数据集转换为单行数据
  - 便于宿主语言处理

嵌入式SQL——结构

- 由宿主语言处理数据
  - 宿主变量
  - 共享变量
- 由内嵌的SQL语句处理数据库的访问
  - 共享变量

嵌入式SQL——结构

```
……
Exec sql Include Sqlca;
Exec sql Begin declare Section;
    Char sno(5);
    Char cno(3);
    Int     grade;
Exec sql End Declare Section;
main(){
    Exec sql Declare C1 Cursor For
            Select sno,cno,grade
            From SC;
    Exec sql Open C1;
```

```
For(;;){
    Exec sql Fetch C1 into :sno,:cno,:grade
    if(sqlca.sqlcode <> SUCCESS)
        break;
    printf("sno:%s,cno:%s:%d",sno,cno,grade);
    }
    Exec sql Close C1;
}
```

光标的定义，打开和关闭

嵌入式SQL——简单操作

- 说明性语句
- 数据定义语句
- 数据控制语句
- 涉及单行的数据操纵语句

嵌入式SQL——简单操作

- 说明性语句

  Exec sql Begin Declare Setcion;

  ……

  Exec sql End Declare Setcion;

嵌入式SQL——简单操作

- 涉及变量的处理
  - 变量名以‘:’标识，区别于数据库中的对象（表名、视图名、属性名……）

嵌入式SQL——简单操作

- 查询

  Select …

  Into <共享变量名>[<指示变量>],…

  From …

  Where …

  Group By …

  Having …

  Order By ...

指示变量：用来指示所指的主变量的值或条件，例如主变量是否为空值？是否被截断？

嵌入式SQL——简单操作

```
Exec sql Select sno,cno,grade
    Into :sno,:cno,:grade:grade_id
    From sc
    Where    sno = :given_sno and
             cno = :given_cno;
```

<0示grade为Null

指示变量

嵌入式SQL——游标的使用

- 对象
  - 返回多行数据的查询结果集
  - 根据查询结果修改或删除单个元组的操作（Current形式）

嵌入式SQL——游标的使用

- 方法
  - 定义游标
    - Exec sql Declare <游标名> Cursor For <查询块>;
  - 打开游标
    - Exec sql Open <游标名>;　//执行查询
      //定位于第一行的前一行
  - 移动指针/获取数据
    - Exec sql Fetch <游标名> Into <共享变量名> <指示变量名>,...

嵌入式SQL——游标的使用

- 判断是否到底
  - sqlca.sqlcode
- 关闭游标
  - Exec sql Close <游标名>;

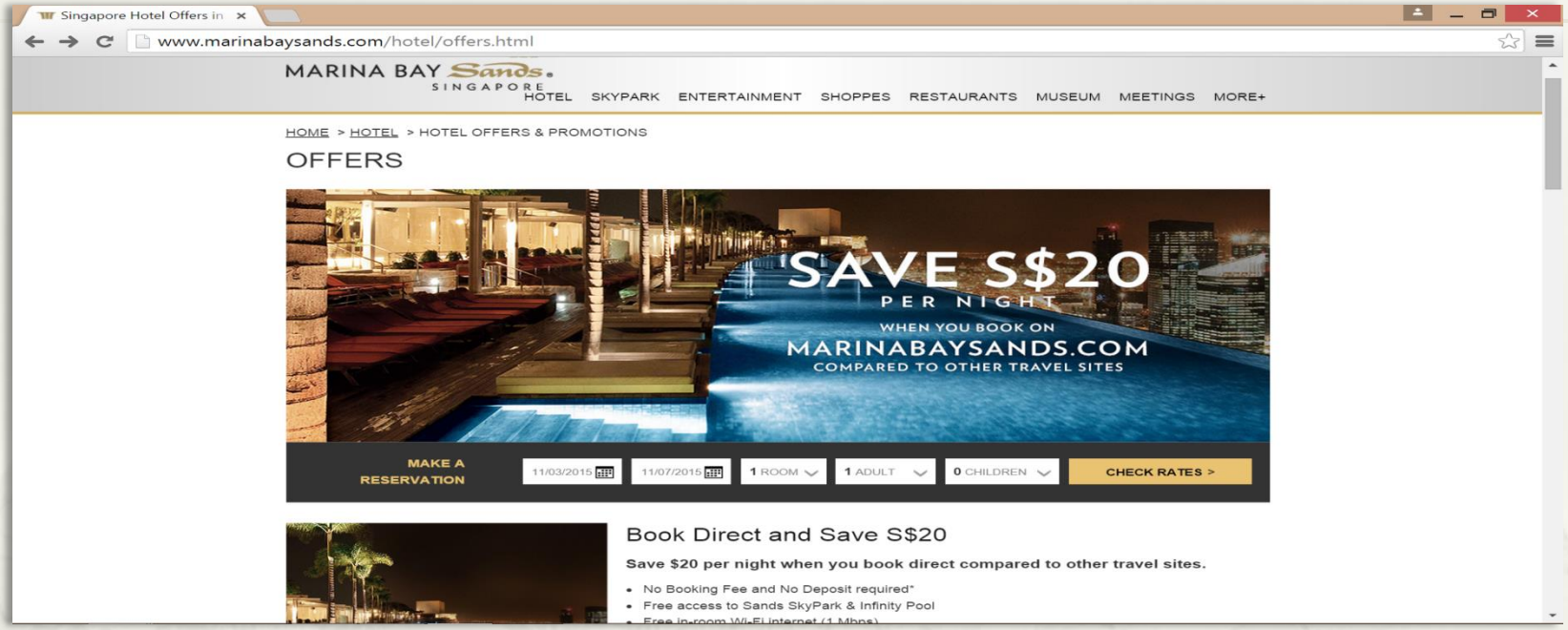
嵌入式SQL——游标的使用

- Current 形式的 Update
- 定义游标
  - Exec sql Declare <游标名> Cursor for
    <查询块> For Update Of <属性名>;
- 打开游标
- Fetch游标
- Update
  - Exec sql Update ……Where Current Of <游标名>

嵌入式SQL——游标的使用

- Current 形式的 Delete
- 定义游标
  - Exec sql Declare <游标名> Cursor for <查询块>;
- 打开游标
- Fetch游标
- Delete
  - Exec sql delete From … Where Current Of <游标名>

嵌入式SQL——动态SQL



* SQL 语句在编译时未知
* 在运行时才知道

嵌入式SQL——动态SQL

主语言程序：

接受用户输入的字符串

转换成可执行的SQL语句

两种方式：

* Exec Sql Prepare <query name> FROM <text of the query>;

  Exec Sql execute <query name>;

* Exec Sql Execute Immediate <text of the query>;

程序实例： 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 DECLARE SECTION;    /*主变量说明开始*/
    char Deptname[20];
    char Hsno[9];
    char Hsname[20];
    char Hssex[2];
    int HSage;
    int NEWAGE;
EXEC SQL END DECLARE SECTION;      /*主变量说明结束*/
long SQLCODE;
EXEC SQL INCLUDE SQLCA;            /*定义SQL通信区*/
```

```
int main(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     /*SX对应的语句*/
                FROM Student
                WHERE SDept = :deptname;
        EXEC SQL OPEN SX;       /*打开游标SX，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*/
        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');
```

```
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*/
}
```

过程化SQL

- 关系数据库管理系统自己的过程化语言。
- 例如：Oracle PL/SQL, Microsoft SQL Server 的 Transact-SQL, IBM DB2 的SQL PL 等

过程化SQL 基本结构

* 定义部分

DECLARE 变量，常量，游标等

* 执行部分

BEGIN

SQL语句，过程化语句， 异常处理部分

END

过程化SQL的用处

SQL2003标准给出了基于过程化SQL，来实现：

* 存贮过程
* 函数

PL/SQL: 数据库过程化语言

基本结构

* 定义部分

DECLARE

变量,常量,异常等说明

* 执行部分

BEGIN

EXCEPTION

END

PL/SQL: 数据库过程化语言

变量常量的定义

- 变量定义: 变量名 数据类型[NOT NULL]:初值表达式
- 常量定义: 常量名 数据类型 CONSTANT:=常量表达式

赋值语句

- 变量名称:=表达式

PL/SQL: 数据库过程化语言

* 各种控制语句
* IF … THEN …END IF
* IF … THEN … ELSE … END IF
* LOOP … END LOOP
* WHILE … END LOOP
* FOR… IN… LOOP … END LOOP

存贮过程的创建,执行和删除

- CREATE Procedure 过程名([参数1,参数2,…]) AS <PL/SQL>块
- CALL/PERFORM Procedure过程名(参数列表) 或EXEC SQL CALL Procedure 过程名
- DROP PROCEDURE 过程名()

存贮过程的实例

* CREATE PROCEDURE **TRANSFER** (inAccount INT,outAccount INT,amount FLOAT) AS

  DECLARE /*定义存储过程TRANSFER，其参数为转入账户、转出账户、转账额度*/

  totalDepositOut FLOAT

  totalDepositIn Float;

  inAccountnum INT;

  BEGIN ….. END

存贮过程的实例 （续）

```
BEGIN                          /*检查转出账户的余额 */
SELECT Total INTO totalDepositOut FROM Account
WHERE accountnum=outAccount;
 IF totalDepositOut IS NULL THEN
  /*如果转出账户不存在或账户中没有存款*/
     ROLLBACK;          /*回滚事务*/
      RETURN;
END IF;
```

存贮过程的实例（续）

```
IF totalDepositOut< amount THEN          /*如果账户存款不足*/
        ROLLBACK;                        /*回滚事务*/
        RETURN;
END IF;
SELECT Accountnum INTO inAccountnum FROM Account
WHERE accountnum=inAccount;
IF inAccounum IS NULL THEN       /*如果转入账户不存在*/
        ROLLBACK;                /*回滚事务*/
        RETURN;
ENDIF;
```

存贮过程的实例 （续）

```
UPDATE Account SET total=total-amount
WHERE accountnum=outAccount;
                                    /* 修改转出账户余额，减去转出额 */
UPDATE Account SET total=total + amount
WHERE   accountnum=inAccount;
                                    /* 修改转入账户余额，增加转入额 */
COMMIT;                             /* 提交转账事务 */
END;
```

存贮过程的优点

* 运行效率高.提供在服务器端快速执行SQL语句的有效途径.
* 降低客户机和服务器之间的通信量.
* 方便实施企业规则

执行:

* CALL Procedure TRANSFER(0100388111, 2333999,1000)

函数

- 函数的定义

CREATE FUNCTION <name> (<parameters>)
RETURNS <type> AS < local declarations
function body>;

- 函数的执行

作为一个表达式放到适合返回值的地方.

- 修改函数

ALTER FUNCTION 函数名1 rename to 函数名2

课堂练习

* 用存贮过程实现学生通过课程名来搜索该课程信息

```
Create procedure searchByCname(IN course_name
char(10)) as
Select C.CID, C.cname, C.teacher
from courses C
Where C.cname=course_name
```

调用该存贮过程

* 在交互式环境中

```
CALL searchByCname(Database)
```

* 在嵌入式环境中

```
EXEC SQL BEGIN DECLARE SECTION
CHAR coursename[10]
EXEC SQL END DECLARE SECTION
EXEC SQL CALL seachByCname(:coursename)
```

SQL/API 的样例

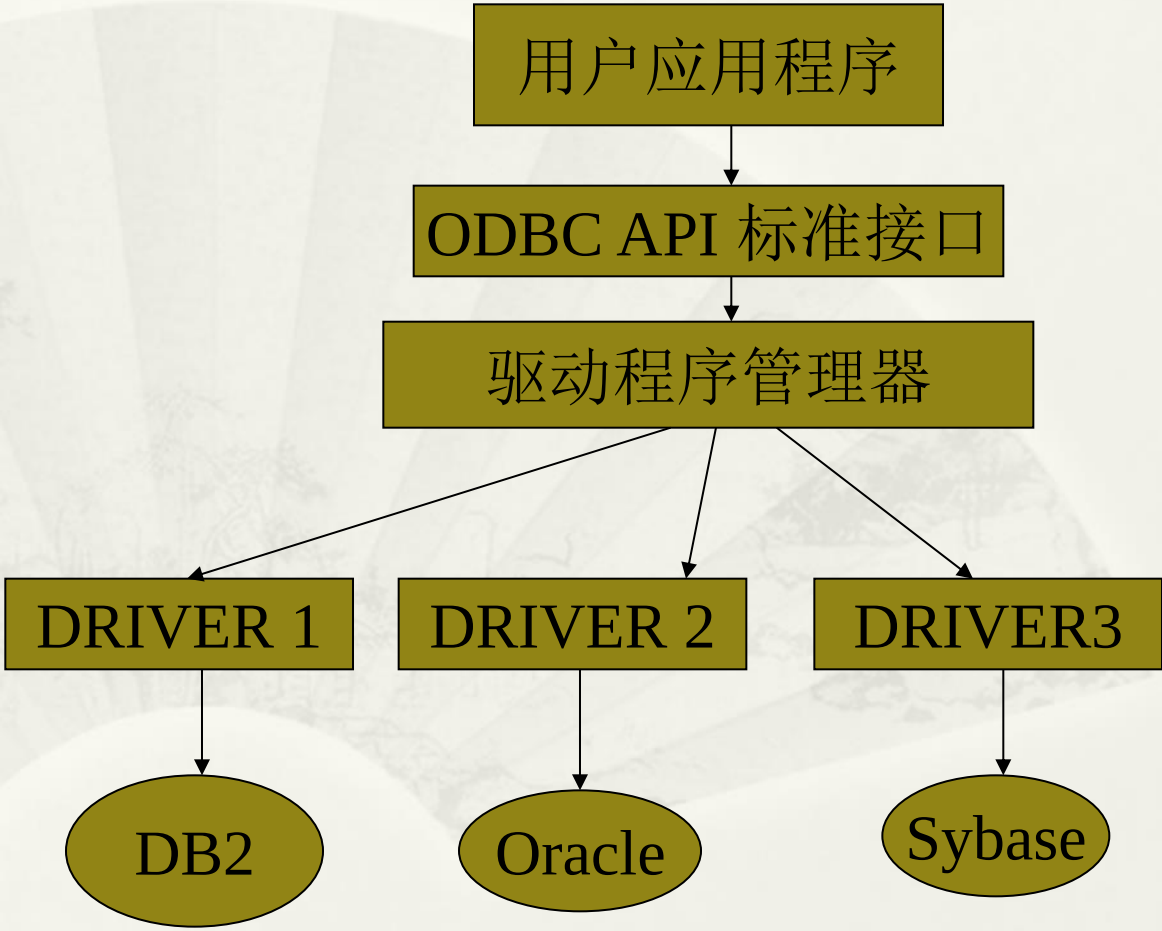
```
#include “sqlcli.h”
SQLHSTMT        hstmt;
…
SQLPrepare(hstmt,“Insert Into customer values(……)”);
SQLExcute(hstmt);
…
```

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**一组访问数据库的应用程序编程接口**（Application Programming Interface，API ）

ODBC工作原理

- 请求连接数据库
- 发送SQL语句
- 为结果分配存贮空间
- 获取执行结果或错误信息
- 进行数据处理并提交结果
- 事务提交或回滚
- 断开连接



应用场景

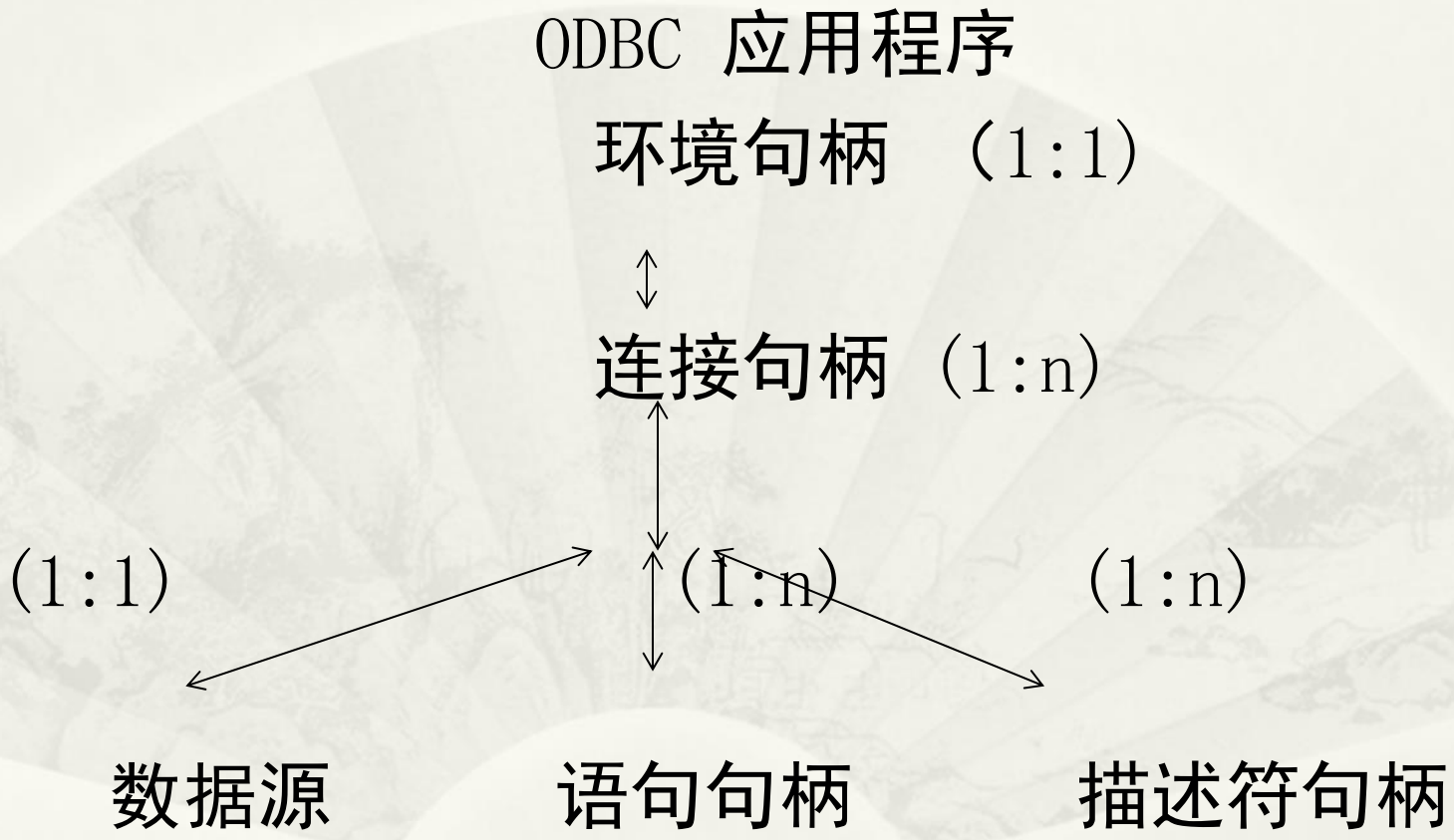
- 假设某个学校在SQL Server和KingbaseES上创建了两个数据库：学校人事数据库和教学科研数据库。
  - 学校的信息系统要从这两个数据库中存取数据
  - 为了方便地与两个数据库连接，为学校人事数据库创建一个数据源名PERSON，为教学科研数据库创建一个名为EDU的数据源
  - 当要访问每一个数据库时，只要与PERSON和EDU连接即可，不需要记住使用的驱动程序、服务器名称、数据库名 。

ODBC 数据结构

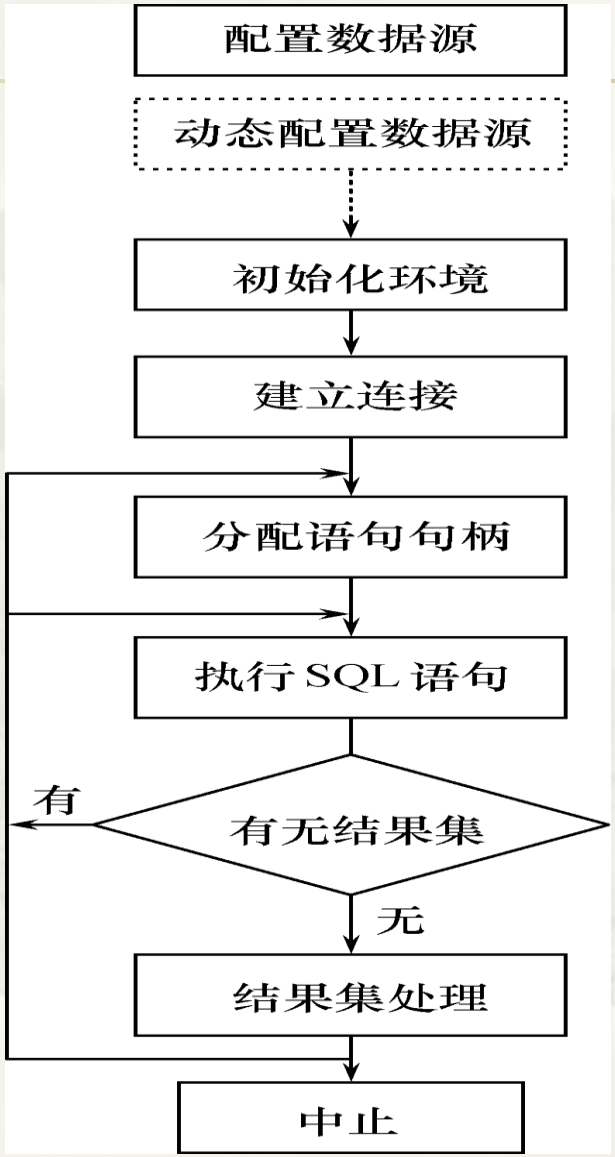
- *Environments* : represent the DBMS installation. （环境句柄）
- *Connections* : （连接句柄）logins to the database.
- *Statements* : （语句句柄） SQL statements to be passed to a connection.
- *Descriptions* :描述符句柄 ，描述SQL语句的参数，结果集的元数据集合

- 句柄是32位整数值，代表一个指针

应用程序句柄之间的关系



ODBC的工作流程



JDBC

* JAVA 作为主语言
* *Java Database Connectivity*（JDBC）是一个库，定义SQL嵌入到JAVA的一系列函数

驱动器管理与连接到数据源

The JDBC classes

```
import java.sql.*;
Class.forName(com.mysql.jdbc.Driver);
Connection myCon =
  DriverManager.getConnection(…);
```

加载mySql驱动器

Loaded by forName

URL of the database your name, and password go here.

执行SQL语句

* JDBC 有两种类型的语句类:

1. *Statement* = an object 可以有一个SQL语句作为参数，也可以没有。
2. *PreparedStatement* = an object 已经有了一个SQL语句作为参数。

创建语句

* The Connection class 有方法来创建 Statement 和 preparedStatement.

```
Statement stat1 = myCon.createStatement();
PreparedStatement stat2 =
  myCon.createStatement(
      " SELECT sid, sname FROM Students " +
      " WHERE Sdept= 'cs' "
  );
```

createStatement 不带参数，返回 a Statement 对象; 带参数，则返回 a PreparedStatement 对象.

执行 SQL 语句

- JDBC 查询和更新是不同的。
- Statement 和 PreparedStatement 对象有方法 executeQuery 和 executeUpdate.
  - Statements类对象：需要参数才可以查询或更新。
  - PreparedStatements对象：不需要属性。

举例：更新

* stat1 是一个 Statement类对象：

```
stat1.executeUpdate(
  ”INSERT INTO Students ” +
  ”VALUES(01111,’Brass Rail’,25)”
);
```

举例：查询

stat2 是一个PreparedStatement对象，已经有参数，即查询语句 “SELECT sid, sname FROM Students WHERE Sdept=‘cs’ ”。

* executeQuery 返回一个ResultSet 类对象。

```
ResultSet CSstudents =
  stat2.executeQuery();
```

存取结果集

* ResultSet类型的对象像一个光标。
* 该类型对象有方法：Method next() 会把光标移到下一个，如果结束，
  * next() 返回 值 false.

存取元组中的分量

- ResultSet 对象存取某一个分量，使用：
- Method getX（i），X 是某种类型， i 是第几个分量，它的类型是X. 例如：getString(1)，getFloat(2)

举例：存取元组分量

* CSstudents 是ResultSet对象
  是查询句 “SELECT sid, sname FROM Students WHERE Sdept=‘cs’ ”的结果。

```
while ( CSstudents.next() ) {
  SID = Menu.getString(1);
  Sname = Menu.getString(2);
  /*对变量 SID 以及Sname 处理*/}
```

小结: 数据库编程

嵌入式SQL: 把SQL嵌入到传统编程语言中.

存贮过程/函数: 提供一种标准的面向SQL的编程语言.

数据库之间互相连接的规范(一组函数): **ODBC,JDBC**

作业

* 课堂作业：PYTHON和SQLite语言如何结合？

答案：一般步骤

```
# import sqlite3 module
import sqlite3
# create a connection object
conn = sqlite3.connect('<DBname>.<db/sqlite/db3>')
# create a cursor object
cur = conn.cursor()
# call the cursor's execute() method to perform SQL commands
cur.execute('<SQL statement>')
# save (commit) the changes
conn.commit()
# close the connection
conn.close()
```

课堂作业

- 用Python实现和mydb数据库的连接，并执行如下查询语句：

1. 查找年龄小于25岁的学生，并按年龄排序
2. 查找每个系学生的最大年龄

创建一个表

```
cur.execute('''DROP TABLE IF EXISTS students;''')
cur.execute('''CREATE TABLE students(
      sid int PRIMARY KEY,
      name char[10] NOT NULL,
      dept char[2],
      age int DEFAULT 20); ''')
conn.commit()
```

- Triple-quotes – for string literals that span multiple lines.
- We need to commit once we are certain about the changes.

  In SQLite3, before committing, a *journal* file is maintained.

插入

```
* cur.execute('''INSERT INTO students
  VALUES(?,?,?,?)''', (1,'smith','cs',35))
* conn.commit()
```

* Use “?” as a placeholder and provide the values in a tuple

插入多个元组

```
* tmp = [(2,'martin','cs',20),
*        (3,'Bern','cs',28),
*        (4,'Hone','cs',23),
*        (5,'Lihong','ee',18),
*        (6,'John','ee',29),
*        (7,'Flower','ee',25),
*        (8,'martin','ee',20),
*        (9,'smith','ma',21),
*        (10,'wang','ma',19)]
* cur.executemany('INSERT INTO students
  VALUES(?,?,?,?)', tmp)
* conn.commit()
```

Batch insert multiple rows, preparing the tuples of values in a list and call `cur.executemany()`

删除

```
* cur.execute('DELETE FROM students WHERE age > 30')
* conn.commit()
```

修改

```
* cur.execute('''UPDATE students
*                SET age = 17
*                WHERE name = 'martin' AND dept = “cs”
*            ''')
* conn.commit()
```

查询

```
cur.execute('''SELECT * FROM students
               WHERE age < 25 ORDER BY age
            ''')
for row in cur:
      print row
```

- To retrieve data after executing a SELECT statement, the cursor can be treated as an iterator.

```
cur.execute('''SELECT dept, MAX(age)
               FROM students GROUP BY dept
            ''')
for row in cur:
      print "dept. = ", row[0]
      print "MAX(age) = ", row[1], "\n"
```