



2024年春季学期

数据库系统概论

An Introduction to Database Systems

第八章 数据库编程

中国科学技术大学 大数据学院

黄振亚, huangzhy@ustc.edu.cn



嵌入式 SQL

54

- 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 不使用游标的SQL语句
- 8.1.4 使用游标的SQL语句
- 8.1.5 动态SQL
- 8.1.6 小结



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
- 无法满足要到执行时才能够确定要提交的SQL语句

□ 动态嵌入式SQL

- 动机：编译阶段无法获得完整的SQL语句，需要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



动态SQL简介（续）

56

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



一、使用SQL语句主变量

57

- **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



使用SQL语句主变量（续）

58

[例8.7] 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 */
```



二、动态参数

59

- 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
 - PERPARE语句准备主变量
 - 执行语句EXECUTE绑定数据或主变量



动态参数（续）

60

- 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>;]**



其他语言的嵌入式SQL

62

- 其他语言如Python、PHP、JAVA都可以作为主语言进行嵌入式SQL
- 这几种语言连接数据库的流程都大同小异，主要包括：
 - 引入DB-API模块(如MySQL、Oracle，而php不需要引入模块)
 - 获取与数据库的连接
 - 执行SQL语句和存储过程
 - 关闭数据库连接



Python连接MySQL例子

63

```
import MySQLdb          #引入相应模块
def mysql_conn():
    try:
        # 连接数据库：host为要连接的数据库服务器主机名；user为数据库的用户名，默认为当前用户；passwd连接密码，db为连接的数据库名
        conn = MySQLdb.connect(host = '192.168.8.100',user = 'mysql',passwd = '123456',db = 'mydatabase')
        cursor = conn.cursor() #使用cursor方法获取游标，用于访问和操作数据库中的数据

        # 在mysql中要进行的操作，如查询语句
        sql = "SELECT COUNT(*) FROM mydatabase.user"
        cursor.execute(sql) #执行sql语句的操作
```



Python连接MySQL例子（续）

64

取出游标（指针）结果集中的所有行，返回的结果集一个元组(tuples)
，除了fetchall()，还支持fetchone()，取出下一个结果，还有
fetchmany(size)，取出多行（size行）结果

```
alldata = cursor.fetchall()
```

```
count = alldata[0][0]
```

SQL更新语句

```
sql2 = "UPDATE mydatabase SET age = 20 WHERE user.id = %s"
```

```
age = '007'          # 主变量
```

```
cursor.execute(sql2,age) #传入主变量，执行更新语句
```

```
cursor.close() #关闭游标指针释放资源
```

```
conn.close() #关闭数据库连接并释放资源
```

```
print count
```

```
except Exception,e:
```

```
print "Can not Connect to mysql server"
```



PHP连接MySQL例子

65

```
<?php
```

```
//连接数据库
```

```
$conn=mysql_connect("localhost", "root", "password");
```

```
//执行查询语句
```

```
$result=mysql_db_query("Database", "SELECT * FROM  
`info`", $conn);
```

```
// 获取查询结果
```

```
$row=mysql_fetch_row($result);
```

```
echo '<font face="verdana">';
```

```
echo '<table border="1" cellpadding="1" cellspacing="2">';
```



PHP连接MySQL例子（续）

66

// 定位到第一条记录

```
mysql_data_seek($result, 0);
```

// 循环取出记录

```
while ($row=mysql_fetch_row($result))
```

```
{
```

```
    echo "<tr></b>";
```

```
    for ($i=0; $i<mysql_num_fields($result); $i++ )
```

```
    {
```

```
        echo '<td bgcolor="#00FF00">';
```

```
        echo $row[$i];
```

```
        echo '</td>';
```

```
    }
```

```
    echo "</tr></b>";
```

```
}
```



PHP连接MySQL例子（续）

67

```
echo "</table></b>";  
echo "</font>";  
// 释放资源  
mysql_free_result($result);  
// 关闭连接  
mysql_close($conn);  
?>
```



JAVA连接MySQL例子

68

```
import java.sql.*; //引入模块
public class JDBCtest {
public static void main(String[] args){
// 驱动程序名
String driver = "com.mysql.jdbc.Driver";
// URL指向要访问的数据库名scutes
String url = "jdbc:mysql://127.0.0.1:3306/scutes";
// MySQL配置时的用户名
String user = "root";
// MySQL配置时的密码
String password = "root";
try {
// 加载驱动程序
Class.forName(driver);
```




JAVA连接MySQL例子（续）

69

// 连续数据库

```
Connection conn = DriverManager.getConnection(url, user, password);  
if(!conn.isClosed())  
    System.out.println("Succeeded connecting to the Database!");
```

// statement用来执行SQL语句

```
Statement statement = conn.createStatement();
```

// 要执行的SQL语句

```
String sql = "select * from student";
```

// 结果集

```
ResultSet rs = statement.executeQuery(sql);
```



JAVA连接MySQL例子（续）

70

```
rs.close(); //断开游标，释放资源
conn.close(); //断开数据库连接，释放资源
//各种异常处理
} catch(ClassNotFoundException e) {
    System.out.println("Sorry,can`t find the Driver!");
    e.printStackTrace();
} catch(SQLException e) {
    e.printStackTrace();
} catch(Exception e) {
    e.printStackTrace();
}
}
}
```



8.1 嵌入式SQL

71

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 不使用游标的SQL语句

8.1.4 使用游标的SQL语句

8.1.5 动态SQL

8.1.6 小结



8.1.6 小结

72

- 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
- 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



第八章 数据库编程

75

8.1 嵌入式SQL

8.2 过程化SQL(PL/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



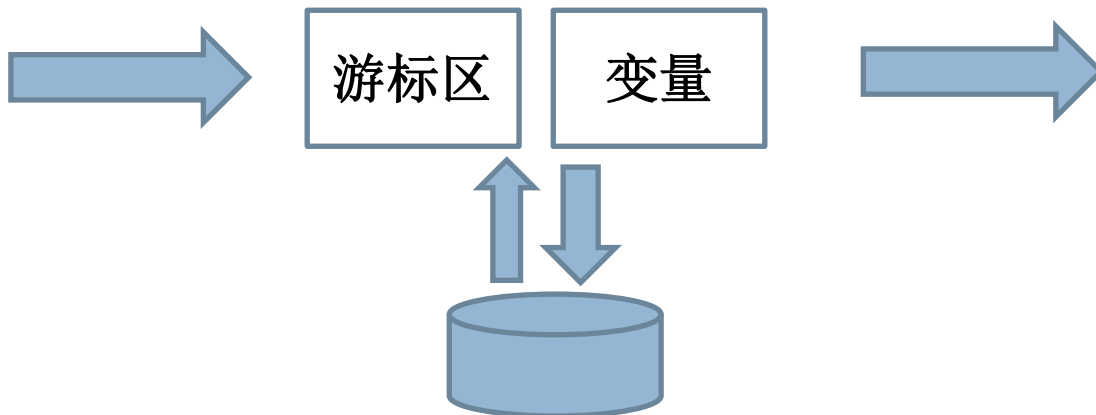
8.2 过程化SQL

76

□ PL/SQL技术路线的优势

- 尽量减少系统之间的通信和协同
- 嵌入式SQL：高级语言主导过程，利用高级语言的控制功能
- 过程化SQL：数据库主导过程，扩展SQL语言功能

高级语言



数据库



8.2 过程化SQL

77

8.2.1 过程化SQL的块结构

8.2.2 变量和常量的定义

8.2.3 流程控制



8.2.1 过程化SQL的块结构

78

- 过程化SQL (PL/SQL)
 - SQL的扩展
 - 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
 - 增加了过程化语句功能
 - 增加变量、常量等定义语句
 - 增加变量赋值语句
 - 增加流程控制语句



过程化SQL的块结构（续）

79

□ 过程化SQL块的基本结构

1. 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

2. 执行部分

BEGIN

SQL语句、过程化SQL(PL/SQL)的流程控制语句

EXCEPTION

异常处理部分

END;



8.2 过程化SQL

80

8.2.1 过程化SQL的块结构

8.2.2 变量和常量的定义

8.2.3 流程控制



8.2.2 变量和常量的定义

81

1. PL/SQL变量定义的语法形式

- 变量名 数据类型 [[NOT NULL]:=初值表达式] 或
- 变量名 数据类型 [[NOT NULL] 初值表达式]

2. 常量定义类似于变量定义

- 常量名 数据类型 **CONSTANT** :=常量表达式
- 常量必须要给一个值，并且该值在存在期间或常量的作用域内不能改变。如果试图修改它，过程化SQL将返回一个异常

3. 赋值语句

- 变量名称 :=表达式
- **SET** 变量名称 = 表达式



8.2.2 变量和常量的定义

□ 举例：变量定义

```
DECLARE          /*定义变量*/  
  
totalDepositOut Float := 0;  
totalDepositIn Float := 0;  
inAccountnum INT NOT NULL;
```

```
delimiter //  
create trigger count_check before insert  
on SC for each row  
begin  
    declare msg varchar(200);  
    if ( 2 <= Any(select count(*) from SC group by cno)) then  
        set msg = "The count of guest is above 2."  
        signal sqlstate 'HY000' SET message_text = msg;  
    end if;  
end //  
delimiter ;
```

□ 举例：常量定义

```
errorMsg string CONSTSNT := “Nested SQL  
error”
```

□ 举例：赋值

```
SET Sno=Sno+1;
```



8.2 过程化SQL

83

8.2.1 过程化SQL的块结构

8.2.2 变量和常量的定义

8.2.3 流程控制



8.2.3 流程控制

84

□ PL/SQL过程化SQL功能

1. 条件控制语句
2. 循环控制语句
3. 错误处理



流程控制（续）

85

1. 条件控制语句

IF-THEN, IF-THEN-ELSE和**嵌套的IF**语句

(1) IF condition THEN

Sequence_of_statements;

END IF;

(2) IF condition THEN

Sequence_of_statements1;

ELSE

Sequence_of_statements2;

END IF;

(3) 在THEN和ELSE子句中还可以再包含IF语句，即IF语句可以嵌套



2. 循环控制语句

LOOP, WHILE-LOOP和FOR-LOOP

(1) 简单的循环语句LOOP

LOOP

Sequence_of_statements;

END LOOP;

多数数据库服务器的过程化SQL都提供EXIT、BREAK或LEAVE等循环结束语句，保证LOOP语句块能够结束



2. 循环控制语句（续）

(2) WHILE-LOOP

WHILE condition LOOP

Sequence_of_statements;

END LOOP;

- 每次执行循环体语句之前，首先对条件进行求值
- 如果条件为真，则执行循环体内的语句序列
- 如果条件为假，则跳过循环并把控制传递给下一个语句



流程控制（续）

2. 循环控制语句（续）

(3) FOR-LOOP

FOR count IN [REVERSE] bound1 ... bound2 LOOP

Sequence_of_statements;

END LOOP;

- | | | |
|--------------------|-------------------|------------------------|
| ■ MySQL | ■ Oracle | ■ MS SQL Server |
| ● WHILE 循环 | ● WHILE 循环 | ● WHILE 循环 |
| ● REPEAT 循环 | ● FOR 循环 | |
| ● LOOP 循环 | ● LOOP 循环 | |



3. 错误处理（Exception）

- 如果过程化SQL在执行时出现异常，则应该让程序在产生异常的语句处停下来，根据异常的类型去执行异常处理语句
- SQL标准对数据库服务器提供什么样的异常处理做出了建议，要求过程化SQL管理器提供完善的异常处理机制



控制结构（续）：定义触发器

90

定义一个BEFORE行级触发器，为教师表Teacher定义完整性规则“教授的工资不得低于4000元，如果低于4000元，自动改为4000元”。

```
CREATE TRIGGER Insert_Or_Update_Sal
  BEFORE INSERT OR UPDATE ON Teacher
  /*触发事件是插入或更新操作*/
  FOR EACH ROW                /*行级触发器*/
  BEGIN                        /*定义触发动作体，是PL/SQL过程块*/
    IF (new.Job='教授') AND (new.Sal < 4000) THEN
      new.Sal :=4000;
    END IF;
  END;
```



例子

```
1  /*判断某个学生是否满足毕业要求 并修改student表中的graduate属性*/
2  /*创建PROCEDURE
3  输入: Fail_LIMIT 挂科次数限制action
4  AVG_GRADE_LIMIT 平均GRADE限制
5  */
6  DELIMITER //
7  • CREATE PROCEDURE GRADUATE (IN Fail_LIMIT int, IN AVG_GRADE_LIMIT int)
8  BEGIN
9      DECLARE FAIL int;
10     DECLARE AVG_GRADE int;
11     DECLARE Sno int DEFAULT 0;
12     loop_label: LOOP
13         SELECT COUNT(*) INTO FAIL FROM SC WHERE SC.Sno = Sno AND SC.Grade < 60;
14         SELECT avg(Grade) INTO AVG_GRADE FROM SC WHERE SC.Sno = Sno AND SC.Grade >= 60;
15         IF FAIL <= Fail_LIMIT AND AVG_GRADE >= AVG_GRADE_LIMIT THEN
16             UPDATE Student SET graduate = 'success' WHERE Student.Sno = Sno;
17         ELSE
18             UPDATE Student SET graduate = 'fail' WHERE Student.Sno = Sno;
19         END IF;
20         SET Sno=Sno+1;
21         IF Sno>=100 THEN
22             LEAVE loop_label;
23         END IF;
24     END LOOP;
25 END; //
26
27 • CALL GRADUATE(1, 70); //
28 • SELECT * FROM student;
29
```

定义部分

执行部分



第八章 数据库编程

92

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



8.3 存储过程和函数

93

8.3.1 存储过程

8.3.2 函数

8.3.3 过程化SQL中的游标



8.3.1 存储过程

□ 存储过程

由过程化SQL语句书写的一段程序，经编译和优化后(命名)存储在数据库服务器中，使用时只要调用即可

□ 存储过程的优点

- (1) 运行效率高
- (2) 降低了客户端和服务端之间的通信量
- (3) 方便实施企业规则



8.3.1 存储过程

95

□ PL/SQL块类型

□ 命名块

- 编译后保存在数据库中，可以被反复调用，运行速度较快。
存储过程和函数是命名块

□ 匿名块

- 每次执行时都要进行编译，它不能被存储到数据库中，也不能在其他过程化SQL块中调用



存储过程（续）

□ 存储过程的用户接口

- (1) 创建存储过程
- (2) 执行存储过程
- (3) 修改存储过程
- (4) 删除存储过程



2. 存储过程的用户接口

97

(1) 创建存储过程

CREATE OR REPLACE PROCEDURE 过程名([参数1,参数2,...]) AS <过程化SQL块>;

- 过程名：数据库服务器合法的对象标识
- 参数列表：用名字来标识调用时给出的参数值，必须指定值的数据类型。参数也可以定义输入参数、输出参数或输入/输出参数，默认为输入参数
- 过程体：一个<过程化SQL块>，包括声明部分和可执行语句部分



存储过程的用户接口（续）

98

- [例8.9] 利用存储过程来实现下面的应用：从账户1转指定数额的款项到账户2中。

```
CREATE PROCEDURE TRANSFER(inAccount INT,  
outAccount INT, amount FLOAT)
```

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

```
AS DECLARE /*定义变量*/
```

```
totalDepositOut Float;
```

```
totalDepositIn Float;
```

```
inAccountnum INT;
```



存储过程的用户接口（续）

99

```
BEGIN                                /*检查转出账户的余额*, 执行SQL/  
  
SELECT Total INTO totalDepositOut FROM Accout  
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;
```



存储过程的用户接口（续）

100

IF totalDepositOut < amount THEN /*如果账户存款不足*/

ROLLBACK; /*回滚事务*/

RETURN;

END IF;

/*检查转入账号*/

SELECT Accountnum INTO inAccountnum FROM Account

WHERE accountnum=inAccount;

IF inAccount IS NULL THEN /*如果转入账户不存在*/

ROLLBACK; /*回滚事务*/

RETURN;

ENDIF;



存储过程的用户接口（续）

101

```
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;
```



存储过程的用户接口（续）

102

（2）执行存储过程

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

- 使用**CALL**或者**PERFORM**等方式激活存储过程的执行
- 在过程化SQL中，数据库服务器支持在过程体中调用其他存储过程

- [例8.10] 从账户01003815868转10000元到01003813828账户中。

CALL PROCEDURE

TRANSFER(01003813828, 01003815868,10000);



存储过程的用户接口（续）

103

（3）修改存储过程

ALTER PROCEDURE 过程名1 RENAME TO 过程名2;

（4）删除存储过程

DROP PROCEDURE 过程名();



8.3 存储过程和函数

104

8.3.1 存储过程

8.3.2 函数

8.3.3 过程化SQL中的游标



8.3.2 函数

- 函数和存储过程的异同
 - 同：都是持久性存储模块
 - 异：函数必须指定返回的类型



函数（续）

106

1.函数的定义语句格式

CREATE OR REPLACE FUNCTION 函数名 ([参数1,参数2,...]) **RETURNS** <类型> **AS** <过程化SQL块>;

2.函数的执行语句格式

CALL/SELECT 函数名 ([参数1,参数2,...]);

3.修改函数

□重命名

ALTER FUNCTION 过程名1 **RENAME TO** 过程名2;

□重新编译

ALTER FUNCTION 过程名 **COMPILE**;



8.3 存储过程和函数

107

8.3.1 存储过程

8.3.2 函数

8.3.3 过程化SQL中的游标



函数（续）

108

过程化SQL中的游标

- 概念与8.1中一致
- 使用方法类似
 - 1. 声明**Declare**
 - 2. 打开**Open**
 - 3. 推进**Fetch**
 - 4. 关闭**Close**

课后在MySQL上尝试



过程化SQL中的游标

```
1  -- 计算给定学生的不及格学分
2  Delimiter //
3  DROP PROCEDURE IF EXISTS cursor_test;
4  CREATE PROCEDURE cursor_test ( IN sn VARCHAR (50), OUT total INT )
5  BEGIN
6  DECLARE state INT DEFAULT 0;
7  DECLARE sn1 VARCHAR(50);
8  DECLARE cred INT;
9  DECLARE
10     ct CURSOR FOR
11     (SELECT sc.sno, credit FROM sc, course WHERE course.cno=sc.cno AND score<60);
12  DECLARE CONTINUE HANDLER FOR NOT FOUND SET state = 1;
13  SET total = 0;
14  OPEN ct;
15  REPEAT
16     FETCH ct INTO sn1, cred;
17     IF state = 0 THEN
18         IF sn1=sn THEN
19             SET total = total + cred;
20         END IF;
21     END IF;
22     UNTIL state = 1
23  END REPEAT;
24  CLOSE ct;
25  END //
26  Delimiter;
```

学号, 不及格学分

mysql80 test

```
1  set @sno='s5';
2  call cursor_test(@sno,@total);
3  select @sno,@total;
```

信息 结果 1 剖析 状态

@sno	@total
s5	5

University

```

1  -- 计算给定学生的GPA
2  Delimiter //
3  DROP FUNCTION IF EXISTS fun;
4  CREATE FUNCTION fun(sn VARCHAR(50))
5  RETURNS FLOAT ←
6  READS SQL DATA
7  BEGIN
8      DECLARE state INT DEFAULT 0; -- cursor结束标记
9      DECLARE grade,cred,total_c,total_g FLOAT DEFAULT 0;
10     DECLARE sn1 VARCHAR(50);
11     DECLARE c_count INT;
12     DECLARE t, gpa FLOAT DEFAULT 0;
13     DECLARE
14         ct CURSOR FOR
15         (SELECT score,credit FROM sc,course c WHERE sc.cno=c.cno AND sno=sn AND score IS NOT NULL);
16     DECLARE CONTINUE HANDLER FOR NOT FOUND SET state = 1;
17     OPEN ct;
18     REPEAT
19         FETCH ct INTO grade,cred; -- 每一门课程的成绩和学分
20         IF state = 0 THEN
21             CASE
22                 WHEN grade>=95 THEN SET t=4.3;
23                 WHEN grade>=90 AND grade<95 THEN SET t=4.0;
24                 WHEN grade>=85 AND grade<90 THEN SET t=3.7;
25                 WHEN grade>=82 AND grade<85 THEN SET t=3.3;
26                 ELSE SET t=3;
27             END CASE;
28             SET total_g=total_g + t*cred; -- 计算总的学分*绩点
29             SET total_c=total_c + cred; -- 计算总的学分
30         END IF;
31     UNTIL state = 1
32     END REPEAT;
33     CLOSE ct;
34     SET gpa=total_g/total_c;
35     RETURN gpa;
36 END //
37 Delimiter;

```

$$GPA = \frac{\sum \text{课程学分} * \text{课程学分绩点}}{\sum \text{课程学分}}$$

score IS NOT NULL;

```

1  SELECT
2      sno,
3      sname,
4      fun(sno) AS GPA
5  FROM
6      student;

```

信息	结果 1	剖析	状态
	sno	sname	GPA
▶	s1	a	3.57143
	s2	b	3
	s3	c	3
	s4	d	(Null)
	s5	c	3.28571



第八章 数据库编程

112

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~~

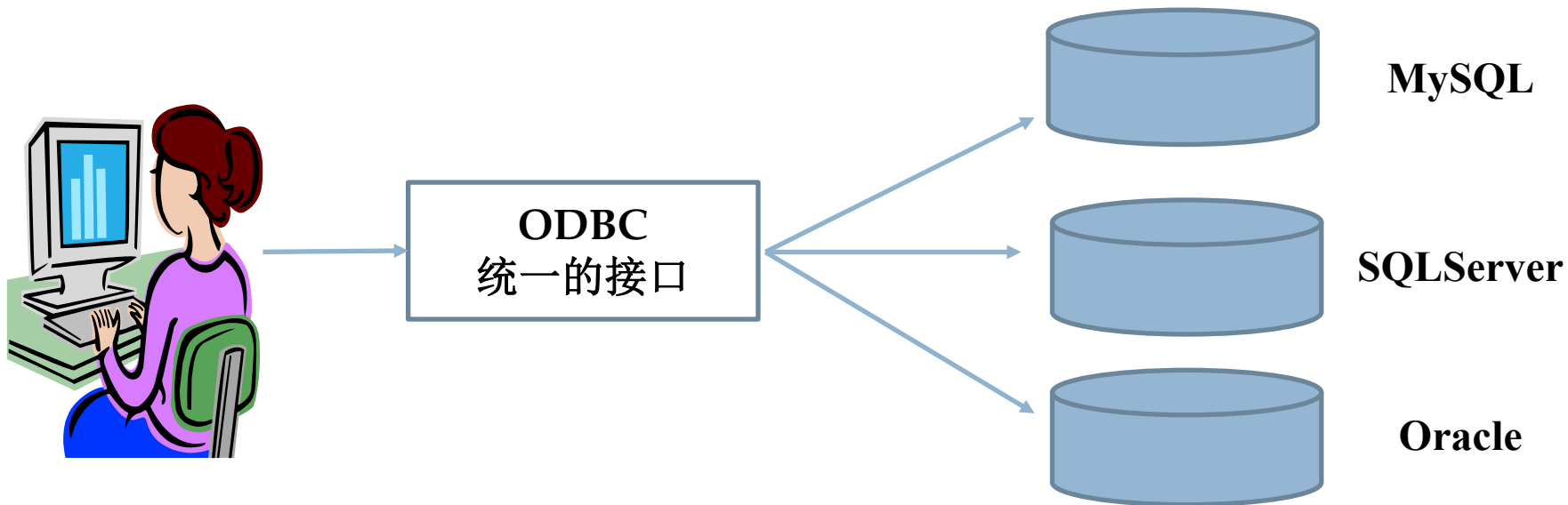
~~*8.6 JDBC编程~~

8.7 小结



8.4 ODBC编程

- 突破SQL语言的局限性
 - 嵌入式SQL：利用高级语言的表达能力
 - 过程化SQL：扩展SQL语言的过程控制能力
 - ODBC：更大的视野，将数据库看做是一类数据源





8.3 ODBC编程

114

- **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



8.3 ODBC编程

115

8.3.1 数据库互连概述

8.3.2 ODBC工作原理概述

8.3.3 ODBC API 基础

8.3.4 ODBC的工作流程

8.3.5 小结



8.3.1 数据库互连概述

116

- **ODBC产生的原因:**
 - **开放数据库连接 (Open Database Connectivity, ODBC)**
 - **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**



数据库互连概述（续）

117

□ 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
- 它建立了一组规范（接口标准），提供了一组访问数据库的标准API
 - 一个基于ODBC的应用程序对数据库的操作不依赖任何DBMS，不直接与DBMS打交道，所有的数据库操作由对应的DBMS的ODBC驱动程序完成

□ ODBC约束力:

- 规范应用开发
- 规范RDBMS应用接口



8.3 ODBC编程

118

8.3.1 数据库互连概述

8.3.2 ODBC工作原理概述

8.3.3 ODBC API 基础

8.3.4 ODBC的工作流程

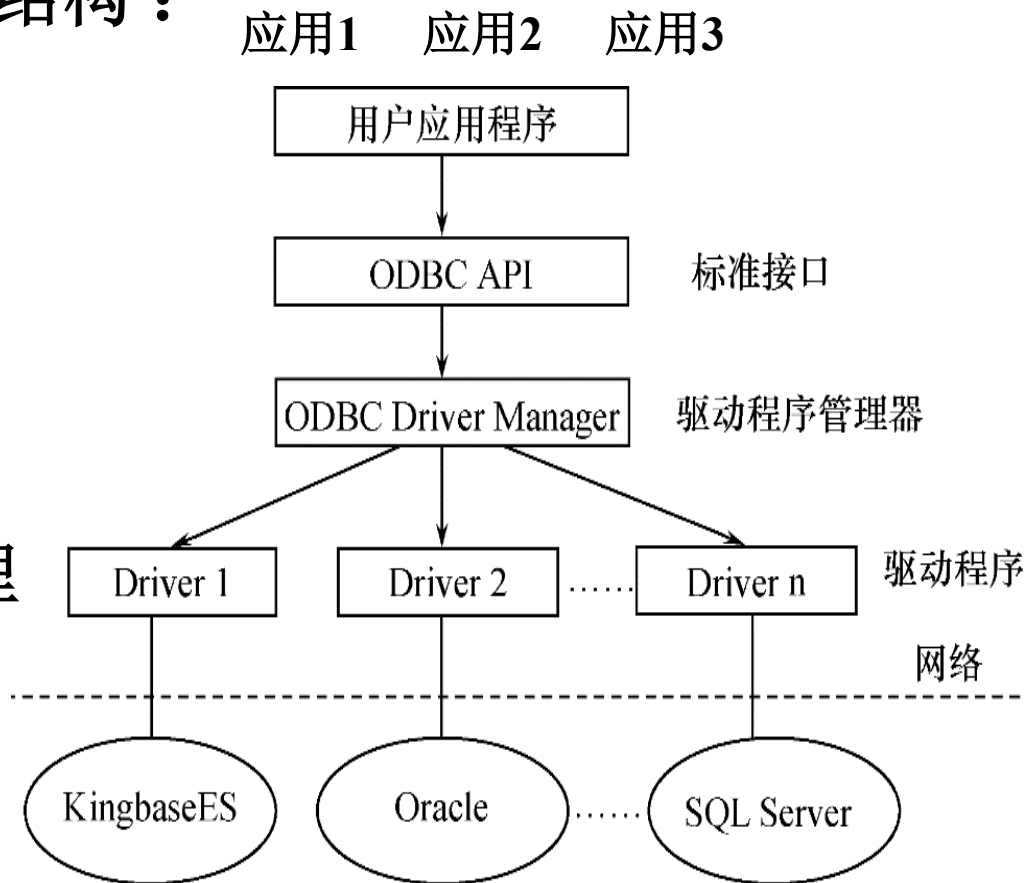
8.3.5 小结



8.3.2 ODBC工作原理概述

□ ODBC应用系统的体系结构：

- 一、 用户应用程序
- 二、 驱动程序管理器
- 三、 数据库驱动程序
- 四、 ODBC数据源管理





一、应用程序

120

- **ODBC应用程序包括的内容：**
 - 请求连接数据库
 - 断开与数据源的连接
 - 操作数据库，获取数据库执行状态
 - 向数据源发送SQL语句
 - 为SQL语句执行结果分配存储空间，定义所读取的数据格式
 - 获取数据库操作结果，或处理错误
 - 进行数据处理并向用户提交处理结果
 - 请求事务的提交和回滚操作



二、驱动程序管理器

121

- **驱动程序管理器：**用来**管理各种驱动程序**
 - 包含在**ODBC32.DLL**中
 - 管理应用程序和驱动程序之间的通信
 - 建立、配置或删除数据源并查看系统当前所安装的数据库

ODBC驱动程序

- **主要功能：**
 - 装载**ODBC驱动程序**
 - 选择和连接正确的驱动程序
 - 管理数据源
 - 检查**ODBC调用参数**的合法性
 - 记录**ODBC函数**的调用等



三、数据库驱动程序

122

- ODBC通过**驱动程序**来提供应用系统与数据库平台的独立性
- ODBC应用程序不能直接存取数据库
 - 各种操作请求由驱动程序管理器提交给某个RDBMS的ODBC驱动程序
 - 通过调用驱动程序所支持的函数来存取数据库
 - 数据库的操作结果也通过驱动程序返回给应用程序
 - 如果应用程序要操纵不同的数据库，就要动态地链接到不同的驱动程序上。



数据库驱动程序（续）

123

□ ODBC驱动程序类型：

□ 单束

- 数据源和应用程序在同一台机器上
- 驱动程序直接完成对数据文件的I/O操作
- 驱动程序相当于数据管理器

□ 多束

- 支持客户机/服务器、客户机/应用服务器/数据库服务器等网络环境下的数据访问
- 由驱动程序完成数据库访问请求的提交和结果集接收
- 应用程序使用驱动程序提供的结果集管理接口操纵执行后的结果数据



四、ODBC数据源

124

- **数据源**：是最终用户需要访问的数据，包含了数据库位置和数据库类型等信息，是一种数据连接的抽象
- 数据源对最终用户是透明的
 - ODBC给每个被访问的数据源指定唯一的数据源名（**Data Source Name**，简称**DSN**），并映射到所有必要的、用来存取数据的低层软件
 - 在连接中，用**数据源名**来代表**用户名、服务器名、所连接的数据库名**等
 - 最终用户无需知道**DBMS**或其他数据管理软件、网络以及有关**ODBC**驱动程序细节



ODBC数据源管理（续）

125

例如，假设某个学校在MS 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
- 为方便与两个数据库连接，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



8.3 ODBC编程

126

8.3.1 数据库互连概述

8.3.2 ODBC工作原理概述

8.3.3 ODBC API 基础

8.3.4 ODBC的工作流程

8.3.5 小结



8.3.3 ODBC API 基础

127

□ ODBC 应用程序接口的一致性

□ API一致性

- API一致性级别有核心级、扩展1级、扩展2级

□ 语法一致性

- 语法一致性级别有最低限度SQL语法级、核心SQL语法级、扩展SQL语法级



ODBC API 基础（续）

128

- 一、函数概述
- 二、句柄及其属性
- 三、数据类型



一、函数概述

129

- **ODBC 3.0 标准提供了76个函数接口：**
 - 分配和释放环境句柄、连接句柄、语句句柄；
 - 连接函数（SQLDriverconnect等）；
 - 与信息相关的函数（如获取描述信息函数SQLGetinfo、SQLGetFuction）；
 - 事务处理函数（如SQLEndTran）；
 - 执行相关函数（SQLExecdirect、SQLExecute等）；
 - 编目函数，ODBC 3.0提供了11个编目函数如SQLTables、SQLColumn等，应用程序可以通过对编目函数的调用来获取数据字典的信息如权限、表结构等



二、句柄及其属性

130

- 句柄是32位整数值，代表一个指针
- ODBC 3.0中句柄分类：
 - 环境句柄
 - 连接句柄
 - 语句句柄
 - 描述符句柄



句柄及其属性（续）

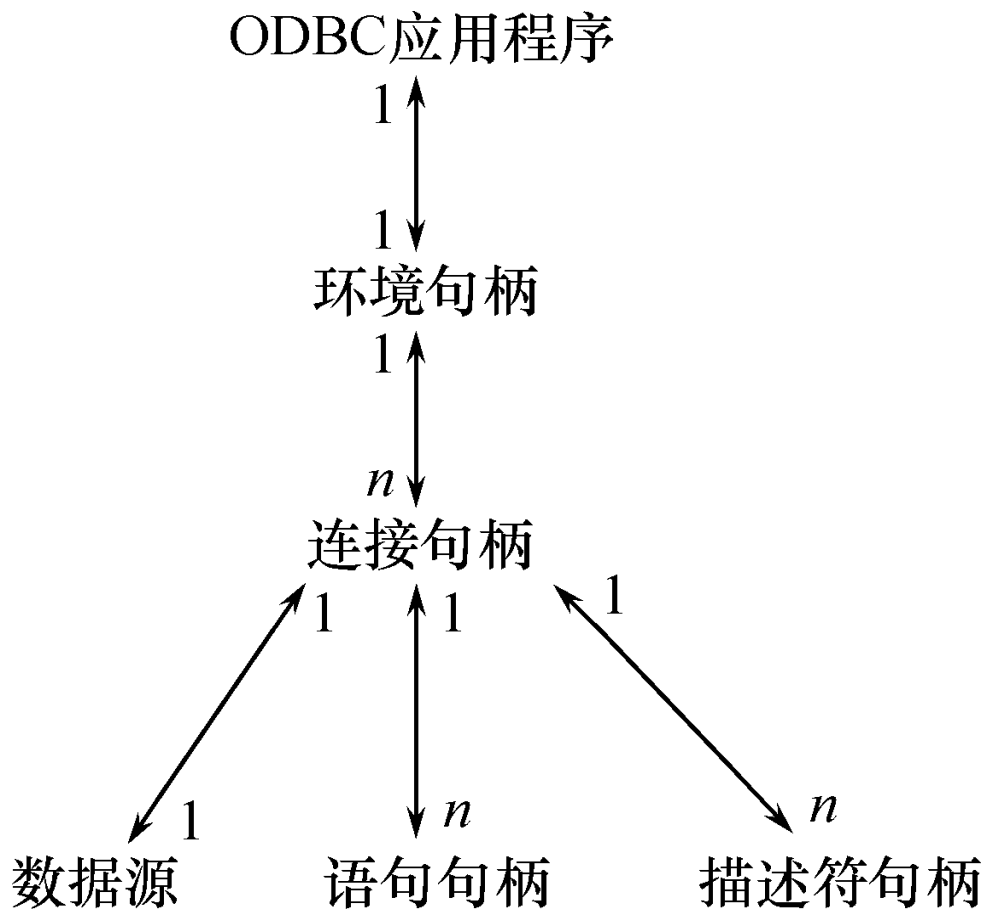
131

1. 每个ODBC应用程序需要建立一个ODBC环境，分配一个环境句柄，存取数据的全局性背景如环境状态、当前环境状态诊断、当前在环境上分配的连接句柄等；
2. 一个环境句柄可以建立多个连接句柄，每一个连接句柄实现与一个数据源之间的连接；
3. 在一个连接中可以建立多个语句句柄，它不只是一个SQL语句，还包括SQL语句产生的结果集以及相关的信息等；
4. 在ODBC 3.0中又提出了描述符句柄的概念，它是描述SQL语句的参数、结果集列的元数据集。



句柄及其属性 (续)

应用程序句柄之间的关系



应用程序句柄之间的关系



8.3 ODBC编程

136

8.3.1 数据库互连概述

8.3.2 ODBC工作原理概述

8.3.3 ODBC API 基础

8.3.4 ODBC的工作流程

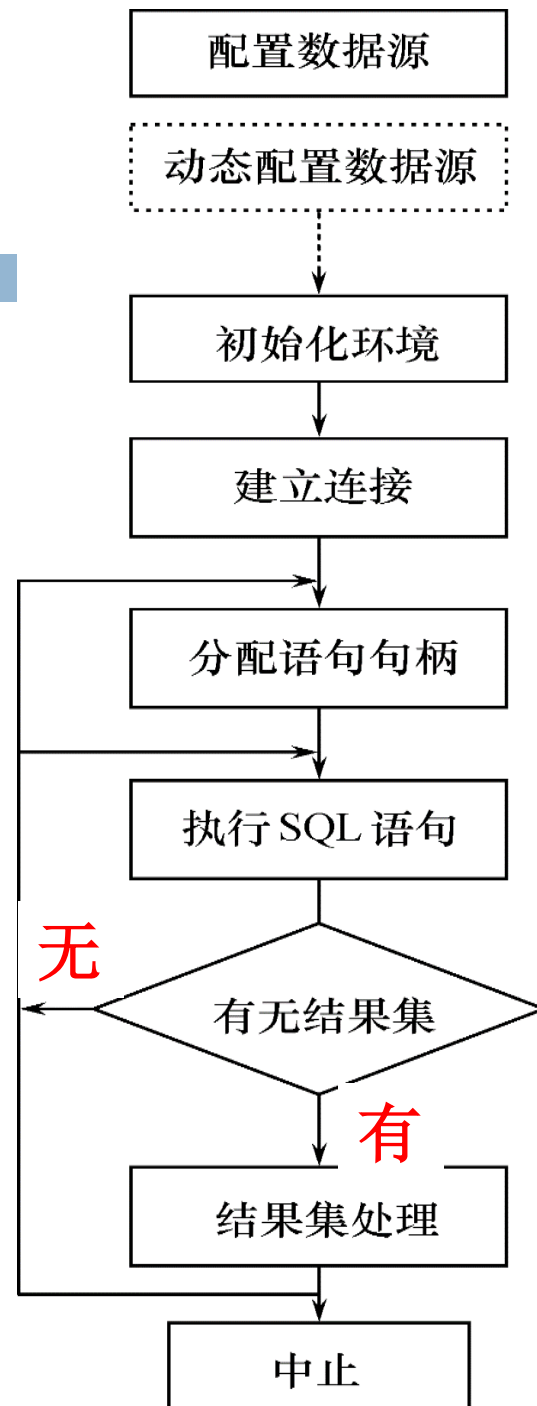
8.3.5 小结



8.3.4 ODBC的工作流程

137

□ ODBC的工作流程:





ODBC的工作流程（续）

138

[例8.12] 将KingbaseES数据库中Student表的数据备份到SQL SERVER数据库中。

- 该应用涉及两个不同的RDBMS中的数据源
- 使用ODBC来开发应用程序，只要改变应用程序中连接函数（SQLConnect）的参数，就可以连接不同RDBMS的驱动程序，连接两个数据源



ODBC的工作流程（续）

139

- 在应用程序运行前，已经在KingbaseES和SQL SERVER中分别建立了STUDENT关系表

CREATE TABLE Student

(Sno CHAR (9) PRIMARY KEY,

Sname CHAR (20) UNQUE

Ssex CHAR (2) ,

Sage SMALLINT,

Sdept CHAR (20)

) ;



ODBC的工作流程（续）

140

- 应用程序要执行的操作是：
 - 在KingbaseES上执行**SELECT * FROM STUDENT;**
 - 把获取的结果集，通过多次执行
INSERT INTO STUDENT (Sno, Sname, Ssex, Sage, Sdept) VALUES (?, ?, ?, ?, ?);
 - 插入到SQL SERVER的**STUDENT**表中



ODBC的工作流程（续）

141

□ 操作步骤：

- 一、配置数据源
- 二、初始化环境
- 三、建立连接
- 四、分配语句句柄
- 五、执行SQL语句
- 六、结果集处理
- 七、中止处理



一、配置数据源

142

- 配置数据源两种方法：

- (1)运行数据源管理工具来进行配置；

- (2)使用**Driver Manager** 提供的**ConfigDsn**函数来增加、修改或删除数据源

- 在 [例8.13] 中，采用了第一种方法创建数据源。因为要同时用到**KingbaseES**和**SQL Server**，所以分别建立两个数据源，将其取名为**KingbaseES ODBC**和**SQLServer**。



配置数据源（续）

143

[例8.13] 创建数据源的详细过程

```
#include <stdlib.h>  
#include <stdio.h>  
#include <windows.h>  
#include <sql.h>  
#include <sqlext.h>  
#include <Sqltypes.h>  
#define SNO_LEN 30  
#define NAME_LEN 50  
#define DEPART_LEN 100  
#define SSEX_LEN 5
```



配置数据源（续）

144

[例13] 创建数据源---第一步：定义句柄和变量

```
int main()
{
    /* Step 1 定义句柄和变量 */
    //以king开头的表示的是连接KingbaseES的变量
    //以server开头的表示的是连接SQLSERVER的变量
    SQLHENV    kinghenv, serverhenv;           //环境句柄， 2个
    SQLHDBC    kinghdbc, serverhdbc;         //连接句柄， 2个
    SQLHSTMT   kinghstmt, serverhstmt;       //语句句柄， 2个
    SQLRETURN  ret;
    SQLCHAR    sName [NAME_LEN] , sDepart [DEPART_LEN] ,
    sSex [SSEX_LEN] , sSno [SNO_LEN] ;
    SQLINTEGER sAge;
    SQLINTEGER cbAge=0, cbSno=SQL_NTS, cbSex=SQL_NTS,
    cbName=SQL_NTS, cbDepart=SQL_NTS;
}
```



二、初始化环境

145

- 没有和具体的驱动程序相关联，由**Driver Manager**来进行控制，并配置环境属性
- 应用程序通过调用连接函数和某个数据源进行连接后，**Driver Manager**才调用所连的驱动程序中的**SQLAllocHandle**，来真正分配环境句柄的数据结构



初始化环境代码

146

[例13] 创建数据源---第二步：初始化环境

/* Step 2 初始化环境，创建环境句柄 */

```
ret=SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE,  
&kinghenv);
```

```
ret=SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE,  
&serverhenv);
```

/* Step 2 初始化环境，初始化环境句柄变量*/

```
ret=SQLSetEnvAttr (kinghenv, SQL_ATTR_ODBC_VERSION,  
(void*)SQL_OV_ODBC3, 0);
```

```
ret=SQLSetEnvAttr (serverhenv, SQL_ATTR_ODBC_VERSION,  
(void*)SQL_OV_ODBC3, 0);
```




三、建立连接

147

- 应用程序调用**SQLAllocHandle**分配连接句柄，通过**SQLConnect**、**SQLDriverConnect**或**SQLBrowseConnect**与数据源连接
- **SQLConnect**连接函数，输入参数为：
 - 配置好的数据源名称
 - 用户ID
 - 口令
- [例13]中**KingbaseES ODBC**为数据源名字，**SYSTEM**为用户名，**MANAGER**为用户密码



建立连接代码

148

[例13] 创建数据源---第三步：建立连接

/* Step 3 :建立连接，创建链接句柄*/

```
ret=SQLAllocHandle(SQL_HANDLE_DBC, kinghenv, &kinghdbc);
```

```
ret=SQLAllocHandle(SQL_HANDLE_DBC, serverhenv, &serverhdbc);
```

/* 建立连接，初始化连接句柄变量：链接数据源*/

```
ret=SQLConnect (kinghdbc, "KingbaseES ODBC", SQL_NTS,  
"SYSTEM", SQL_NTS, "MANAGER", SQL_NTS);
```

```
if (!SQL_SUCCEEDED(ret))//连接失败时返回错误值
```

```
return-1;
```

```
ret=SQLConnect (serverhdbc, "SQLServer", SQL_NTS, "sa",  
SQL_NTS, "sa", SQL_NTS);
```

```
if (!SQL_SUCCEEDED(ret) ) //连接失败时返回错误值
```

```
return -1;
```



四、分配语句句柄

149

- 处理任何SQL语句之前，应用程序还需要首先分配一个语句句柄
- 语句句柄含有具体的SQL语句以及输出的结果集等信息
- [例8.13]中分配了两个语句句柄：
 - 一个用来从KingbaseES中读取数据产生结果集（kinghstmt）
 - 一个用来向SQLSERVER插入数据（serverhstmt）
- 应用程序还可以通过SQLtStmtAttr来设置语句属性（也可以使用默认值）
- [例13]中结果集绑定的方式为按列绑定



分配语句句柄代码

150

[例8.13] 创建数据源---第四步

/ Step 4 :初始化语句句柄， 创建语句句柄 */*

```
ret=SQLAllocHandle (SQL_HANDLE_STMT, kinghdbc, &kinghstmt);
```

```
ret=SQLSetStmtAttr (kinghstmt, SQL_ATTR_ROW_BIND_TYPE,  
(SQLPOINTER) // 初始化语句句柄， 设置语句句柄的属性
```

```
SQL_BIND_BY_COLUMN, SQL_IS_INTEGER );
```

```
ret=SQLAllocHandle(SQL_HANDLE_STMT, serverhdbc, &serverhstmt);
```

[例8.12]中结果集绑定的方式为按列绑定

[例8.12]中分配了两个语句句柄

- 一个用来从KingbaseES中读取数据产生结果集（kinghstmt）
- 一个用来向SQL Server插入数据（serverhstmt）



五、执行SQL语句

151

- 应用程序处理SQL语句的两种方式：
 - 预处理（SQLPrepare、SQLExecute适用于语句的多次执行）
 - 直接执行（SQLExecdirect）
- 如果SQL语句含有参数，应用程序为每个参数调用**SQLBindParameter**，并把它们绑定至应用程序变量
- 应用程序可以直接通过改变应用程序缓冲区的内容从而在程序中动态的改变SQL语句的具体执行
- 应用程序根据语句的类型进行的处理
 - 有结果集的语句（select或是编目函数），则进行结果集处理。
 - 没有结果集的函数，可以直接利用本语句句柄继续执行新的语句或是获取行计数（本次执行所影响的行数）之后继续执行。



六、结果集处理

152

- 应用程序可以通过**SQLNumResultCols**来获取结果集中的列数
- 通过**SQLDescribeCol**或是**SQLColAttribute**函数来获取结果集每一列的名称、数据类型、精度和范围



结果集处理（续）

153

- ODBC中使用游标来处理结果集数据
- ODBC中游标类型：
 - forward-only游标，是ODBC的默认游标类型
 - 可滚动（scroll）游标：
 - 静态（static）
 - 动态（dynamic）
 - 码集驱动（keyset-driven）
 - 混合型（mixed）



结果集处理（续）

154

- 结果集处理步骤：
 - ODBC游标打开方式不同于嵌入式SQL，不是显式声明而是系统自动产生一个游标（Cursor），当结果集刚刚生成时，游标指向第一行数据之前
 - 应用程序通过SQLBindCol，把查询结果绑定到应用程序缓冲区中，通过SQLFetch或是SQLFetchScroll来移动游标获取结果集中的每一行数据
 - 对于如图像这类特别的数据类型当一个缓冲区不足以容纳所有的数据时，可以通过SQLGetData分多次获取
 - 最后通过SQLClosecursor来关闭游标



结果集处理（续）

155

- 在[例8.13]中，
 - 使用SQLExecdirect获取KingbaseES中的结果集，并将结果集根据各列不同的数据类型绑定到用户程序缓冲区
 - 向SQL Server插入数据时，采用了预编译的方式，首先通过SQLPrepare来预处理SQL语句，将每一列绑定到用户缓冲区
 - 应用程序可以直接修改结果集缓冲区的内容



程序源码

156

[例8.13] 创建数据源---第五步：执行SQL语句

/* Step 5 :两种方式执行语句 */

/* 执行SQL语句，预编译带有参数的语句，对于SQLServer源*/

```
ret=SQLPrepare(serverhstmt, "INSERT INTO STUDENT(SNO, SNAME, SSEX,  
SAGE, SDEPT) VALUES (?, ?, ?, ?, ?)", SQL_NTS);
```

```
if (ret==SQL_SUCCESS || ret==SQL_SUCCESS_WITH_INFO)
```

```
{
```

```
ret=SQLBindParameter(serverhstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,  
SQL_CHAR, SNO_LEN, 0, sSno, 0, &cbSno);
```

```
ret=SQLBindParameter(serverhstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,  
SQL_CHAR, NAME_LEN, 0, sName, 0, &cbName);
```

```
ret=SQLBindParameter(serverhstmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR,  
SQL_CHAR, 2, 0, sSex, 0, &cbSex);
```

```
ret=SQLBindParameter(serverhstmt, 4, SQL_PARAM_INPUT,  
SQL_C_LONG, SQL_INTEGER, 0, 0, &sAge, 0, &cbAge);
```



程序源码

157

```
ret=SQLBindParameter(serverhstmt, 5, SQL_PARAM_INPUT, SQL_C_CHAR,  
SQL_CHAR, DEPART_LEN, 0, sDepart, 0, &cbDepart);  
}
```

*/*执行SQL语句，直接执行，对于Kingbase源*/*

```
ret=SQLExecDirect(kinghstmt, "SELECT * FROM STUDENT", SQL_NTS);
```

```
if (ret==SQL_SUCCESS || ret==SQL_SUCCESS_WITH_INFO)
```

```
{
```

//SQLBindCol把查询结果绑定到应用缓冲区中

```
ret=SQLBindCol(kinghstmt, 1, SQL_C_CHAR, sSno, SNO_LEN, &cbSno);
```

```
ret=SQLBindCol(kinghstmt, 2, SQL_C_CHAR, sName, NAME_LEN,  
&cbName);
```

```
ret=SQLBindCol(kinghstmt, 3, SQL_C_CHAR, sSex, SSEX_LEN,  
&cbSex);
```

```
ret=SQLBindCol(kinghstmt, 4, SQL_C_LONG, sAge, 0, &cbAge);
```

```
ret=SQLBindCol(kinghstmt, 5, SQL_C_CHAR, sDepart, DEPART_LEN,  
&cbDepart);
```

```
}
```



程序源代码

158

[例8.13] 创建数据源---第六步：结果集处理

```
/* Step 6 : 用游标处理结果集并执行预编译后的语句*/  
while ( (ret=SQLFetch(kinghstmt) ) !=SQL_NO_DATA_FOUND)  
{  
    if (ret==SQL_ERROR) printf("Fetch error \n");  
    else  
        // 执行SQL, 把缓冲区数据放入SQLServer  
        ret=SQLExecute(serverhstmt);  
}
```



七、中止处理

159

- 应用程序中止步骤：
 - 首先释放语句句柄
 - 释放数据库连接
 - 与数据库服务器断开
 - 释放ODBC环境



中止处理代码

160

[例8.13] 创建数据源---第七步：中止处理

```
/* Step 7 中止处理*/
SQLFreeHandle(SQL_HANDLE_STMT, kinghstmt); //释放语句句柄
SQLDisconnect(kinghdbc); //断开连接
SQLFreeHandle(SQL_HANDLE_DBC, kinghdbc); //释放数据库连接
SQLFreeHandle(SQL_HANDLE_ENV, kinghenv); //释放ODBC环境
SQLFreeHandle(SQL_HANDLE_STMT, serverhstmt);
SQLDisconnect(serverhdbc);
SQLFreeHandle(SQL_HANDLE_DBC, serverhdbc);
    SQLFreeHandle(SQL_HANDLE_ENV, serverhenv);
return 0;
}
```



Python ODBC

161

- Python标准数据库接口为Python DB-API，其支持非常多的数据库，不同的数据库需要下载不同的DB API模块
- DB-API是一个规范，定义了一系列必须的对象和数据存取方式，以便为各种各样的底层数据库系统和多种多样的数据库接口程序提供一致的访问接口，是一种**ODBC编程**
- Python连接数据库的流程：
 - 引入DB-API模块（如MySQL、Oracle）
 - 获取与数据库的连接
 - 执行SQL语句和存储过程
 - 关闭数据库连接



Python ODBC例子

162

import pyodbc #引入相应模块

Specifying the ODBC driver, server name, database, etc. directly, 连接SQL Server, 修改DRIVER的赋值, 可以连接不同的数据库

```
cnxn = pyodbc.connect('DRIVER={SQL  
Server};SERVER=localhost;DATABASE=testdb;UID=me;PWD=pass'  
)
```

Using a DSN, but providing a password as well

```
cnxn = pyodbc.connect('DSN=test;PWD=password')
```

Create a cursor from the connection

```
cursor = cnxn.cursor()
```




Python连接MySQL例子（续）

163

```
cursor.execute("select user_id, user_name from users")
row = cursor.fetchone()
print('name:', row[1])          # access by column index (zero-based)
print('name:', row.user_name)  # access by name
```

```
cursor.execute("""
select user_id, user_name
from users
where last_logon < ?
and bill_overdue = ?
""", datetime.date(2001, 1, 1), 'y') #支持参数传入
```



Python连接MySQL例子（续）

164

□ PyMySQL

- 在 Python3.x 版本中用于连接 MySQL服务器

□ 使用

- 导入pymysql模块: `import pymysql`
- 连接database: `conn = pymysql.connect(...)`
- 得到一个可以执行SQL语句的句柄:
 - `cursor = conn.cursor()`
- 定义要执行的SQL语句: `sql = ...`
- 执行SQL: `cursor.execute(sql)`
- 关闭句柄: `cursor.close()`
- 关闭数据库连接: `conn.close()`



8.3 ODBC编程

165

8.3.1 数据库互连概述

8.3.2 ODBC工作原理概述

8.3.3 ODBC API 基础

8.3.4 ODBC的工作流程

8.3.5 小结



8.3.5 小结

166

- 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