1.JDBC练习题

在MySQL中创建一张学生表， 

**用JDBC对该表进行基本操作:**

1.插入如下数据

('s001','老大',20,'计算机学院')

('s002','老二',19,'计算机学院')

('s003','老三',18,'计算机学院')

('s004','老四',17,'计算机学院')

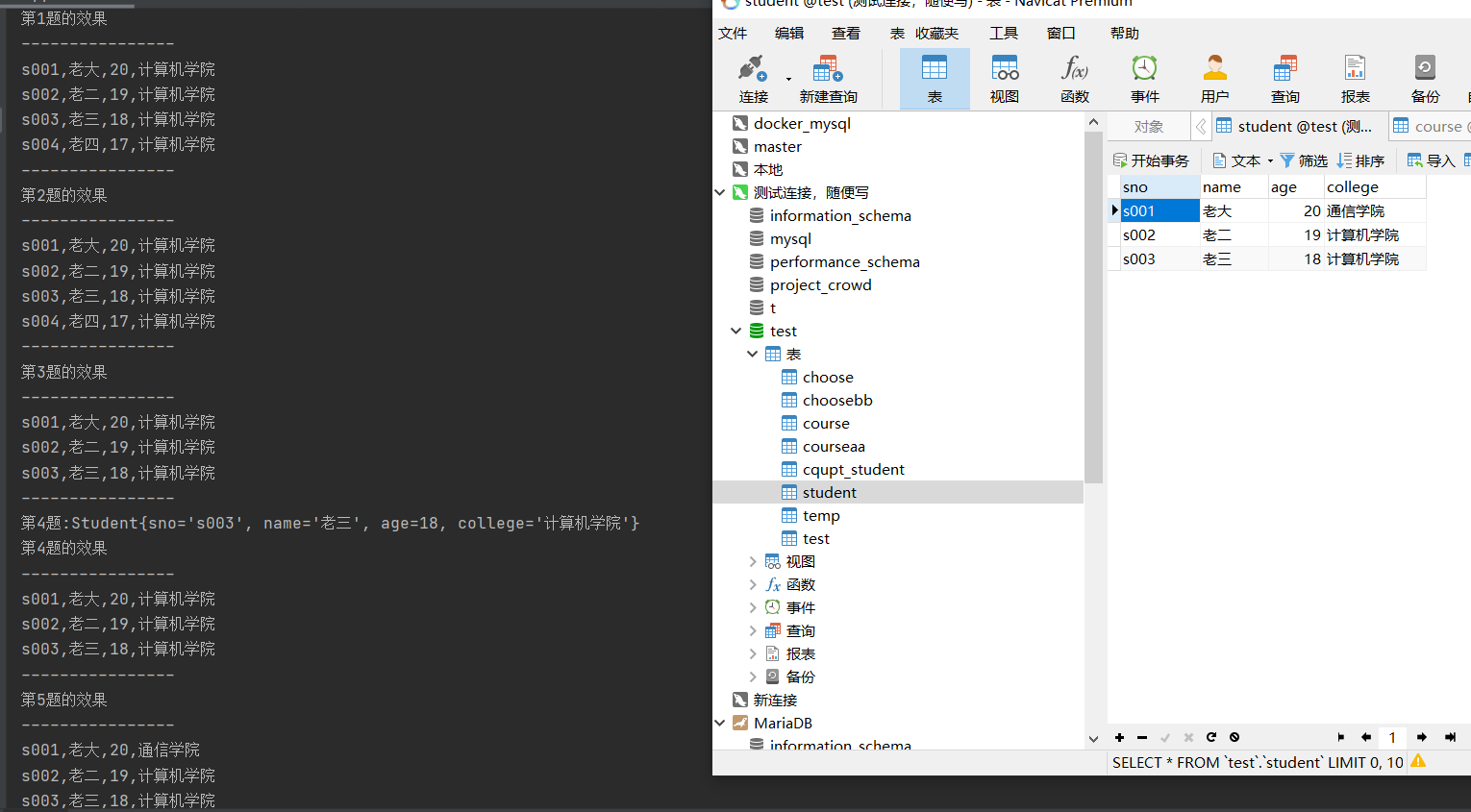
2.查看表中的所有信息，遍历输出到控制台

3.把sno为s004的记录删除

4.查询sno为s003的记录

5.把sno为s001的记录修改为('s001','老大',20,'通信学院')

效果

 CREATE TABLE students (

sno VARCHAR(10) PRIMARY KEY,

sname VARCHAR(20),

age INT,

department VARCHAR(30)

);

import java.sql.\*;

public class StudentDatabase {

public static void main(String[] args) {

Connection conn = null;

Statement stmt = null;

try {

// 1.连接数据库

conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydatabase", "root", "password");

// 2.创建Statement对象

stmt = conn.createStatement();

// 3.插入数据

String sql = "INSERT INTO students VALUES " +

"('s001','老大',20,'计算机学院')," +

"('s002','老二',19,'计算机学院')," +

"('s003','老三',18,'计算机学院')," +

"('s004','老四',17,'计算机学院')";

int rowCount = stmt.executeUpdate(sql);

System.out.println(rowCount + " rows inserted.");

// 4.查询数据

sql = "SELECT \* FROM students";

ResultSet rs = stmt.executeQuery(sql);

// 5.遍历输出结果

while (rs.next()) {

String sno = rs.getString("sno");

String sname = rs.getString("sname");

int age = rs.getInt("age");

String department = rs.getString("department");

System.out.println(sno + "\t" + sname + "\t" + age + "\t" + department);

}

// 6.删除记录

sql = "DELETE FROM students WHERE sno='s004'";

rowCount = stmt.executeUpdate(sql);

System.out.println(rowCount + " rows deleted.");

// 7.查询记录

sql = "SELECT \* FROM students WHERE sno='s003'";

rs = stmt.executeQuery(sql);

if (rs.next()) {

String sno = rs.getString("sno");

String sname = rs.getString("sname");

int age = rs.getInt("age");

String department = rs.getString("department");

System.out.println(sno + "\t" + sname + "\t" + age + "\t" + department);

}

// 8.修改记录

sql = "UPDATE students SET department='通信学院' WHERE sno='s001'";

rowCount = stmt.executeUpdate(sql);

System.out.println(rowCount + " rows updated.");

} catch (SQLException e) {

e.printStackTrace();

} finally {

// 9.关闭资源

try {

if (stmt != null) stmt.close();

if (conn != null) conn.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

}

2.java练习题

用java语言实现冒泡排序、插入排序、快速排序、合并排序、堆排序

//冒泡排序

public class BubbleSort {

public void bubbleSort(int[] array) {

int n = array.length;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (array[j] > array[j + 1]) {

int temp = array[j];

array[j] = array[j + 1];

array[j + 1] = temp;

}

}

}

}

}

//插入排序

public class InsertionSort {

public void insertionSort(int[] array) {

int n = array.length;

for (int i = 1; i < n; ++i) {

int key = array[i];

int j = i - 1;

while (j >= 0 && array[j] > key) {

array[j + 1] = array[j];

j = j - 1;

}

array[j + 1] = key;

}

}

}

//快速排序

public class QuickSort {

public void quickSort(int[] array, int low, int high) {

if (low < high) {

int pi = partition(array, low, high);

quickSort(array, low, pi - 1);

quickSort(array, pi + 1, high);

}

}

private int partition(int[] array, int low, int high) {

int pivot = array[high];

int i = (low - 1);

for (int j = low; j < high; j++) {

if (array[j] <= pivot) {

i++;

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

}

int temp = array[i + 1];

array[i + 1] = array[high];

array[high] = temp;

return i + 1;

}

}

//合并排序

public class MergeSort {

public void mergeSort(int[] array, int left, int right) {

if (left < right) {

int middle = (left + right) / 2;

mergeSort(array, left, middle);

mergeSort(array, middle + 1, right);

merge(array, left, middle, right);

}

}

private void merge(int[] array, int left, int middle, int right) {

int n1 = middle - left + 1;

int n2 = right - middle;

int[] L = new int[n1];

int[] R = new int[n2];

for (int i = 0; i < n1; ++i)

L[i] = array[left + i];

for (int j = 0; j < n2; ++j)

R[j] = array[middle + 1 + j];

int i = 0, j = 0;

int k = left;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

array[k] = L[i];

i++;

} else {

array[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

array[k] = L[i];

i++;

k++;

}

while (j < n2) {

array[k] = R[j];

j++;

k++;

}

}

}

//堆排序

public class HeapSort {

public void heapSort(int[] array) {

int n = array.length;

// Build max heap

for (int i = n / 2 - 1; i >= 0; i--)

heapify(array, n, i);

// Extract elements from heap one by one

for (int i = n - 1; i > 0; i--) {

int temp = array[0];

array[0] = array[i];

array[i] = temp;

heapify(array, i, 0);

}

}

private void heapify(int[] array, int n, int i) {

int largest = i;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < n && array[l] > array[largest])

largest = l;

if (r < n && array[r] > array[largest])

largest = r;

if (largest != i) {

int temp = array[i];

array[i] = array[largest];

array[largest] = temp;

heapify(array, n, largest);

}

}

}