《数据库概论》实验二 高级SQL 实验报告

姓名 191220066 学号 17767065642 联系方式 191220066@smail.nju.edu.cn

实验环境

操作系统：Win10

MySQL：8.0.23

JDK 1.8，Junit4.12

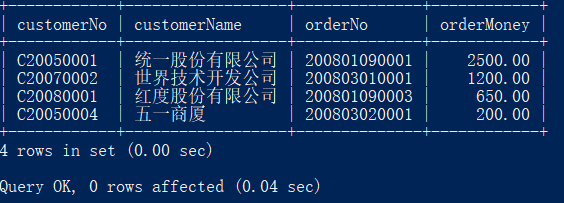
实验过程

高级语言使用java,项目地址https://github.com/Jinglong-Liu/jdbc-demo

1. 声明并调用存储过程，完成以下任务：
2. 存储过程功能：根据输入的商品名称，找到订购了该商品的客户编号、 客户名称、订单编号、订货数量和订货金额，并按订货金额降序输出；

调用该存储过程查询订购了“32M DRAM”商品的客户编号、客户名称、 订单编号、订货数量和订货金额，并按订货金额降序输出。

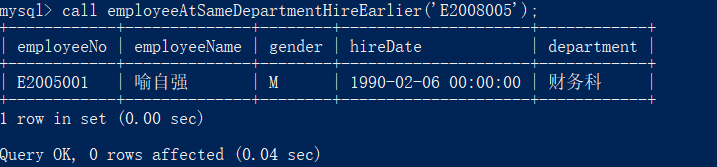
1. **drop** **procedure** if exists productInfo;
2. delimiter $$
3. **create** **procedure** productInfo(IN pName **varchar**(40))
4. **BEGIN**
5. **select** o.customerNo,c.customerName,o.orderNo,od.quantity \* od.price orderMoney
6. **from** OrderMaster o,Customer c,OrderDetail od,Product p
7. **where** pName = p.productName and od.productNo = p.productNo
8. and od.orderNo = o.orderNo and o.customerNo = c.customerNo
9. **order** **by** orderMoney **desc**;
10. **END**$$
11. delimiter ;
13. call productInfo('32M DRAM');



1. 存储过程功能：根据输入的员工编号，查询比该员工雇佣日期早的同一 部门的员工编号、姓名、性别、雇佣日期、所属部门；

调用该存储过程查询比员工编号“E2008005”雇佣日期早的同一部门的 员工编号、 姓名、性别、雇佣日期、所属部门。

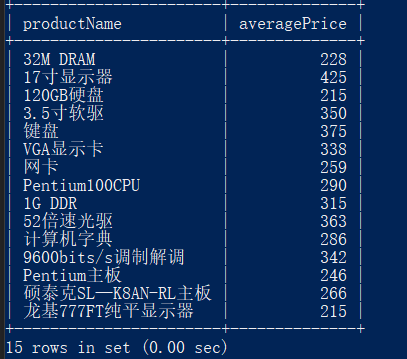
1. **drop** **procedure** if exists employeeAtSameDepartmentHireEarlier;
2. delimiter $$
3. **create** **procedure** employeeAtSameDepartmentHireEarlier (IN eNo **char**(8))
4. **BEGIN**
5. **select** e.employeeNo,e.employeeName,e.gender,e.hireDate,e.department
6. **from** Employee e,Employee e2
7. **where** e2.employeeNo = eNo and e.department = e2.department and e.hireDate < e2.hireDate;
8. **END** $$
9. delimiter ;
10. call employeeAtSameDepartmentHireEarlier('E2008005');



1. 声明并调用存储函数，完成以下任务：
2. 存储函数功能：根据输入的商品名称，返回该商品订购平均价；

调用该存储函数查询商品基本信息表中所有商品名称及其订购平均价。

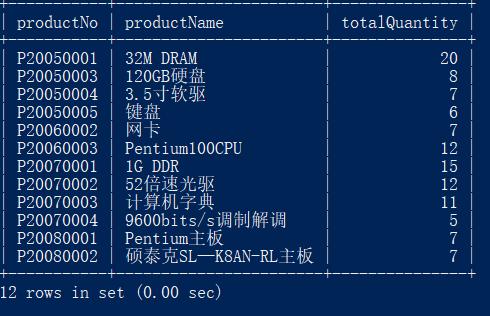
1. **set** **global** log\_bin\_trust\_function\_creators = 1;
2. **drop** **function** if exists averageOrderPrice;
3. delimiter $$
4. **create** **function** averageOrderPrice (pName **varchar**(40))
5. **returns** **integer**
6. **BEGIN**
7. **declare** ave **integer**;
8. **select** sum(od.quantity \* od.price)/sum(od.quantity) **into** ave
9. **from** Product p,OrderDetail od
10. **where** p.productName = pName and od.productNo = p.productNo;
11. **return** ave;
12. **END** $$
13. delimiter ;
14. **select** productName,averageOrderPrice(productName) averagePrice **from** Product p;



1. 存储函数功能：根据输入的商品编号，统计该商品的销售总量；

调用该存储函数查询销售总量大于4的商品编号、商品名称及销售数量。

1. **drop** **function** if exists totalQuantity;
2. delimiter $$
3. **create** **function** totalQuantity (pNo **char**(9))
4. **returns** **integer**
5. **BEGIN**
6. **declare** total **integer**;
7. **select** sum(quantity) **into** total
8. **from** OrderDetail od  **where** pNo = productNo;
9. **return** total;
10. **END** $$
11. delimiter ;
12. --
13. **select** distinct p.productNo,p.productName,totalQuantity(p.productNo) totalQuantity
14. **from** Product p,OrderDetail od
15. **where** p.productNo = od.productNo and totalQuantity(p.productNo) > 4;



1. 创建触发器，完成以下任务：
2. 当插入一项商品时，如果商品价格大于1000，则将其设置为1000。
3. **drop** **trigger** if exists reviseProductPrice;
4. delimiter $$
5. **create** **trigger** reviseProductPrice
6. BEFORE **INSERT** **on** Product **for** each row
7. **BEGIN**
8. if new.productPrice > 1000 **then**
9. **set** new.productPrice = 1000;
10. **end** if;
11. **END** $$

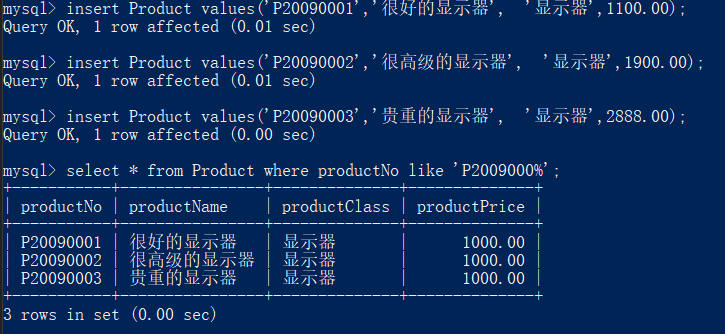
测试代码

insert Product values('P20090001','很好的显示器',  '显示器',1100.00);

insert Product values('P20090002','很高级的显示器',  '显示器',1900.00);

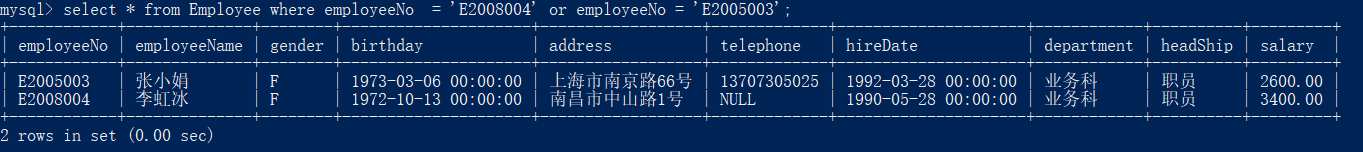
insert Product values('P20090003','贵重的显示器',  '显示器',2888.00);

select \* from Product where productNo like 'P2009000%';



1. 当员工完成一个新的订单时，薪水增加5%；如果该员工是1992年前入职的，则再增加3%。
2. **drop** **trigger** if exists addSalaryAfterFinishOrder;
3. delimiter $$
4. **create** **trigger** addSalaryAfterFinishOrder
5. BEFORE **INSERT** **on** OrderMaster
6. **for** each row
7. **BEGIN**
8. **update** Employee e
9. **set** e.salary = e.salary \* 1.05
10. **where** new.employeeNo = e.employeeNo;
11. **update** Employee e **set** e.salary = e.salary \* 1.03
12. **where** new.employeeNo = e.employeeNo and e.hireDate < "19920101";
13. **END** $$
14. delimiter;

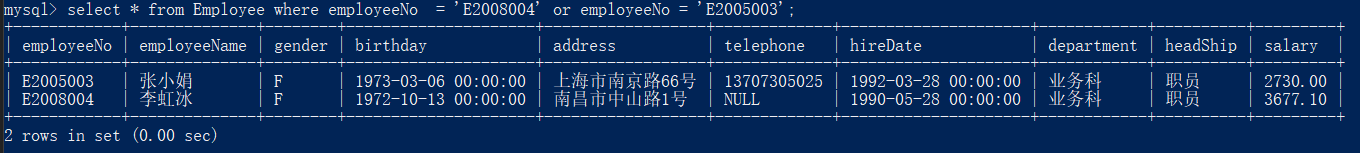
插入数据前



测试代码

1. **insert** OrderMaster **values**('200807010001','C20050001','E2008004','20080701',0.00,'I000000011');
2. **insert** OrderMaster **values**('200807010002','C20050001','E2005003','20080701',0.00,'I000000012');

插入两条数据后



2600 \* 1.05 = 2730；

3400 \* 1.05 \* 1.08 = 3677

1. 使用高级程序设计语言访问SQL并执行如下的查询、插入、删除、更新：

以下部分需要使用高级程序设计语言(这里使用java,jdbc)访问数据库。为此先写一个模板框架，勇于数据库的连接，再抽象出query(String sql)和update(String sql)两个接口，分别执行查询和更新的sql操作。以后只需要生成sql语句字符串，调用方法即可执行sql语句

连接类主要方法如下

1. **import** java.sql.\*;
3. **public** **class** DBConnection {
4. **static** **final** String JDBC\_DRIVER = "com.mysql.jdbc.Driver";
5. //static final String JDBC\_DRIVER = com.mysql.cj.jdbc.Driver;
6. **static** **final** String DB\_URL = "jdbc:mysql://localhost:3306/OrderDB?characterEncoding=utf8&useSSL=false";
7. **private** **static** String userName = "root";
8. **private** **static** String password = "root";
9. **private** **static** String dbName = "OrderDB";
10. **private** **static** Connection connection = **null**;
12. /\*\*
13. \* Create a new Connection
14. \* @return
15. \*/
16. **public** **static** Connection getConnection() {
17. **if**(connection != **null**){
18. **return** connection;
19. }
20. Connection con = **null**;
21. **try** {
22. Class.forName(JDBC\_DRIVER);
23. con = DriverManager.getConnection(DB\_URL, userName, password);
24. } **catch** (SQLException | ClassNotFoundException e) {
25. e.printStackTrace();
26. }
27. **return** con;
28. }
29. }

此外还有释放连接的方法，主要是封装异常处理。篇幅有限且内容比较重复，不再赘述。

Sql执行类SQLExecutor.java

1. **import** java.sql.Connection;
2. **import** java.sql.ResultSet;
3. **import** java.sql.SQLException;
4. **import** java.sql.Statement;
6. **public** **class** SQLExecutor {
7. Connection conn;
8. /\*\*
9. \* Create and a db connection.
10. \*/
11. **public** SQLExecutor(){
12. conn = DBConnection.getConnection();
13. }
14. /\*\*
15. \* execute an sql query command
16. \* @param sql
17. \* @return resultSet
18. \*/
19. **public** ResultSet query(String sql){
20. ResultSet resultSet = **null**;
21. Statement stmt = **null**;
22. **try** {
23. stmt = conn.createStatement();
24. resultSet = stmt.executeQuery(sql);
25. } **catch** (SQLException throwables) {
26. throwables.printStackTrace();
27. }
28. **return** resultSet;
29. }
30. /\*\*
31. \* execute an sql update command
32. \* @param sql
33. \*/
34. **public** **void** update(String sql){
35. Statement stmt = **null**;
36. **try** {
37. stmt = conn.createStatement();
38. stmt.executeUpdate(sql);
39. } **catch** (SQLException throwables) {
40. throwables.printStackTrace();
41. }
42. }
43. }

辅助类

1. **public** **class** DBHelper {
2. **private** SQLExecutor sqlExecutor = **new** SQLExecutor();
3. //methods
4. }

在DBHelper类中调用sqlExecutor.query(String sql)或sqlExecutor.update(String sql)即可完成sql语句的执行。下面一系列具体要求，都在这个类中实现对应的方法。

查询到的结果可以格式化输出。

1. /\*\*
2. \* out put resultSet.
3. \* @param resultSet
4. \* @param out:PrintStream,like "System.out".
5. \*/
6. **public** **void** outputResultSet(ResultSet resultSet, PrintStream out){
7. //用于格式化输出一个resultSet，具体代码略
8. }

使用高级程序设计语言访问SQL并执行如下的查询、插入、删除、更新：

1. 查询职工工资按从高到低排序的前20的职工编号、职工姓名和工资；

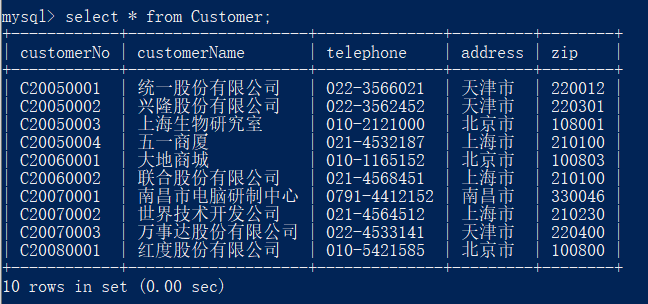
DBHelper helper = **new** DBHelper();

helper.queryEmployeeOrderBySalary(20);

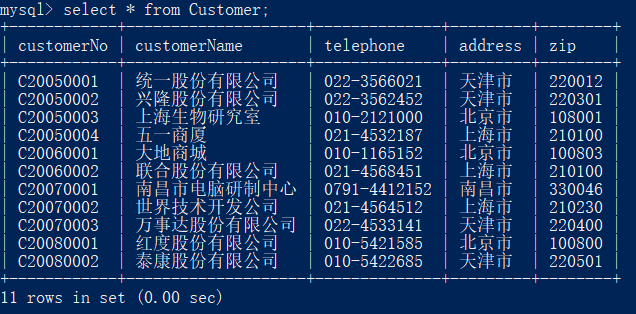
1. **//public** **class** DBHelper {
2. **private** SQLExecutor sqlExecutor = **new** SQLExecutor();
3. /\*\*
4. \* @param limit
5. \* 1）查询职工工资按从高到低排序的前20的职工编号、职工姓名和工资；
6. \*/
7. **public** **void** queryEmployeeOrderBySalary(**int** limit){
8. String sql = "select employeeNo,employeeName,salary " +
9. "from Employee order by salary desc limit " +
10. limit+";";
11. ResultSet resultSet = sqlExecutor.query(sql);
12. outputResultSet(resultSet,System.out); } 

测试用例如下

1. @Test
2. **public** **void** queryEmployeeOrderBySalary() {
3. String queryEmployeeSQL = "select \* from Employee;";
4. List<List<String>>q1 = helper.getValues(queryEmployeeSQL);
5. //call method.
6. ResultSet resultSet = helper.queryEmployeeOrderBySalary(20);
7. List<List<String>>result = helper.getValues(resultSet);
8. assertTrue(result.size() - 1 <= 20);//result.get(0) is colName.
9. **for**(**int** i = 1;i<result.size() - 1;i++){
10. **double** salary1 = Double.parseDouble(result.get(i).get(2));
11. **double** salary2 = Double.parseDouble(result.get(i+1).get(2));
12. assertTrue(salary1 >= salary2);//Test order desc.
13. }
14. List<List<String>>q2 = helper.getValues(queryEmployeeSQL);
15. assertEquals(q1.size(),q2.size());
16. **for**(**int** i = 0;i < q1.size();i++){
17. assertArrayEquals(q1.get(i).toArray(),q1.get(i).toArray());
18. }  //Assert the method didn't change the table.
19. }
20. 为客户表插入一条新的客户信息，客户编号“C20080002”，客户名称“泰康股份有限公司”，客户电话“010-5422685”，客户地址“天津市”，客户邮编“220501”；
21. **//public** **class** DBHelper
22. /\*\*
23. \* 2）为客户表插入一条新的客户信息，
24. \* 客户编号“C20080002”，客户名称“泰康股份有限公司”，客户电话“010-5422685”，客户地址“天津市”，客户邮编“220501”；
25. \* insert a customer.
26. \*/
27. **public** **void** insertCustomer(){
28. dropCustomer();
29. String sql = "insert Customer values('C20080002','泰康股份有限公司',  '010-5422685',  '天津市', '220501');";
30. sqlExecutor.update(sql);
31. System.out.println("插入数据成功");
32. }

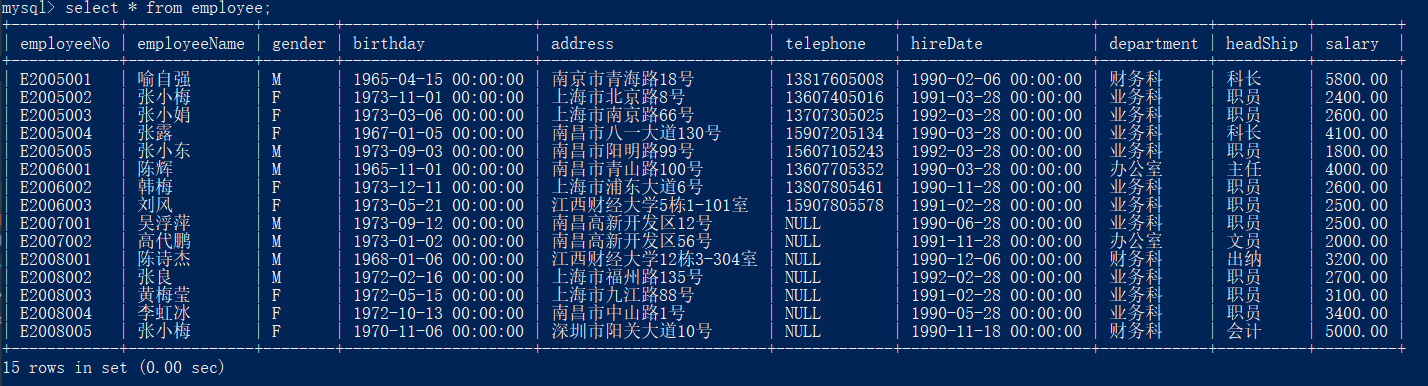


插入后



测试用例

1. @Test
2. **public** **void** insertCustomer() {
3. helper.dropCustomer();//if contains the customer,delete it.
4. String sql = "select \* from Customer where customerNo = \'C20080002\';";
5. //return first line if not empty,or return [];
6. String[] result1 = helper.getValue(sql);
7. assertEquals(result1.length,0);//Assert no such customer;
8. helper.insertCustomer();//calling method.
9. String[] result2 = helper.getValue(sql);//after calling the method
10. String[] expect2 = **new** String[]{"C20080002","泰康股份有限公司","010-5422685","天津市","220501"};
11. assertArrayEquals(result2,expect2);
12. }
13. 删除员工表中薪水高于5000的员工信息；
14. helper.deleteEmployeeSalaryOver(5000);//call.
15. /\*\*
16. \* 3）删除员工表中薪水高于5000的员工信息；
17. \* @param salaryLimit :the limit of salary.
18. \*/
19. **public** **void** deleteEmployeeSalaryOver(**int** salaryLimit){
20. String sql = "delete from Employee where salary > " +salaryLimit +";";
21. sqlExecutor.update(sql);
22. System.out.println("删除数据成功");
23. }

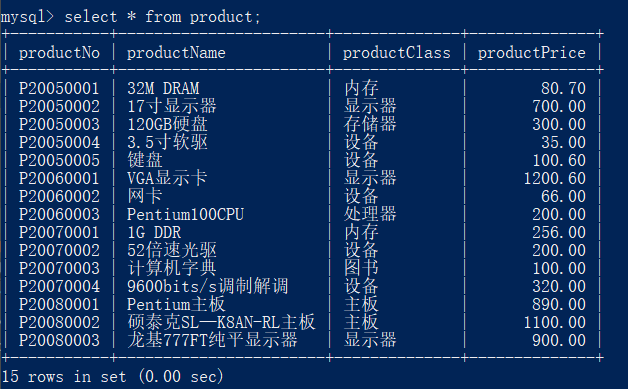


删除后

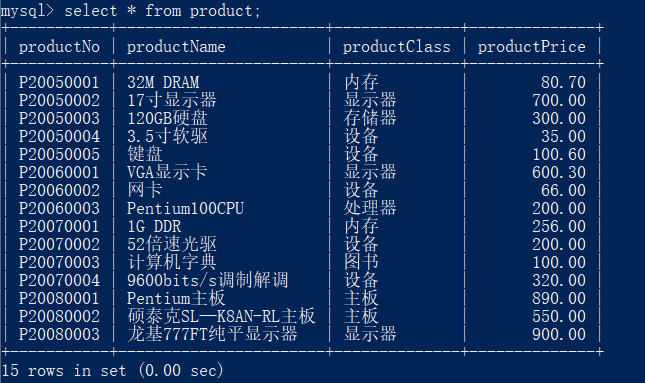


测试用例

1. @Test
2. **public** **void** deleteEmployeeSalaryOver() {
3. String insertSQL = "insert Employee values('E2005001','喻自强','M','19650415','南京市青海路18号','13817605008', '19900206','财务科','科长',5800);";
4. String sql = "select \* from Employee where salary > 5000";
5. List<List<String>>result1 = helper.getValues(sql);
6. **if**(result1.size() == 1){
7. helper.getSqlExecutor().update(insertSQL);//If has no employee's salary > 5000,insert an employee.
8. }
9. result1 = helper.getValues(sql);
10. assertNotEquals(result1.size(),1);//Before calling method:Contains at least one row.
11. helper.deleteEmployeeSalaryOver(5000);//Calling method.
12. result1 = helper.getValues(sql);    //After calling method.
13. assertEquals(result1.size(),1);//Only contains colName.
14. helper.displayTable("Employee");//display the employee table.
15. }
16. 更新商品基本信息表中价格超过1000的商品价格变为原来的50%；
17. /\*\*
18. \* 4）更新商品基本信息表中价格超过1000的商品价格变为原来的50%；
19. \* half the price if price over the limit.
20. \* sql = update Product set productPrice = 0.5 \* productPrice where productPrice > limit"
21. \* @param priceLimit:limit of price to half.
22. \*/
23. **public** **void** halfPriceProductPriceOver(**int** priceLimit){
24. String sql = "update Product set productPrice = 0.5 \* productPrice where productPrice > "+ priceLimit+ ";";
25. sqlExecutor.update(sql);
26. System.out.println("修改数据成功");
27. }
28. //call in Main class
29. helper.halfPriceProductPriceOver(1000);



修改后

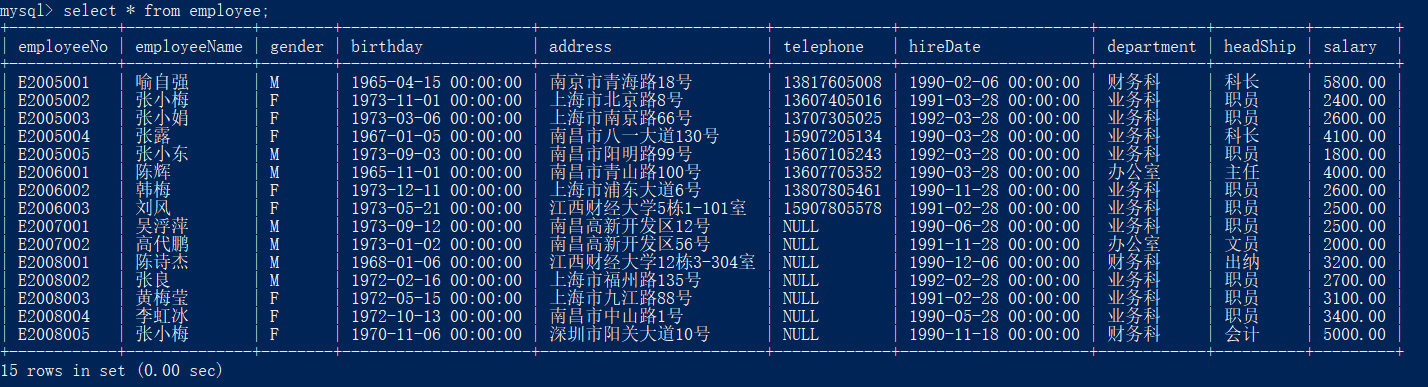


测试用例

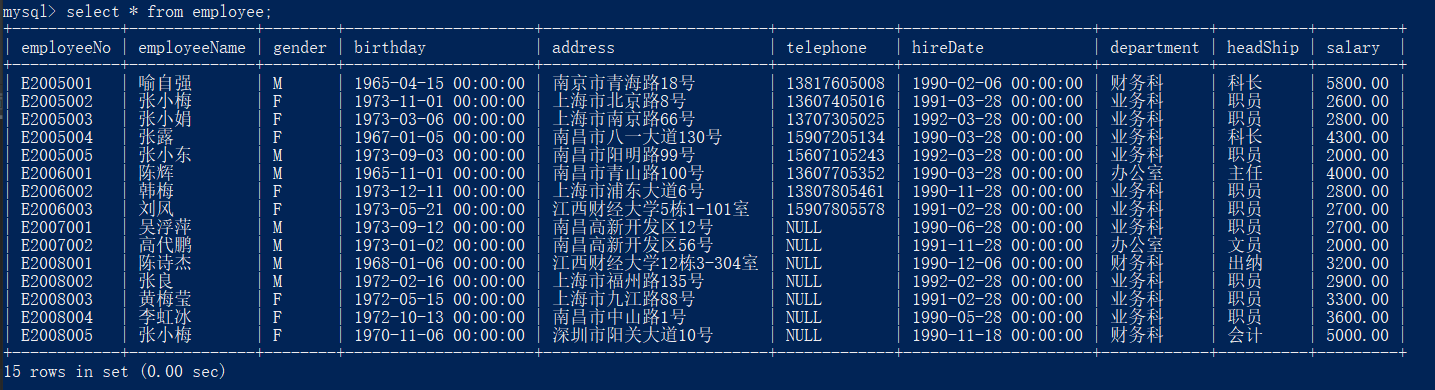
1. @Test
2. **public** **void** halfPriceProductPriceOver() {
3. String querySQL = "select productNo,productPrice from product";
4. List<List<String>>result1 = helper.getValues(querySQL);
5. helper.displayTable("Product");//display the old product table.
6. helper.halfPriceProductPriceOver(1000);//Calling method.
7. List<List<String>>result2 = helper.getValues(querySQL);
8. assertEquals(result1.size(),result2.size());
9. **for**(**int** i = 1;i < result1.size();i++){
10. **double** price1 = Double.parseDouble(result1.get(i).get(1));
11. **double** price2 = Double.parseDouble(result2.get(i).get(1));
12. **if**(price1 > 1000){  //if original price > 100,half it.
13. assertEquals(price1/2,price2,0.01);
14. }
15. **else**{  //else assert equals.
16. assertArrayEquals(result1.get(i).toArray(),result2.get(i).toArray());
17. }
18. }
19. helper.displayTable("Product");//display the new product table.
20. }
21. 使用高级程序设计语言中的动态SQL功能完成如下的任务（不保留4中对表信息的修改）：

这里使用prepareStatement进行动态SQL查询

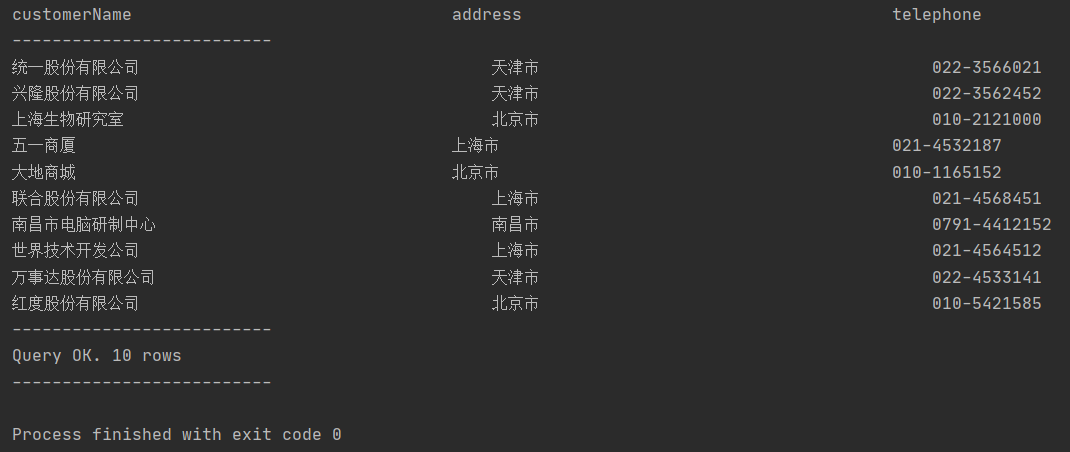
1. //SQLExecutor.java
2. **public** **void** dynamicUpdate(String sql,String []args){
3. **try**{
4. PreparedStatement stmt = conn.prepareStatement(sql);//prepare
5. **for**(**int** i = 0;i< args.length;i++){
6. stmt.setString(i+1,args[i]);  //set args.
7. }
8. stmt.execute();
9. }
10. **catch** (SQLException throwables){
11. }
12. }
13. 为“业务科”（作为外部输入参数）所有员工增加200的薪水。
14. /\*\*
15. \* 5-1）为“业务科”（作为外部输入参数）所有员工增加200的薪水。
16. \* @param department the employee's department.
17. \* @param increase:the salary increase.
18. \*/
19. **public** **void** addSalaryByDepartment(String department,**int** increase){
20. String sql = "update Employee set salary = salary + ? where department = ?;";
21. String[] args = **new** String[2];
22. args[0] = String.valueOf(increase);
23. args[1] = department;
24. sqlExecutor.dynamicUpdate(sql,args);
25. System.out.println("修改数据成功");
26. }



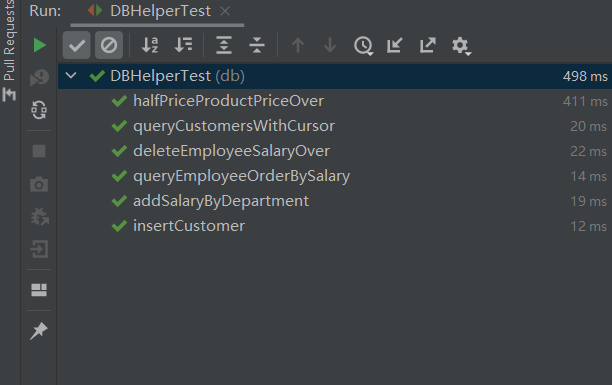
修改后



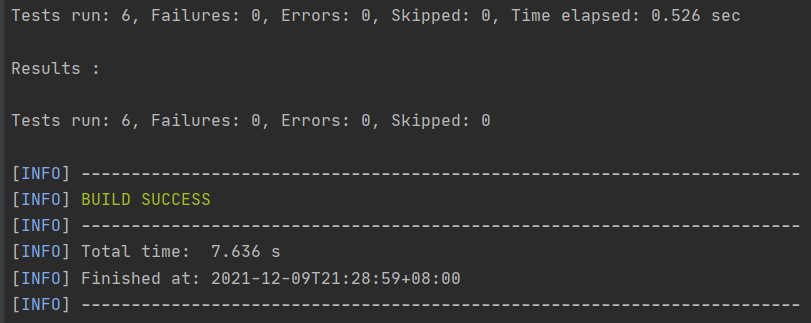
1. 查询客户表中的客户名称、客户地址及客户电话并输出。（注：结合游 标的使用）
2. **public** ResultSet queryWithCursor(String sql,**int** fetchSize){
3. ResultSet resultSet = **null**;
4. PreparedStatement stmt = **null**;
5. **try** {
6. stmt = conn.prepareStatement(sql);
7. stmt.setFetchSize(fetchSize);  //设置每次返回的数量
8. resultSet = stmt.executeQuery();
9. } **catch** (SQLException throwables) {
10. throwables.printStackTrace();
11. }
12. **return** resultSet;
13. }
14. **public** ResultSet queryCustomersWithCursor(){
15. String sql = "select customerName,address,telephone from Customer";
16. ResultSet resultSet = sqlExecutor.queryWithCursor(sql,1);
17. outputResultSet(resultSet,System.out);
18. **return** resultSet;
19. }
20. @Test
21. **public** **void** queryCustomersWithCursor(){
22. helper.queryCustomersWithCursor();
23. }



最终通过全部自编测试用例(6/6)



mvn test



详细代码，测试用例见附件或 https://github.com/Jinglong-Liu/jdbc-demo

实验中遇到的困难及解决办法

[详细说明你认为本次实验中比较困难的地方，也可以对实验设计提出建议]

1. 经实验验证，存储此过程的输入类型参数不能为varchar,必须指定长度，比如varchar(40)；另外char也是如此，如果类型为char,等价于char(1)调用时只接受长度为1的参数，不然会提示“data too long”。因此要写成char(8)之类

参考文献及致谢

https://www.fatalerrors.org/a/mysql-error-1418-hy000.html