《Java 应用技术》课程实验报告——数据库操作

应承峻 3170103456

1. 实验描述

编写 Java 程序，产生自动将数据库中一张表转换成类操作的 Java 程序。包括基本的数据库查询、显示与增、删、改操作。

输入指定数据库的表名，显示或产生 Java 程序文件。

2. 实验思路

为了实现通过 Java API 进行数据库操作的功能，我们定义了 DatabaseHelp 类，其包含了一个构造方法，连接和释放连接的方法，以及数据库的增删查改方法。

```
public class DatabaseHelper {

    private String username;
    private String password;
    private String url;
    private Connection conn;

    public DatabaseHelper(String username, String password, int port, String databaseName) {...}

    public void connect() throws SQLException {...}

    public void release() throws SQLException {...}

    public void insertRow(String tableName, Object[] values) throws SQLException {...}

    public void insertRow(String tableName, Vector<Pair> pairs) throws SQLException {...}

    /**
     * @apiNote conditions example: bookName='abc' and price>=100
     */
    public void removeRow(String tableName, String conditions) throws SQLException {...}

    public void select(String tableName) throws SQLException {...}

    public void select(String tableName, String[] columns) throws SQLException {...}

    public void select(String tableName, String conditions) throws SQLException {...}

    public void select(String tableName, String[] columns, String conditions) throws SQLException {...}

    public void updateRow(String tableName, Vector<Pair> pairs, String conditions) throws SQLException {...}

}
```

构造方法接收四个参数，分别为 MySQL 数据库的用户名、密码、端口和需要连接数据库的名称，url 为根据以上参数构造好的 JDBC 路径。

```
public DatabaseHelper(String username, String password, int port, String databaseName) {
    try {
        Class.forName("com.mysql.jdbc.Driver");
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    this.username = username;
    this.password = password;
    this.url = "jdbc:mysql://localhost:" + port + "/" + databaseName + "?characterEncoding=utf-8&useSSL=false";
}
```

插入方法共有 2 个重载函数，第一个重载函数接收一个 Object 的数组，表示需要插入的一行中所有的字段值，而第二个重载函数接收一个 Pair 类型的向量(Vector)，其表示需要插入的一行中某些键值对的值，Pair 类的定义如下：

```
public class Pair {
    Object key;
    Object value;
    public Pair(Object key, Object value) {
        this.key = key;
        this.value = value;
    }
}
```

插入代码的实现原理是，通过给定的键值对来构造动态 SQL 语句，并且使用动态绑定参数（PrepareStatement）的方式来防止 SQL 注入攻击。在这里采用 StringBuilder 而不是 String 作为构造字符串的原因是因为在这个过程中，需要遍历每一个键值对，并在字符串中拼接，而 StringBuilder 相比于 String，在多次动态拼接时具有非常高的效率。在字符串拼接完成后调用 executeUpdate()方法即可执行语句，当出现异常时会抛出异常。

```
public void insertRow(String tableName, Vector<Pair> pairs) throws SQLException {
    StringBuilder builder = new StringBuilder("insert into " + tableName + "(" + pairs.get(0).key);
    for (int i = 1; i < pairs.size(); i++) builder.append(",").append(pairs.get(i).key);
    builder.append(")").append(" values(?");
    for (int i = 1; i < pairs.size(); i++) builder.append(",?");
    builder.append(");");
    PreparedStatement ps = conn.prepareStatement(builder.toString());
    for (int i = 0; i < pairs.size(); i++) ps.setObject( parameterIndex: i + 1, pairs.get(i).value);
    ps.executeUpdate();
    ps.close();
    System.out.println("插入数据成功！");
}
```

删除一条语句则比较简单，其格式为”delete from <数据表名> where <条件>”，根据该方法传入两个参数即可。

```
/**
 * @apiNote conditions example: bookName='abc' and price>=100
 */
public void removeRow(String tableName, String conditions) throws SQLException {
    Statement stat = conn.createStatement();
    stat.executeUpdate( sql: "delete from " + tableName + " where " + conditions);
    System.out.println("删除记录成功！");
    stat.close();
}
```

查询的情况最为复杂，查询的格式可以是以下 4 种（对应 4 个重载函数）：

1. select * from <数据表名>;
2. select * from <数据表名> where <条件>;
3. select <字段 1，字段 2…> from <数据表名>;
4. select <字段 1，字段 2…> from <数据表名> where <条件>;

但实际上我们可以发现，case 1、case 3 分别是 case 2 和 case 4 的简化版，即：

select * from …; 等价于 select * from … where 1;

select * from … where …; 等价于 select <字段 1，字段 2…> from … where 1;

因此，case 1、case 3 的方法可以通过调用方法 2、4 实现，即：

```
public void select(String tableName) throws SQLException {
    select(tableName, conditions: "1");
}

public void select(String tableName, String[] columns) throws SQLException {
    select(tableName, columns, conditions: "1");
}
```

当情况为 case 4 时，我们同样通过 StringBuilder 来构造 SQL 字符串，然后将查询结果存储在 ResultSet 中，然后可以通过 ResultSet 得到字段的信息 ResultSetMetaData，通过 ResultSetMetaData 我们可以得到字段的个数和每个字段的名称。接下来需要统计记录的条数，一个简单而可行的方法是，先通过 last()方法将 ResultSet 的游标移动到最末尾一行，然后读出此时的行号并输出，再通过 beforeFirst()方法将游标移动到头部。

```java
public void select(String tableName, String[] columns, String conditions) throws SQLException {
    StringBuilder builder = new StringBuilder("select " + columns[0]);
    for (int i = 1; i < columns.length; i++) builder.append(",").append(columns[i]);
    builder.append(" from ").append(tableName).append(" where ").append(conditions);
    Statement stat = conn.createStatement();
    ResultSet rs = stat.executeQuery(builder.toString());
    ResultSetMetaData data = rs.getMetaData();
    rs.last();
    System.out.println("共查询到" + rs.getRow() + "条记录：");
    rs.beforeFirst();
    for (int i = 1; i <= data.getColumnCount(); i++) {
        System.out.printf("%-16s\t", data.getColumnName(i));
    }
    System.out.println();
    while (rs.next()) {
        for (int i = 1; i <= data.getColumnCount(); i++) {
            System.out.printf("%-16s\t", rs.getObject(data.getColumnName(i)));
        }
        System.out.println();
    }
    rs.close();
    stat.close();
}
```

修改数据与查询数据类似，其格式为”update <表名> set <P1=V1, P2=V2 …> where <条件>”，因此只需传入这三个参数构造动态字符串即可。

```java
public void updateRow(String tableName, Vector<Pair> pairs, String conditions) throws SQLException {
    StringBuilder builder = new StringBuilder("update " + tableName + " set " + pairs.get(0).key + " = ? ");
    for (int i = 1; i < pairs.size(); i++) builder.append(",").append(pairs.get(i).key).append(" = ? ");
    builder.append(" where ").append(conditions);
    PreparedStatement ps = conn.prepareStatement(builder.toString());
    for (int i = 0; i < pairs.size(); i++) ps.setObject( parameterIndex: i + 1, pairs.get(i).value);
    ps.executeUpdate();
    ps.close();
    System.out.println("修改数据成功！");
}
```

3. 实验结果

在本次实验中，我们通过一组测试数据和一系列的增、删、查、改操作来验证代码对数据库操作的正确性，测试程序如下：

```java
/* 插入方法一：依次按照每个字段插入 */
db.insertRow( tableName: "book", new Object[] {"1" , "Java程序设计", "测试者", "ZJU", "34.56"});
db.insertRow( tableName: "book", new Object[] {"2" , "算法导论", "测试者", "ABC", "66.66"});
db.insertRow( tableName: "book", new Object[] {"3" , "Python程序设计", "应承峻", "PDF", "18.22"});
/* 插入方法二：按照自定义的字段和顺序插入 */
Vector<Pair> pairs = new Vector<>();
pairs.add(new Pair("bookName", "计算机网络"));
pairs.add(new Pair("price", "100.00"));
pairs.add(new Pair("author", null));
db.insertRow( tableName: "book", pairs);

/* 查询方法一：查询表的所有列 */
db.select( tableName: "book");
System.out.println("--------------------------------");
/* 查询方法二：根据条件查询表的所有列 */
db.select( tableName: "book", conditions: "price >= 50 or author = '应承峻'");
System.out.println("--------------------------------");
/* 查询方法三：查询表的给定列 */
db.select( tableName: "book", new String[] {"bookName", "price"});
System.out.println("--------------------------------");
/* 查询方法四：根据条件查询表的给定列 */
db.select( tableName: "book", new String[] {"bookName", "price"}, conditions: "price >= 50 or author = '应承峻'");
System.out.println("--------------------------------");
```

```
/* 修改方法一：根据条件修改表的指定列 */
Vector<Pair> updatePairs = new Vector<>();
updatePairs.add(new Pair("price", "88.88"));
db.updateRow( tableName: "book", updatePairs,  conditions: "bookName='算法导论'");

db.select( tableName: "book");  //测试修改操作
System.out.println("---------------------------------");

/* 删除方法一：按照给定的条件删除 */
db.removeRow( tableName: "book",  conditions: " price > 60 and author = '测试者' ");

db.select( tableName: "book");  //测试删除操作
System.out.println("---------------------------------");
```

（1）在插入 4 条数据后，我们对整张数据表进行一次查询，得到以下输出结果：

```
插入数据成功!
插入数据成功!
插入数据成功!
插入数据成功!
共查询到4条记录：
id                  bookName            author              publish             price
1                   Java程序设计            测试者               ZJU                 34.56
2                   算法导论                测试者               ABC                 66.66
3                   Python程序设计          应承峻               PDF                 18.22
6                   计算机网络               null                null                100.0
---------------------------------
```

（2）根据条件（price >= 50 or author = '应承峻'）查询表的结果如下：

```
共查询到3条记录：
id                  bookName            author              publish             price
2                   算法导论                测试者               ABC                 66.66
3                   Python程序设计          应承峻               PDF                 18.22
6                   计算机网络               null                null                100.0
---------------------------------
```

（3）查询数据表中的给定列(bookName 和 price)，结果如下：

```
共查询到4条记录：
bookName            price
Java程序设计            34.56
算法导论                66.66
Python程序设计          18.22
计算机网络               100.0
---------------------------------
```

（4）根据(2)中的条件查询(3)中的给定列，结果如下：

```
共查询到3条记录：
bookName            price
算法导论                66.66
Python程序设计          18.22
计算机网络               100.0
---------------------------------
```

（5）根据条件（bookName='算法导论'）修改表的给定列（price=88.88）：

```
修改数据成功!
共查询到4条记录：
id                  bookName            author              publish             price
1                   Java程序设计            测试者               ZJU                 34.56
2                   算法导论                测试者               ABC                 88.88
3                   Python程序设计          应承峻               PDF                 18.22
6                   计算机网络               null                null                100.0
---------------------------------
```

（6）删除表中符合条件（price > 60 and author = '测试者'）的行：

```
删除记录成功!
共查询到3条记录：
id                  bookName            author              publish             price
1                   Java程序设计            测试者               ZJU                 34.56
3                   Python程序设计          应承峻               PDF                 18.22
6                   计算机网络               null                null                100.0
---------------------------------
```

从程序运行结果来看，通过了以上 6 个测试点，该程序能够实现数据库中间件的功能。

4. 实验心得

在本次实验中，我复习和回顾了上学期所学的数据库知识，以及了解了如何使用 JDBC 来连接数据库，并在数据库中进行操作，在实验中，主要遇到了如下问题：一开始直接写好后运行发现运行不了，原因是我们还没有把数据库的驱动导入到项目中，因此需要先下载数据库的驱动(com.mysql.jdbc.Driver)，然后打开 Project-Structures 将驱动导入到 Modules 的 Dependencies 中即可。

```
java.lang.ClassNotFoundException: com.mysql.jdbc.Driver <2 internal calls>
    at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:499)
    at java.base/java.lang.Class.forName0(Native Method)
    at java.base/java.lang.Class.forName(Class.java:291)
    at DatabaseHelper.<init>(DatabaseHelper.java:15)
    at Main.main(Main.java:6)
```