《数据库概论》实验三 使用高级程序设计语言访问数据库

实验报告

姓名 王明 学号 161220124 联系方式 1090616897@qq.com

实验环境

我使用的操作系统是 windows10、使用的数据库软件是 MySQL WorkBench8.0.12。

实验过程

说明：为了测试在 Java 中使用 SQL 语句能真正修改数据库的数据，我每条指令执行过后都进行了打印，然后再在 WorkBench 中查看表的情况，因此每条指令执行过后都有了 Java 截图和 WorkBench 截图。

1、在高级程序设计语言中通过数据库连接 API 直接访问 MySQL：

◆ 1.1. 连接数据库：

```
String driverName="com.mysql.jdbc.Driver";  //数据库驱动
String dbURL="jdbc:mysql://localhost:3306/exp2" +
        "?useUnicode=true&characterEncoding=utf-8&useSSL=false";  //数据库的url链接
String userName="root";  //数据库用户名
String userPwd="wmkljy";  //数据库的密码
Connection dbConn = null;  //链接初始化为空
try {
    Class.forName(driverName).newInstance();      //创建数据库驱动实例
    dbConn=DriverManager.getConnection(dbURL,userName,userPwd);  //创建与数据库的连接
    System.out.println("连接数据库成功");
```

◆ 1.2. 利用连接创建一个静态的状态用于之后的操作：

```
    //创建一个静态Statement
    Statement stmt = dbConn.createStatement();
```

◆ 1.3. SQL 查询：

```java
//查询语句
ResultSet rs = stmt.executeQuery( sql: "select * from Stuff where StuffNo = 3;");
ResultSetMetaData m = rs.getMetaData();   //得到返回结果集
int columns = m.getColumnCount();    //得到属性的数目
for(int i = 1; i <= columns; i++) {     //输出属性名
    System.out.printf("%-20s",m.getColumnName(i));
}
System.out.println();
while(rs.next()){          //格式化输出返回的结果
    for(int i = 1; i <= columns; i++)
    {
        System.out.printf("%-20s",rs.getString(i));
    }
    System.out.println();
}
System.out.println();
```

java 中的结果：

```
D:\Android\java-jdk\bin\java ...
连接数据库成功
StuffName           StuffNo             Age                 Salary              SectionNo
Tom                 3                   40                  27000               4
```

MySQL 中的结果：

	StuffName	StuffNo	Age	Salary	SectionNo
▶	Lily	1	22	30000	1
	Alice	2	26	33600	2
	Tom	3	40	27000	4
	Jim	4	25	31000	4
	Jack	5	33	46000	2
	Molly	6	30	40000	3
*	NULL	NULL	NULL	NULL	NULL

◆ 1.4. SQL 插入：

```java
//插入语句
stmt.executeUpdate( sql: "insert into Stuff values('Bob',7,27,38000,2);");
```

java 中的结果：

```
D:\Android\java-jdk\bin\java ...
连接数据库成功
StuffName           StuffNo             Age                 Salary              SectionNo
Lily                1                   22                  30000               1
Alice               2                   26                  33600               2
Tom                 3                   40                  27000               4
Jim                 4                   25                  31000               4
Jack                5                   33                  46000               2
Molly               6                   30                  40000               3
Bob                 7                   27                  38000               2
```

MySQL 中的结果：

StuffName	StuffNo	Age	Salary	SectionNo
Lily	1	22	30000	1
Alice	2	26	33600	2
Tom	3	40	27000	4
Jim	4	25	31000	4
Jack	5	33	46000	2
Molly	6	30	40000	3
Bob	7	27	38000	2
NULL	NULL	NULL	NULL	NULL

◆ 1.5. SQL 删除：

```
//删除语句
stmt.executeUpdate( sql: "delete from Stuff where StuffNo = 7;");
```

java 中的结果：

```
D:\Android\java-jdk\bin\java ...
连接数据库成功
StuffName       StuffNo         Age             Salary          SectionNo
Lily            1               22              30000           1
Alice           2               26              33600           2
Tom             3               40              27000           4
Jim             4               25              31000           4
Jack            5               33              46000           2
Molly           6               30              40000           3
```

MySQL 中的结果：

StuffName	StuffNo	Age	Salary	SectionNo
Lily	1	22	30000	1
Alice	2	26	33600	2
Tom	3	40	27000	4
Jim	4	25	31000	4
Jack	5	33	46000	2
Molly	6	30	40000	3
NULL	NULL	NULL	NULL	NULL

◆ 1.6. SQL 更新 1：

```
//更新语句
stmt.executeUpdate( sql: "update Stuff set Salary = 70000 where StuffNo = 6;");
```

java 中的结果：

```
D:\Android\java-jdk\bin\java ...
连接数据库成功
StuffName	StuffNo	Age	Salary	SectionNo
Lily	1	22	30000	1
Alice	2	26	33600	2
Tom	3	40	27000	4
Jim	4	25	31000	4
Jack	5	33	46000	2
Molly	6	30	70000	3
```

MySQL 中的结果：

StuffName	StuffNo	Age	Salary	SectionNo
Lily	1	22	30000	1
Alice	2	26	33600	2
Tom	3	40	27000	4
Jim	4	25	31000	4
Jack	5	33	46000	2
Molly	6	30	70000	3
NULL	NULL	NULL	NULL	NULL

◆ 1.7. SQL 更新 2：

```
//更新语句
stmt.executeUpdate( sql: "update Stuff set Age = 50 where StuffNo = 2;");
```

java 中的结果：

```
D:\Android\java-jdk\bin\java ...
连接数据库成功
StuffName	StuffNo	Age	Salary	SectionNo
Lily	1	22	30000	1
Alice	2	50	33600	2
Tom	3	40	27000	4
Jim	4	25	31000	4
Jack	5	33	46000	2
Molly	6	30	70000	3
```

MySQL 中的结果：

StuffName	StuffNo	Age	Salary	SectionNo
Lily	1	22	30000	1
Alice	2	50	33600	2
Tom	3	40	27000	4
Jim	4	25	31000	4
Jack	5	33	46000	2
Molly	6	30	70000	3
NULL	NULL	NULL	NULL	NULL

◆ 1.8. 断开连接：

```
//断开链接
dbConn.close();
```

2、在高级程序设计语言中通过数据库连接池来管理连接：

◆ 2.1. DBCP 配置：

```
//加载DBCP配置文件
static{
    try{
        FileInputStream is = new FileInputStream( name: "config/dbcp.properties");
        properties.load(is);
    }catch(IOException e){
        e.printStackTrace();
    }

    try{
        dataSource = BasicDataSourceFactory.createDataSource(properties);
    }catch(Exception e){
        e.printStackTrace();
    }
}
```

配置文件如下：

```
########DBCP配置文件##########
#驱动名
driverClassName=com.mysql.jdbc.Driver
#url
url=jdbc:mysql://localhost:3306/exp2?useUnicode=true&characterEncoding=utf-8&useSSL=false
#用户名
username=root
#密码
password=wmkljy
#初试连接数
initialSize=30
#最大活跃数
maxTotal=100
#最大idle数
maxIdle=20
#最小idle数
minIdle=5
#最长等待时间(毫秒)
maxWaitMillis=1000
#程序中的连接不使用后是否被连接池回收(该版本要使用removeAbandonedOnMaintenance和removeAbandonedOnBorrow)
#removeAbandoned=true
removeAbandonedOnMaintenance=true
removeAbandonedOnBorrow=true
#连接在所指定的秒数内未使用才会被删除(秒)(为配合测试程序才配置为1秒)
removeAbandonedTimeout=1
```

◆ 2.2. DBCP 连接数据库：

```
//从连接池中获取一个连接
public static Connection getConnection() {
    Connection connection = null;
    try {
        connection = dataSource.getConnection();
        System.out.println("连接数据库成功");
```

◆ 2.3. DBCP 利用连接创建一个静态的状态用于之后的操作：

```
//创建一个静态Statement
Statement stmt = connection.createStatement();
```

◆ 2.4. DBCP 使用 SQL 查询：

```
//查询语句
ResultSet rs = stmt.executeQuery( sql: "select * from Stuff where StuffNo = 5;");
```

java 中的结果：

```
D:\Android\java-jdk\bin\java ...
连接数据库成功
StuffName       StuffNo         Age             Salary          SectionNo
Jack            5               33              46000           2
```

MySQL 中的结果：

StuffName	StuffNo	Age	Salary	SectionNo
Lily	1	22	30000	1
Alice	2	50	33600	2
Tom	3	40	27000	4
Jim	4	25	31000	4
Jack	5	33	46000	2
Molly	6	30	70000	3
NULL	NULL	NULL	NULL	NULL

◆ 2.5. DBCP 关闭连接：

```
try {
    connection.setAutoCommit(false);
    connection.close();
} catch (SQLException e) {
    e.printStackTrace();
}
```

◆ 2.6.直接连接模式和连接池模式的效率比较：

实验 1：

内容：

使用直接连接模式打开连接、执行 SQL、关闭连接一共 2000 次。

结果：

```
start time:1544629584258; end time:1544629590456; Run Time:6198(ms)
```

实验 2：

内容：

使用连接池模式打开连接、执行 SQL、关闭连接一共 2000 次。

连接池的配置：最大允许的连接数：5。

线程设置：使用单线程。

结果：

```
start time:1544629851381; end time:1544629852850; Run Time:1469(ms)
```

实验 3：

内容：

使用连接池模式打开连接、执行 SQL、关闭连接一共 2000 次。

连接池的配置：最大允许的连接数：5。

线程设置：使用 2 个线程。

结果：

```
start time:1544629921298; end time:1544629922219; Run Time:921(ms)
```

实验 4：

内容：

使用连接池模式打开连接、执行 SQL、关闭连接一共 2000 次。

连接池的配置：最大允许的连接数：5。

线程设置：使用 5 个线程。

结果：

```
start time:1544630386317; end time:1544630386799; Run Time:482(ms)
```

实验 5：

内容：

使用连接池模式打开连接、执行 SQL、关闭连接一共 2000 次。

连接池的配置：最大允许的连接数：5。

线程设置：使用 8 个线程。

结果：

```
start time:1544630449370; end time:1544630449849; Run Time:479(ms)
```

实验 6：

内容：

使用连接池模式打开连接、执行 SQL、关闭连接一共 2000 次。

连接池的配置：最大允许的连接数：30。

线程设置：使用 20 个线程。

结果：

```
start time:1544630735647; end time:1544630736056; Run Time:409(ms)
```

实验 7：

内容：

使用连接池模式打开连接、执行 SQL、关闭连接一共 2000 次。

连接池的配置：最大允许的连接数：30。

线程设置：使用 30 个线程。

结果：

```
start time:1544630555130; end time:1544630555588; Run Time:458(ms)
```

汇总：

实验序号	实验模式	线程数	连接池配置	用时（毫秒）
1	直接连接	1	无	6198
2	连接池	1	5	1469
3	连接池	2	5	921
4	连接池	5	5	482
5	连接池	8	5	479
6	连接池	20	30	409
7	连接池	30	30	458

结论：

1、连接池的连接效率比直接连接要高的多。
2、连接池使用并发执行可以大大提高连接效率。
3、连接池的最大连接数会限制连接效率。
4、提升线程数会提升效率，但是并发的线程数并不是越多就效率越高，当线程数超过一定的阈值，可以发现执行效率是下降的。

实验中遇到的困难及解决办法

遇到的主要的问题是 Java 多线程的程序使用，这个之前完全没有接触过，后来在网上找资料自学，才实现了多线程的功能。不得不佩服多线程并发的执行效率。

参考文献及致谢

参考网站：
http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html
致谢：非常感谢博主对于多线程的总结，让我很快入门 java 多线程的程序编写。