MYSQL 和 JDBC 整合

第一部分 MYSQL

第一节 DDL（data definition languages）

DDL（data definition languages）：定义数据段，数据库，表，列，索引等数据库对象，主要的语句关键字包括`CREATE`、`DROP`、`ALTER`等。

一、CREATE（库，表）

（一）命名规则

- 数据库名不得超过30个字符，变量名限制为29个
- 必须只能包含 A–Z, a–z, 0–9, _共63个字符
- 不能在对象名的字符间留空格
- 必须不能和用户定义的其他对象重名
- 必须保证你的字段没有和保留字、数据库系统或常用方法冲突
- 保持字段名和类型的一致性,在命名字段并为其指定数据类型的时候一定要保证一致性。假如数据类型在一个表里是整数,那在另一个表里就不要变成字符型了

（二）创建库

```
CREATE database xxxxx;
SHOW databases;
USE employees;
```

（三）创建表

1.创建全新的表

- 表名
- 列名, 数据类型, 尺寸

```
CREATE TABLE dept(
deptno INT(2),
dname VARCHAR(14),
loc VARCHAR(13));
```

2.基于现有的表创建全新的表

```
create table emp1 as select * from employees;
create table emp2 as select * from employees where
1=2; --创建的emp2是空表
```

二、ALTER

（一）添加列（add）

```
ALTER TABLE dept80
ADD job_id varchar(15);
```

（二）修改列（modify）

```
ALTER TABLE dept80
MODIFY (last_name VARCHAR(30));
```

（三）删除列（drop）

```
ALTER TABLE  dept80
DROP COLUMN  job_id;
```

（四）重命名列（change）

```
ALTER TABLE  dept80
CHANGE department_name dept_name varchar(15);
```

（五）修改表名（表操作）

```
ALTER table dept
RENAME TO detail_dept;
```

三、删除表与清空表

（一）删除表

```
DROP TABLE dept80;
```

（二）清空表

- TRUNCATE TABLE语句:
  - 删除表中所有的数据
  - 释放表的存储空间

```
TRUNCATE TABLE detail_dept;
```

- TRUNCATE语句不能回滚
- 可以使用 DELETE 语句删除数据,可以回滚

第二节 DML（Data Manipulation Language）（数据）

DML（Data Manipulation Language）：用于添加、删除。更新和查询数据库记录，主要依据关键字包括：INSERT、DELETE、UPDATE、SELECT 等。

一、INSERT （INTO）

（一）语法限制

使用这种语法一次只能向表中插入一条数据。

为每一列添加一个新值。

按列的默认顺序列出各个列的值。

（二）插入空值

- **向表中插入空值**
  - 隐式方式：在列名表中省略该列的值。

```
INSERT INTO departments
(department_id,department_name )
VALUES (30, 'Purchasing');
```

  - 显示方式：在VALUES 子句中指定空值。

```
INSERT INTO departments
VALUES    (100, 'Finance', NULL, NULL);
```

（三）拷贝数据

- 在 INSERT 语句中加入子查询。
- **不必书写 VALUES 子句。**
- 子查询中的值列表应与 INSERT 子句中的列名对应。

```
INSERT INTO emp2
SELECT *
FROM employees
WHERE department_id = 90;
```

```
INSERT INTO sales_reps(id, name, salary,
commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM   employees
WHERE  job_id LIKE '%REP%';
```

二、DELETE

- 使用 WHERE 子句删除指定的记录。

```
DELETE FROM departments
WHERE  department_name = 'Finance';
```

- 如果省略 WHERE 子句，则表中的全部数据将被删除

```
DELETE FROM  copy_emp;
```

三、UPDATE （SET）

（一）注意

可以一次更新**多条**数据。

如果需要回滚数据，需要保证在DML前，进行设置：**SET AUTOCOMMIT = FALSE;**

（二）具体操作

- 使用 **WHERE** 子句指定需要更新的数据。

```
UPDATE employees
SET    department_id = 70
WHERE  employee_id = 113;
```

- 如果省略 WHERE 子句，则表中的所有数据都将被更新。

```
UPDATE  copy_emp
SET     department_id = 110;
```

四、SELECT （FROM）（WHERE）（ORDER BY）（HAVING）

（一）基础操作

1.列别名，紧跟或用 AS

2.去重复行 DISTINCT

3.显示表结构 DESC

（二）运算符

1.基本运算符（= > <）

2.其他运算符

（1）BETWEEN…AND

（2）IN（set）

（3）LIKE %_

（4）IS NULL

3.逻辑运算符(AND，OR，NOT)

（三）排序与分页

1.排序 ORDER BY ASC/DESC

2.分页 LIMIT

```
--前10条记录：
SELECT * FROM table LIMIT 0,10;

--第11至20条记录：
SELECT * FROM table LIMIT 10,10;

--第21至30条记录：
SELECT * FROM table LIMIT 20,10;
```

（四）多表查询

1.等值连接和非等值连接

非等值链接示例，用 WHERE 来限定

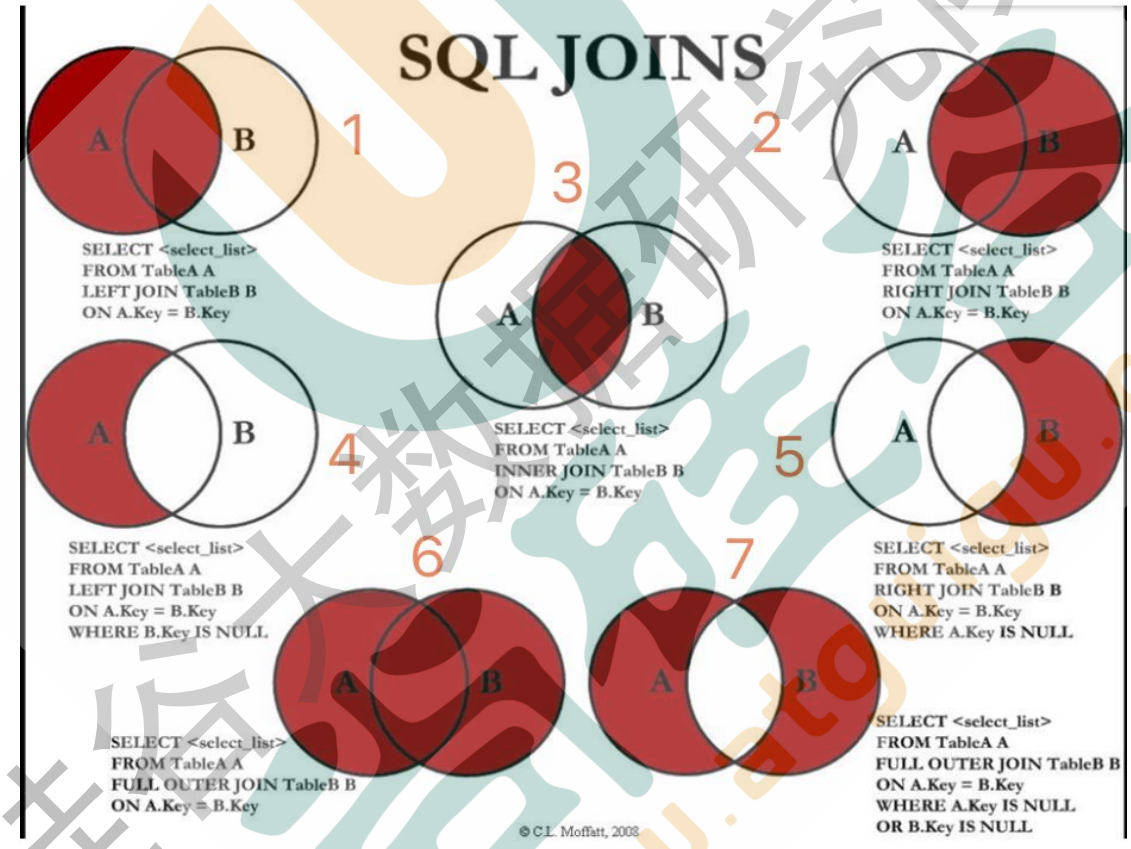
```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e, job_grades j
WHERE  e.salary BETWEEN j.lowest_sal AND
j.highest_sal;
```

2.自连接和非自连接

3.内连接和外连接

内连接：合并具有同一列的两个以上的表的行,结果集中不包含一个表与另一个表不匹配的行

外连接：两个表在连接过程中除了返回满足连接条件的行以外还返回左（或右）表中不满足条件的行，这种连接称为左（或右）外连接。没有匹配的行时，结果表中相应的列为空(NULL)。



（1）左外连接

```
SELECT last_name,department_name
FROM employees e LEFT JOIN departments d
ON e.`department_id` = d.`department_id`;
```

（2）右外连接

```sql
SELECT last_name,department_name
FROM employees e RIGHT JOIN departments d
ON e.`department_id` = d.`department_id`;
```

(3)

```sql
SELECT last_name,department_name
FROM employees e INNER JOIN departments d
On e.department_id = d.department_id;
```

(4)

```sql
SELECT last_name,department_name
FROM employees e LEFT JOIN departments d
ON e.`department_id` = d.`department_id`
WHERE d.`department_id` IS NULL;
```

(5)

```sql
SELECT last_name,department_name
FROM employees e RIGHT JOIN departments d
ON e.`department_id` = d.`department_id`
WHERE e.`department_id` IS NULL;
```

(6)

```sql
SELECT last_name,department_name
FROM employees e LEFT JOIN departments d
ON e.`department_id` = d.`department_id`
WHERE d.`department_id` IS NULL
UNION ALL
SELECT last_name,department_name
FROM employees e RIGHT JOIN departments d
ON e.`department_id` = d.`department_id`;
```

(7)

```
SELECT last_name,department_name
FROM employees e LEFT JOIN departments d
ON e.`department_id` = d.`department_id`
WHERE d.`department_id` IS NULL
UNION ALL
SELECT last_name,department_name
FROM employees e RIGHT JOIN departments d
ON e.`department_id` = d.`department_id`
WHERE e.`department_id` IS NULL;#与左下相比多了这条条件判断
```

（五）单行函数

1.字符串函数

2.数值函数

3.日期函数

4.流程函数

（1）IF

（2）IFNULL

（3）CASE WHEN THEN END

```
SELECT employee_id,salary, CASE WHEN salary>=15000
THEN '高薪'
        WHEN salary>=10000 THEN '潜力股'
        WHEN salary>=8000 THEN '屌丝'
        ELSE '草根' END  "描述"
FROM employees;
```

```
SELECT oid,`status`, CASE `status` WHEN 1 THEN '未付
款'
                    WHEN 2 THEN '已付款'
                    WHEN 3 THEN '已发货'
                    WHEN 4 THEN '确认收货'
                    ELSE '无效订单' END
FROM t_order;
```

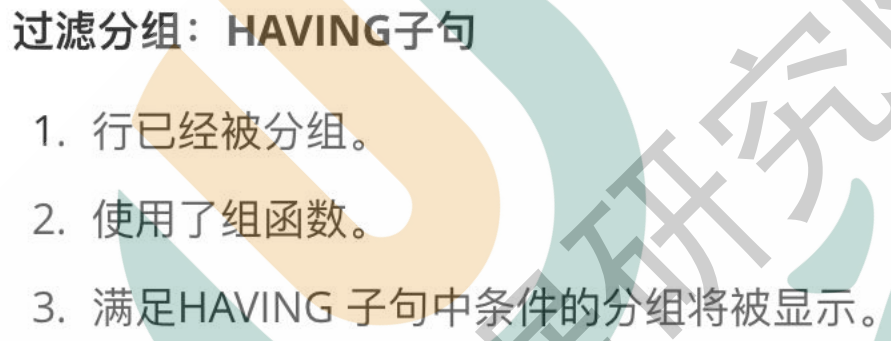
（六）分组函数

1.AVG、SUM、MIN、MAX、COUNT

2.GROUP BY

```
SELECT    department_id, AVG(salary)
FROM      employees
GROUP BY department_id ;
```

3.HAVING




```
SELECT    department_id, MAX(salary)
FROM      employees
GROUP BY department_id
HAVING    MAX(salary)>10000 ;
```

（七）子查询（条件嵌套，不断向外得到结果）

1.单行子查询（返回 1 行）

（1）普通子查询

```
SELECT last_name
FROM   employees
WHERE  salary >
                (SELECT salary
                 FROM   employees
                 WHERE  last_name = 'Abel');
```

（2）子查询中的 HAVING

题目：查询最低工资大于50号部门最低工资的部门id和其最低工资

```
SELECT    department_id, MIN(salary)
FROM      employees
GROUP BY  department_id
HAVING    MIN(salary) >
                        (SELECT MIN(salary)
                         FROM   employees
                         WHERE  department_id = 50);
```

2.多行子查询（IN ANY ALL）

题目：返回其它job_id中比job_id为'IT_PROG'部门所有工资都低的员工的员工号、姓名、job_id以及salary

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL                    9000, 6000, 4800,4200
                   (SELECT salary
                    FROM   employees
                    WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

3.相关子查询

问题：查询员工中工资大于本部门平均工资的员工的last_name,salary和其department_id

方式一：相关子查询

```
SELECT last_name, salary, department_id
FROM   employees outer
WHERE  salary >
          (SELECT AVG(salary)
           FROM   employees
           WHERE  department_id =
                  outer.department_id) ;
```

五、约束

（一）概述

1.约束的定义：为了保证数据的一致性和完整性，SQL 规范以约束的方式对表数据进行额外的条件限制。
2.约束的分类

- 可以在**创建表时规定约束（通过 CREATE TABLE 语句）**，或者在**表创建之后也可以（通过 ALTER TABLE 语句）**
- **根据约束数据列的限制**，约束可分为：
  - **单列约束**：每个约束只约束一列
  - **多列约束**：每个约束可约束多列数据
- **根据约束的作用范围**，约束可分为：
  - **列级约束**：只能作用在一个列上，跟在列的定义后面
  - **表级约束**：可以作用在多个列上，不与列一起，而是单独定义
- **根据约束起的作用**，约束可分为：
  - **NOT NULL 非空约束，规定某个字段不能为空**
  - **UNIQUE 唯一约束，规定某个字段在整个表中是唯一的**
  - **PRIMARY KEY 主键(非空且唯一)约束**
  - **FOREIGN KEY 外键约束**
  - **CHECK 检查约束**
  - **DEFAULT 默认值约束**

（二）NOT NULL

（三）UNIQUE

（四）PRIMARY KEY

（五）FOREIGN KEY

（六）CHECK（无效）

（七）DEFAULT

第三节 DCL（Data Control Language）

DCL（Data Control Language）：用于控制不同的数据段直接的许可和访问级别的语句，定义了数据库表，列，用户的访问权限和安全级别。

主要的语句关键词包括：GRANT、REVOKE、COMMIT、ROLLBACK、SAVEPOINT 等

一、数据库事务

（一）事务组成

1.事务：一组逻辑操作单元，使数据从一种状态变换到另一种状态。

2.组成部分：

（1）一个或多个 DML 语句

（2）一个 DDL

（3）一个 DCL

（二）处理事务

1.宏观处理事务的理论

当在一个事务中执行多个操作时，要么所有的事务都被提交(commit)，那么这些修改就永久地保存下来；要么数据库管理系统将放弃所作的所有修改，整个事务回滚(rollback)到最初状态。

2.具体过程

（1）设置提交状态：SET AUTOCOMMIT = FALSE;**

（2）以第一个 DML 语句的执行作为开始

（3）以下面的其中之一作为结束:

1）COMMIT 或 ROLLBACK 语句**

2）DDL 语句（**自动提交**）

3）用户会话正常结束

4）系统异常终止

（三）事物的 ACID 属性

1.原子性（Atomicity）——事务不可分割，要么都发生，要么都不发生。

2.一致性（Consistency）——总额不变，一致性状态转变为另一个一致性状态。

3.隔离性（Isolation）——一个事务的执行不能被其他事务干扰。

4.持久性（Durability）——事务一旦提交，改变即是永久性的，接下来其他操作和数据库故障不会有影响。

（四）数据库的隔离级别

1.数据库的并发问题

在同时运行的情况下的讨论

（1）脏读：读了未提交的数据，若回滚，则无效

（2）不可重复读：读了提交前的数据，提交后再读，值不同

（3）幻读：读了一个列，新插入以后保存，多了行

2.4 种隔离级别

（1）读未提交数据（READ UNCOMMITTED）

（2）读已提交数据（READ COMMITED）（解决脏读）

（3）可重复读（REPEATABLE READ）（解决脏读和不可重复读）

（4）串行化（SERIALIZABLE）（解决脏读，不可重复读，幻读）

第二部分 JDBC

第一节 获取数据库连接

一、手写的连接：JDBCUtils.getConnection();

（一）四要素及 jdbc.properties

其中，配置文件声明在工程的src目录下：【jdbc.properties】

```
user=root
password=abc123
url=jdbc:mysql://localhost:3306/test
driverClass=com.mysql.jdbc.Driver
```

（二）手写的连接

```
@Test
    public  void testConnection5() throws Exception {
      //1.加载配置文件
        InputStream is =
ConnectionTest.class.getClassLoader().getResourceAsStream("jdbc.properties");//输入流
        Properties pros = new Properties();
        pros.load(is);

        //2.读取配置信息
        String user = pros.getProperty("user");
        String password = pros.getProperty("password");
        String url = pros.getProperty("url");
        String driverClass = pros.getProperty("driverClass");

        //3.加载驱动
        Class.forName(driverClass);

        //4.获取连接
        Connection conn = DriverManager.getConnection(url,user,password);
        System.out.println(conn);

    }
```

//此处 throws 是因为若连接都无法进行，则没有必要进行下一步

二、使用数据库连接池：C3P0；DBCP；Druid

（一）C3P0（配合配置文件）

```java
//整个项目中，我们只需要提供唯一的一个数据库连接池即可。
private static ComboPooledDataSource cpds = new ComboPooledDataSource("helloc3p0");
public static Connection getConnection1() throws SQLException{
    Connection conn = cpds.getConnection();
    return conn;
}
```

其中 helloc3p0 的文件为 xml 格式，其中写有如下信息：

```
cpds.setDriverClass( "com.mysql.jdbc.Driver" );
cpds.setJdbcUrl( "jdbc:mysql:///test" );
cpds.setUser("root");
cpds.setPassword("abc123");
//设置如下的参数，实现对数据库连接池的管理
cpds.setInitialPoolSize(10);
cpds.setMaxPoolSize(100);
```

（二）DBCP（配合配置文件）

```java
//整个项目中，我们只需要提供唯一的一个数据库连接池即可。
private static DataSource dataSource = null;
static{
    try {
        Properties pros = new Properties();
        InputStream is = ClassLoader.getSystemClassLoader().getResourceAsStream("dbcp.properties");
        pros.load(is);
        dataSource = BasicDataSourceFactory.createDataSource(pros);
    } catch (Exception e) {
        e.printStackTrace();
    }

}
public static Connection getConnection2() throws SQLException{

    Connection conn = dataSource.getConnection();
    return conn;
}
```

配置文件 dbcp.properties 信息为

```
username=root
password=abc123
url=jdbc:mysql://localhost:3306/test
driverClassName=com.mysql.jdbc.Driver

maxActive = 20
```

（三）Druid（配合配置文件）

```java
//整个项目中，我们只需要提供唯一的一个数据库连接池即可。
private static DataSource dataSource1 = null;
static{
    try {
        Properties pros = new Properties();

        InputStream is = ClassLoader.getSystemClassLoader().getResourceAsStream("druid.properties");
        pros.load(is);
        dataSource1 = DruidDataSourceFactory.createDataSource(pros);
    } catch (Exception e) {
        e.printStackTrace();
    }

}
public static Connection getConnection3() throws SQLException{

    Connection conn = dataSource1.getConnection();
    return conn;
}
```

配置文件 druid.properties 信息为

```
driverClassName=com.mysql.jdbc.Driver
username=root
password=abc123
url=jdbc:mysql:///test
```

第二节 对数据表进行一系列 CRUD 操作

一、使用 PreparedStatement 实现通用的增删改，查询操作（version 1.0 \ version 2.0）-1.0 与 2.0 的区别就在于处理事务（理解线程）

（一）PreparedStatement 实现通用的增删改（2.0）

```java
public int update(Connection conn, String sql, Object... args) {// args:用于填充占位符。要求：sql中的占位符的个数与args的长度相同

    PreparedStatement ps = null;
    try {

        // 1.获取PreparedStatement(或：预编译sql语句)
        ps = conn.prepareStatement(sql);

        // 2.填充占位符
        for (int i = 0; i < args.length; i++) {
            ps.setObject(i + 1, args[i]);
        }

        // 3.执行
        ps.execute();
        int count = ps.executeUpdate();
        return count;
    } catch (Exception e) {
        e.printStackTrace();
    } finally {

        // 4.关闭资源
        JDBCUtils.closeResource(null, ps);
    }

    return 0;
}
```

2.0 相比于 1.0 版本，不自己 new connection，而是采用外部传参的方式

（二）PreparedStatement 实现查询操作（2.0）

```java
public T query(Connection conn,Class<T> clazz, String sql, Object... args) {

    PreparedStatement ps = null;
    ResultSet rs = null;
    try {
        // 1.预编译sql语句
        ps = conn.prepareStatement(sql);
        // 2.填充占位符
        for (int i = 0; i < args.length; i++) {
            ps.setObject(i + 1, args[i]);
        }

        // 3.执行,得到结果集
        rs = ps.executeQuery();
        // 4. 获取结果集的元数据
        ResultSetMetaData rsmd = rs.getMetaData();
        // 4.1 通过结果集的元数据获取结果集的列数
        int columnCount = rsmd.getColumnCount();
        if (rs.next()) {
            // 创建相应类的对象
            T t = clazz.newInstance();
            // 给t对象的各个查询到的字段对应的属性赋值
            for (int i = 0; i < columnCount; i++) {
                Object value = rs.getObject(i + 1);

                // 4.2 获取表的别名（getColumnLable()),此列名就是对象的属性名
                String columnLabel = rsmd.getColumnLabel(i + 1);
                // // 通过反射，给t对象的各个属性赋值
                Field field = clazz.getDeclaredField(columnLabel);
                field.setAccessible(true);
                field.set(t, value);

            }

            return t;

        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        // 5.关闭资源
        JDBCUtils.closeResource(null, ps, rs);
    }

    return null;

}
```

（三）PreparedStatement 实现查询返回多条数据的操作（2.0）

```java
public List<T> queryForList(Connection conn,Class<T> clazz, String sql, Object... args) {
    PreparedStatement ps = null;
    ResultSet rs = null;
    // 提供一个存放查询到的多个对象的集合
    ArrayList<T> list = new ArrayList<>();
    try {
        // 1.预编译sql语句
        ps = conn.prepareStatement(sql);
        // 2.填充占位符
        for (int i = 0; i < args.length; i++) {
            ps.setObject(i + 1, args[i]);
        }

        // 3.执行,得到结果集
        rs = ps.executeQuery();
        // 4. 获取结果集的元数据
        ResultSetMetaData rsmd = rs.getMetaData();
        // 4.1 通过结果集的元数据获取结果集的列数
        int columnCount = rsmd.getColumnCount();
        while (rs.next()) {
            // 创建相应类的对象
            T t = clazz.newInstance();
            // 给t对象的各个查询到的字段对应的属性赋值
            for (int i = 0; i < columnCount; i++) {
                Object value = rs.getObject(i + 1);

                // 4.2 获取表的别名 (getColumnLable()),此列名就是对象的属性名
                String columnLabel = rsmd.getColumnLabel(i + 1);
                // // 通过反射，给t对象的各个属性赋值
                Field field = clazz.getDeclaredField(columnLabel);
                field.setAccessible(true);
                field.set(t, value);

            }

            list.add(t);

        }
        return list;
```

（四）PreparedStatement 实现查询返回特殊数值的操作（2.0）

```java
public <E> E queryValue(Connection conn,String sql, Object... args) {

    PreparedStatement ps = null;
    ResultSet rs = null;
    try {
        // 1.预编译sql语句
        ps = conn.prepareStatement(sql);
        // 2.填充占位符
        for (int i = 0; i < args.length; i++) {
            ps.setObject(i + 1, args[i]);
        }

        // 3.执行,得到结果集
        rs = ps.executeQuery();
        // 4. 获取结果集的元数据
        ResultSetMetaData rsmd = rs.getMetaData();
        // 4.1 通过结果集的元数据获取结果集的列数
        int columnCount = rsmd.getColumnCount();
        if (rs.next()) {

            Object value = rs.getObject(1);

            return (E) value;
        }

    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        // 5.关闭资源
        JDBCUtils.closeResource(null, ps, rs);
    }

    return null;

}
```

二、使用 DBUtils 提供的 jar 包中的 QueryRunner 类

（一）Runner 的增删改操作

```java
@Test
public void testInsert() {


    Connection conn = null;
    try {
        QueryRunner runner = new QueryRunner();

        conn = JDBCUtils.getConnection3();
        //通过数据库连接池获取数据连接

        String sql = "insert into customers(name,email,birth)values(?,?,?)";
        int count = runner.update(conn, sql, "张可可","zhangkk@126.com",new Date(253454234324L));

        System.out.println("添加了" + count + "条记录");
    } catch (SQLException e) {
        e.printStackTrace();
    }finally{
        JDBCUtils.closeResource(conn, null);
    }
}
```

（二）Runner 的查询操作

1.查询一条信息（这里用 BeanHandler，还可以用 MapHandler-Map 形式，键值对）

```
 * 查询表中的一条记录，使用BeanHandler
 */
@Test
public void testQuery1() throws Exception{

    QueryRunner runner = new QueryRunner();
    Connection conn = JDBCUtils.getConnection3();//建立连接
    String sql = "select id,name,email,birth from customers where id = ?";
    BeanHandler<Customer> resultSetHandler = new BeanHandler<>(Customer.class);//接数据
    Customer customer = runner.query(conn, sql,resultSetHandler, 1);//把数据写入customer类中
    System.out.println(customer);
}
```

```
 * 查询表中的一条记录，以键值对的方式显示。使用MapHandler
 */
@Test
public void testQuery3() throws Exception{

    QueryRunner runner = new QueryRunner();
    Connection conn = JDBCUtils.getConnection3();
    String sql = "select id,name,email,birth from customers where id = ?";

    MapHandler resultSetHandler = new MapHandler();

    Map<String, Object> map = runner.query(conn, sql,resultSetHandler, 8);

    System.out.println(map);

}
```

2.查询多条信息

```
 * 查询表中的多条记录，使用BeanListHandler
 */
@Test
public void testQuery2() throws Exception{

    QueryRunner runner = new QueryRunner();
    Connection conn = JDBCUtils.getConnection3();
    String sql = "select id,name,email,birth from customers where id < ?";
    BeanListHandler<Customer> resultSetHandler = new BeanListHandler<>(Customer.class);
    List<Customer> list = runner.query(conn, sql,resultSetHandler, 8);
    list.forEach(System.out::println);
}
```

3.查询特殊值

```
 * 查询特殊值，使用ScalarHandler
 *
 */
@Test
public void testQuery4() throws Exception{
    QueryRunner runner = new QueryRunner();
    Connection conn = JDBCUtils.getConnection2();

    String sql = "select min(birth) from customers";
    ScalarHandler handler = new ScalarHandler();
    Date minBirth = (Date) runner.query(conn, sql, handler);

    System.out.println(minBirth);
}
```

第三节 关闭连接等操作

一、手写的 JDBCUtils.closeResourse();

二、DBUtils 提供的关闭相关操作

实际上这两项都一样，都是关闭 connection，statement 和 resultset 外加判断语句和 try-catch 语句。

第四节 数据库连接池

一、JDBC 数据库连接池的必要性

（一）传统模式

1.工作步骤

（1）在主程序中建立数据库连接

（2）进行 sql 操作

（3）断开数据库连接

2..问题：

（1）每次都要将 connection 加载到内存，占用较多的系统资源

（2）每一次数据库连接使用后都得断开

（3）不能控制被创建的连接对象数

（二）数据库连接池技术

1.基本思想

（1）建立缓冲池存储连接重复使用

（2）最小连接数与最大连接数制约连接数量

2.优点：

（1）资源重用——避免重复创建

（2）更快的系统反应速度——已初始化过

（3）新的资源分配手段——避免独占数据库

（4）统一的连接管理，避免数据库连接泄露

第五节 操作 BLOB 类型字段

一、概述

（一）BLOB 概念

1.BLOB 是一个二进制的大型对象，可以存储大量数据

2.插入 BLOB 类型的数据必须使用 PreparedStatement（Druid 中使用了 PreparedStatement）6

（二）BLOB 分类

1.TinyBlob 最大 255 字节

2.Blob 最大 65K

3.MediumBlob 最大 16M

4.LongBlob 最大 4G

二、对数据表中的 Blob 类型字段进行增改

```
//获取连接
Connection conn = JDBCUtils.getConnection();

String sql = "insert into
customers(name,email,birth,photo)values(?,?,?,?)";
PreparedStatement ps = conn.prepareStatement(sql);

// 填充占位符
ps.setString(1, "徐海强");
ps.setString(2, "xhq@126.com");
ps.setDate(3, new Date(new java.util.Date().getTime()));
// 操作Blob类型的变量
FileInputStream fis = new FileInputStream("xhq.png");
ps.setBlob(4, fis);
//执行
ps.execute();

fis.close();
JDBCUtils.closeResource(conn, ps);
```

Durid 有 createBlob（）方法

三、对数据表中的 Blob 类型字段进行查询

```
String sql = "SELECT id, name, email, birth, photo FROM customer WHERE id =
?";
conn = getConnection();
ps = conn.prepareStatement(sql);
ps.setInt(1, 8);
rs = ps.executeQuery();
if(rs.next()){
  Integer id = rs.getInt(1);
    String name = rs.getString(2);
  String email = rs.getString(3);
    Date birth = rs.getDate(4);
  Customer cust = new Customer(id, name, email, birth);
    System.out.println(cust);
    //读取Blob类型的字段
  Blob photo = rs.getBlob(5);
  InputStream is = photo.getBinaryStream();
  OutputStream os = new FileOutputStream("c.jpg");
  byte [] buffer = new byte[1024];
  int len = 0;
  while((len = is.read(buffer)) != -1){
    os.write(buffer, 0, len);
  }
    JDBCUtils.closeResource(conn, ps, rs);

  if(is != null){
    is.close();
  }

  if(os !=  null){
    os.close();
  }

}
```

进行了一次 IO 流读入写出

第六节 批量插入

一。理论用法

addBatch(String)：添加需要批量处理的 SQL 语句或是参数；

executeBatch()：执行批量处理语句；

clearBatch():清空缓存的数据;

二、各类用法

（一）PreparedStatement（for 循环）

```
long start = System.currentTimeMillis();

Connection conn = JDBCUtils.getConnection();

String sql = "insert into goods(name)values(?)";
PreparedStatement ps = conn.prepareStatement(sql);
for(int i = 1;i <= 20000;i++){
  ps.setString(1, "name_" + i);
  ps.executeUpdate();
}

long end = System.currentTimeMillis();
System.out.println("花费的时间为: " + (end - start));//82340


JDBCUtils.closeResource(conn, ps);
```

（二）PreparedStatement（Batch）

```
//1.设置为不自动提交数据
conn.setAutoCommit(false);

String sql = "insert into goods(name)values(?)";
PreparedStatement ps = conn.prepareStatement(sql);

for(int i = 1;i <= 1000000;i++){
  ps.setString(1, "name_" + i);

  //1."攒"sql
  ps.addBatch();

  if(i % 500 == 0){
    //2.执行
    ps.executeBatch();
    //3.清空
    ps.clearBatch();
  }
}

//2.提交数据
conn.commit();
```

（三）Druid 用法

需要调用 Druid 接口中的 PreparedStatement 的方法，然后实现上面的操作