知识点列表

编号	名称	描述	级别
1	封装获得连接的方式	通过 java io 读入操作系统文件，将获得连接的过程封装在一个类中。	**
2	数据库的元数据	了解概念和元数据的获取方式	*
3	JDBC 中的事务	JDBC 中的事务处理特点和编程实现	**
4	JDBC 中的批处理	JDBC 中的批处理实现方式	**
5	分页策略 – 基于缓存的分页	通过设置 Statement 对象的参数，使结果集的指针可滚动，从而实现基于缓存的分页策略。	***

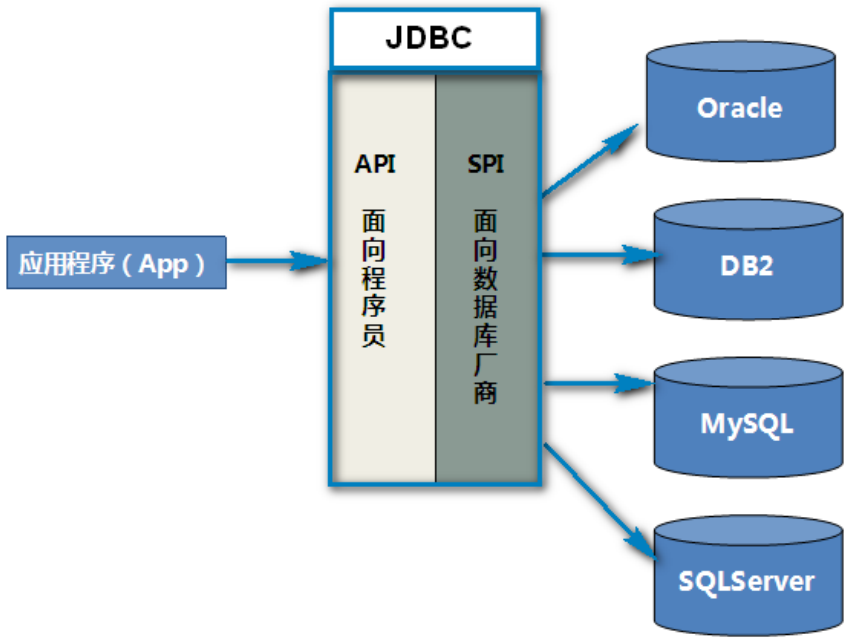
注： "*"理解级别 "**"掌握级别 "***"应用级别

目录

1. JDBC 原理

在没有 JDBC 之前，程序员是自己写驱动连接不同的数据库，这对程序员要求很高，既要求程序员熟练掌握开发语言，还要了解需要连接的数据库，有了 JDBC 后，程序员不需要了解连接各个数据库的底层实现(由各个数据库厂家实现了)，只要掌握这套连接数据库的接口即可。

我们可以用图示表示如下：



2011-12-14

2. JDBC API 的操作过程

1) 获得连接

```
Connection conn = DriverManager.getConnection( url , dbUser , dbPwd ) ;
//url 的格式 : jdbc : oracle : thin : @ip : port : sid
```

2) 构造语句对象，传递 sql 语句，返回结果

- ✓ **第 1 种方式 : Statement**

- **进行 DQL 操作**

```
Statement stmt = conn.createStatement() ;
ResultSet rs = stmt.executeQuery(sql) ;   // rs 指针的初始位置在第一行之前
while(rs.next()){
      rs.getInt("id") ;              //id 是数字 number
      rs.getString("name") ;   //name 是字符串 char(20)
      rs.getDouble("salary") ; //salary 是带小数的数字 number(7,2)
      String bonus = rs.getString("bonus") ;
}
```

- **进行 DML 操作**

```
Statement stmt = conn.createStatement() ;
int n = stmt.executeUpdate(sql) ;   //n 表示 sql 语句( DML )影响的记录数
```

✓ **第 2 种方式：PreparedStatement**

```
String sql = "select * from emp where id = ?" ;
PreparedStatement preStmt = conn.prepareStatement(sql) ;
preStmt.setInt(1 , id) ;
preStmt.executeQuery()  或  preStmt.executeUpdate() ;
```

3) 关闭资源

ResultSet： close()
Statement: close()
Connection: close()

4) 连接池(Connection Pool)

如果应用程序的资源没有及时关闭，会造成资源的浪费等问题，使用连接池可以提高资源使用的效率。

连接池与服务器端会建立一定数量的“连接”，当有客户端连接时，连接池将“连接”分配给各个客户端，一旦客户端任务完成，连接池与客户端的“连接”会断开，等待其他客户端连接。

连接池相当于连接的缓存池。

3. 数据库的元数据(MataData) *

3.1.【案例 1】工具类 ConnectionUtils.java **

希望把对数据库的公共操作提取出来，代码可以重用。

1)获取参数方法　　getParam(String filename)

2)获得连接方法　　getConnection()

3)关闭资源方法　　close(Connection con)

close(Statement sta)

close(ResultSet rst)

- **参考代码**

ConnectionUtils.java

```
1  package day02;
2  import java.io.*;
5  public class ConnectionUtils {
6    private static String url ;
7    private static String dbUser ;
8    private static String dbPassword ;
9
10   /**
11    * 读入filename指定的文件并解析，取出键值对中的数据
12    * 给全局变量url, dbUser, dbPassword赋值
13    *
14    * @param filename : 文件名
15    */
16   public static void getParam(String filename){
17     Properties propes = new Properties();
18
19     File file = new File(filename);
20     try {
21         FileInputStream fis =
22             new FileInputStream(file);
23         //加载输入流指定的文件，数据放入键值对对象
24         propes.load(fis);
25         //获取文件中key对应的value，给全局变量赋值
26         url = propes.getProperty("url");
27         dbUser = propes.getProperty("dbUser");
28         dbPassword = propes.getProperty("dbPassword");
29     } catch (FileNotFoundException e) {
30         e.printStackTrace();
31     } catch (IOException e) {
32         e.printStackTrace();
33     }
34   }
35
```

```
36  /**
37   * 利用getParam方法获得的参数，构造连接并返回
38   * @return 连接对象
39   */
40  public static Connection getConnection(){
41    getParam("src/db_oracle.properties");
42    Connection conn = null;
43    try {
44        conn = DriverManager
45            .getConnection(url,dbUser,dbPassword);
46    } catch (SQLException e) {
47        e.printStackTrace();
48    }
49    return conn;
50  }
51
52  /**
53   * 关闭连接
54   * @param conn
55   */
56  public static void close(Connection conn){
57    if (conn != null){
58        try {
59            conn.close();
60        } catch (SQLException e) {
61            e.printStackTrace();
62        }
63    }
64  }
65
66  /**
67   * 关闭语句对象
68   * @param stmt
69   */
70  public static void close(Statement stmt){
71    if (stmt != null){
72        try {
73            stmt.close();
74        } catch (SQLException e) {
75            e.printStackTrace();
76        }
77    }
78  }
79
```

```
80   /**
81    * 关闭结果集
82    * @param rs
83    */
84   public static void close(ResultSet rs){
85     if (rs != null){
86         try {
87             rs.close();
88         } catch (SQLException e) {
89             e.printStackTrace();
90         }
91     }
92   }
93 }
94
```

3.2.【案例 2】获得数据库和数据表的元数据

- **问题提出：**

  通过连接对象获得数据库的资料，或者通过结果集对象获得数据表的资料。

- **参考代码**

GetTableData.java

```
1  package day02;
2  import java.sql.*;;
3  public class GetTableData {
4    public static void main(String[] args) {
5        //getData("emp");
6        getData("user_xxx");
7    }
8
9    public static void getData(String tablename){
10
11     String sql = "select * from " + tablename;
12     Connection conn = ConnectionUtils.getConnection();
13     Statement stmt = null;
14     ResultSet rs = null;
```

```
15
16      try {
17          /* 演示1 */
18          //数据库的元数据
19          DatabaseMetaData dmd = conn.getMetaData();
20          //数据库名
21          System.out.println(
22              dmd.getDatabaseProductName());
23          //数据库的版本号
24          System.out.println(
25              dmd.getDatabaseMajorVersion());
26          //连接字符串
27          System.out.println(dmd.getURL());
28          //用户名
29          System.out.println(dmd.getUserName());
30
31          /* 演示2 */
32          //结果集元数据
33          stmt = conn.createStatement();
34          rs = stmt.executeQuery(sql);
35          ResultSetMetaData rsmd =
36              rs.getMetaData(); //获取结果集的元数据
37          int columnCount =
38              rsmd.getColumnCount(); //获取列数
39
40          for(int i = 1; i <= columnCount; i++){
41              System.out.print(
42                      rsmd.getColumnName(i) + "     ");
43          }
44          System.out.println();
45          System.out.println("-------------------------")
46
47          while(rs.next()){
48              //注意：JDBC的计数从1开始
49              for (int i = 1; i <= columnCount; i++){
```

```
50                  String value
51                      = rs.getString(
52                              rsmd.getColumnName(i));
53                  System.out.print(value + "\t");
54              }
55              System.out.println();
56          }
57
58      } catch (SQLException e) {
59          e.printStackTrace();
60      } finally{
61          ConnectionUtils.close(rs);
62          ConnectionUtils.close(stmt);
63          ConnectionUtils.close(conn);
64      }
65  }
66 }
```

- **运行结果**

```
Console  SQL Results
<terminated> GetTableData [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Dec 14, 2011
Oracle
10
jdbc:oracle:thin:@192.168.0.26:1521:tarena
OPENLAB
ID     PASSWORD     NAME     PHONE
------------------------
1001     1234     lisi     5678
```

✓ **注意：JDBC 计数从 1 开始**

✓ **知识点：结果集(ResultSet)中可以用列名取值，也可以用列序取值**

```
while(rs.next()){
    rs.getInt("empno") ;      //用列名取值
    rs.getString(2) ;         //用列序取值
}
```

3.3. MetaData(元数据，关于数据的数据)

1) **DatabaseMetaData** 数据库的元数据
   - ✓ 正在连接的数据库的信息，从连接对象(Connection)获取
2) **ResultSetMetaData** 数据结果集的元数据
   - ✓ 和查询出来的结果集相关，从结果集(ResultSet)获取

3.4.【案例 3】关联查询

- **数据准备**

```
--##########Dept表############

drop table dept_xxx;

create table dept_xxx(
  deptno    number(2),
  dname     char(20),
  location  char(20),

  constraint dept_xxx_deptno_pk primary key (deptno)
);

insert into dept_xxx values(10 , 'developer' , 'beijing');
insert into dept_xxx values(20, 'account' , 'shanghai');
insert into dept_xxx values(30, 'sales' , 'guangzhou');
insert into dept_xxx values(40, 'operations', 'tianjin');
commit;

select * from dept_xxx;


--#########Emp表###############

drop table emp_xxx;

create table emp_xxx(
  empno number(4),
  ename varchar2(20),
  job varchar2(15),
  salary number(7,2),
  bonus number(7,2),
  hiredate date,
  mgr number(4),
  deptno number(10),

  constraint emp_xxx_empno_pk primary key (empno),
  constraint emp_deptno_fk foreign key (deptno) references dept_xxx(deptno)
);

```

```
insert into emp_xxx
  values(1001, '张无忌', 'Manager', 10000, 2000, '12-MAR-10', 1005, 10);
insert into emp_xxx
  values(1002, '刘苍松', 'Analyst', 8000, 1000,  '01-APR-11', 1001, 10);
insert into emp_xxx
  values(1003, '李翊', 'Analyst', 9000, 1000,  '11-APR-10', 1001, 10);
insert into emp_xxx
  values(1004, '郭芙蓉', 'Programmer', 5000, null, '01-JAN-11', 1001, 10);
insert into emp_xxx
  values(1005, '张三丰', 'President', 15000, null, '15-MAY-08', null, 20);
insert into emp_xxx
  values(1006, '燕小六','Manager', 5000, 400, '01-FEB-09', 1005, 20);
insert into emp_xxx
  values(1007, '陆无双','clerk', 3000, 500, '01-FEB-09', 1006, 20);
insert into emp_xxx
  values(1008, '黄蓉','Manager', 5000, 500, '1-MAY-09', 1005, 30);
insert into emp_xxx
  values(1009, '韦小宝','salesman', 4000, null, '20-FEB-09', 1008, 30);
insert into emp_xxx
  values(1010, '郭靖','salesman', 4500, 500, '10-MAY-09', 1008, 30);

select * from emp_xxx;

--#########Salgrade表###############

drop table salgrade_xxx;

create table salgrade_xxx(
  grade number(2),
  lowsal number(7,2),
  hisal number(7,2)
);

insert into salgrade_xxx values(1,10001,99999);
insert into salgrade_xxx values(2,8001,10000);
insert into salgrade_xxx values(3,6001,8000);
insert into salgrade_xxx values(4,4001,6000);
insert into salgrade_xxx values(5,1,4000);

commit;

select * from salgrade_xxx;
```

- **SQL 语句：获得员工的姓名、部门名、薪水等级**

```
-- 三个表关联
select e.ename, d.dname, s.grade
from emp_xxx e
join dept_xxx d
      on e.deptno = d.deptno
join salgrade s
      on e.salary between s.lowsal and s.hisal ;
```

Result 1

ENAME	DNAME	GRADE
陆无双	account	1
韦小宝	sales	1
郭靖	sales	2
郭芙蓉	developer	2
燕小六	account	2
黄蓉	sales	2
刘苍松	developer	3
李翊	developer	4
张无忌	developer	4
张三丰	account	5

- **参考代码(代码片段)**

```
 9  public static void getData(String tablename){
10
11    String sql = "select e.ename, d.dname, s.grade " +
12        " from ninglj.emp_ning e  " +
13        " join ninglj.dept_ning d " +
14        " on e.deptno = d.deptno " +
15        " join ninglj.salgrade s  " +
16        " on e.salary between s.lowsal and s.hisal";
17    Connection conn = ConnectionUtils.getConnection();
18    Statement stmt = null;
19    ResultSet rs = null;
20
```

- **运行结果**

Console | SQL Results

```
<terminated> GetTableData [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Dec 14, 2011
Oracle
10
jdbc:oracle:thin:@192.168.0.26:1521:tarena
OPENLAB
ENAME     DNAME     GRADE
------------------------
李翊      ACCOUNTING          1
张无忌    ACCOUNTING          1
刘苍松    ACCOUNTING          2
```

```
郭芙蓉    ACCOUNTING          3
tom       ACCOUNTING          5
张三丰    RESEARCH            1
燕小六    RESEARCH            4
陆无双    RESEARCH            5
黄蓉      SALES               4
郭靖      SALES               4
韦小宝    SALES               4
韦小宝    SALES               5
```

4. JDBC 中的事务 **

1) JDBC 默认事务是自动提交的，也就是每个 DML 操作完成后都自动提交。
可以改为手动提交，如下所示：

```
conn.setAutoCommit(false);  //ture 自动提交
```

2) Sqlplus 中默认事务是非自动提交，默认必须显式或隐式提交
可以使用如下语句设置：

```
SQL> set autocommit on      --打开自动提交
SQL> set autocommit off     --关闭自动提交，默认
```

在一些需要事务的场合(比如工资提取和记录日志两个操作都完成后才提交)，就需要程序员自己控制事务。

4.1.【案例 4】对 emp.salary 字段的修改要记日志

- **数据准备**

```
        -- 日志表
SQL>   create table logs_xxx(
         id number(4) primary key ,
         who varchar2(30) ,
         when date default sysdate ,
         what varchar2(50)
       );
       -- 序列( 用于生成日志表的主键 )
SQL>   create sequence myseq_log_xxx start with 1000 increment by 1 ;

SQL>   select * from emp_xxx ;
```

Console | SQL Results

Result 1

EMPNO	ENAME	JOB	SALARY	BONUS	HIREDATE	MGR	DEPTNO
1001	张无忌	Manager	10000	2000	2010-03-12 00:00:00.0	1005	10
1002	刘苍松	Analyst	8000	1000	2011-04-01 00:00:00.0	1001	10
1003	李翊	Analyst	9000	1000	2010-04-11 00:00:00.0	1001	10
1004	郭芙蓉	Programmer	5000	<NULL>	2011-01-01 00:00:00.0	1001	10
1005	张三丰	President	15000	<NULL>	2008-05-15 00:00:00.0	<NULL>	20
1006	燕小六	Manager	5000	400	2009-02-01 00:00:00.0	1005	20
1007	陆无双	clerk	3000	500	2009-02-01 00:00:00.0	1006	20
1008	黄蓉	Manager	5000	500	2009-05-01 00:00:00.0	1005	30
1009	韦小宝	salesman	4000	<NULL>	2009-02-20 00:00:00.0	1008	30
1010	郭靖	salesman	4500	500	2009-05-10 00:00:00.0	1008	30

- **参考代码，分别测试不同的情况：**

  测试 1：员工薪水被成功修改，记录一条日志，即两条 DML 成功则提交。

  测试 2：员工薪水修改失败，但不是异常，日志表新增的数据被回滚

  测试 3：员工薪水修改正常，但日志新增失败，出现异常，则员工薪水的修改被取消，回滚。

  1) **测试 1：修改员工 1001 的薪水为 20000** --两条 sql 语句都能正确执行

TransactionDemo.java | emp.sql

```
1  package day02;
2  import java.sql.*;
3  /**
4   * 测试JDBC处理事务的功能
5   * @author teacher
6   */
7  public class TransactionDemo {
8
9      public static void main(String[] args) {
10         try{
11             //测试1: 修改empno=1001的员工薪水为20000
12             updateSalary(1001, 20000);
13         }catch(SQLException e){
14             e.printStackTrace();
15         }
16     }
17
18   /**
19    * 修改员工表中某个员工的薪水为指定值,同时记录日志
20    */
21   public static void updateSalary(
22                           int empno, double salary)
23     throws SQLException{
24
25     String sql1 = "update emp_xxx set salary = "
```

```
26          + salary + " where empno = " + empno;
27
28      String sql2 = "insert into logs_xxx " +
29              "values(myseq_log_xxx.nextval, " +
30              "user, sysdate, 'update salary')";
31
32      Connection conn =
33          ConnectionUtils.getConnection();
34      Statement stmt = null;
35
36      try{
37          //1.把JDBC的自动提交关闭
38          conn.setAutoCommit(false);
39          stmt = conn.createStatement();
40
41          //2.执行两条sql语句
42          int n1 = stmt.executeUpdate(sql1);
43          System.out.println("n1 = " + n1);
44          int n2 = stmt.executeUpdate(sql2);
45          System.out.println("n2 = " + n2);
46
47          //3.如果都执行成功了，提交，否则回退
48          if (n1 == 1 && n2 == 1){
49              conn.commit();
50          }
51          else{
52              conn.rollback();
53          }
54
55          //4.恢复默认值
56          conn.setAutoCommit(true);
57
58      }catch(Exception e){
59          System.out.println(
60                  "SQL语句出现了异常！");
61          conn.rollback();
62          e.printStackTrace();
63      }finally{
64          ConnectionUtils.close(stmt);
65          ConnectionUtils.close(conn);
66      }
67    }
68  }
```

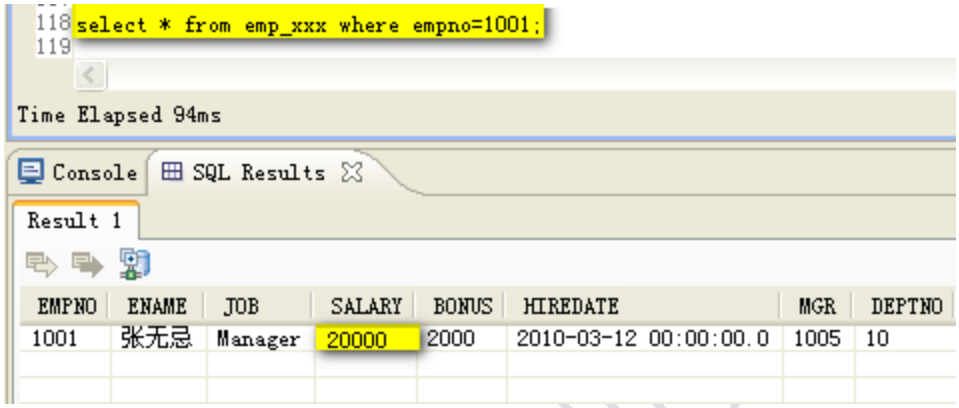
Console SQL Results

```
<terminated> TransactionDemo [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Dec 15, 2
n1 = 1
n2 = 1
```

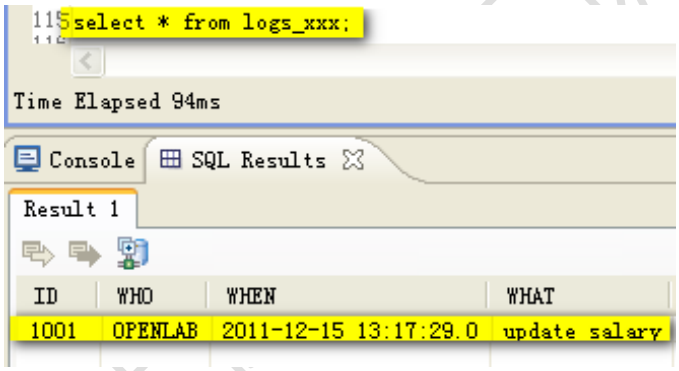
- ✓ n1 = 1： 薪水被成功修改
- ✓ n2 = 1： 成功插入一条日志记录
- ✓ 如果 sql1 和 sql2 都执行成功，事务 commit

结果演示

- Emp_xxx 表数据修改了



- Logs_xxx 表同时记录一条日志



2) 测试 2：修改编号为 2000 的员工工资(该员工不存在) --修改数据没执行成功

TransactionDemo.java

```
1 package day02;
2 import java.sql.*;
3 /**
4  * 测试JDBC处理事务的功能
5  * @author teacher
6  */
7 public class TransactionDemo {
8
```

```
    public static void main(String[] args) {
        try{
            //测试2：修改empno=1020的员工薪水为20000
            updateSalary(1020, 20000);
        }catch(SQLException e){
            e.printStackTrace();
        }
    }

    * 修改员工表中某个员工的薪水为指定值,同时记录日志
    public static void updateSalary(
                        int empno, double salary)
    throws SQLException{

    String sql1 = "update emp_xxx set salary = "
        + salary + " where empno = " + empno;

    String sql2 = "insert into logs_xxx " +
            "values(myseq_log_xxx.nextval, " +
            "user, sysdate, 'update salary')";

    Connection conn =
        ConnectionUtils.getConnection();
    Statement stmt = null;

    try{
        //1.把JDBC的自动提交关闭
        conn.setAutoCommit(false);
        stmt = conn.createStatement();

        //2.执行两条sql语句
        int n1 = stmt.executeUpdate(sql1);
        System.out.println("n1 = " + n1);
        int n2 = stmt.executeUpdate(sql2);
        System.out.println("n2 = " + n2);

        //3.如果都执行成功了，提交，否则回退
        if (n1 == 1 && n2 == 1){
            conn.commit();
        }
        else{
            conn.rollback();
        }

```

```
55          //4.恢复默认值
56          conn.setAutoCommit(true);
57
58      }catch(Exception e){
59          System.out.println(
60                  "SQL语句出现了异常！");
61          conn.rollback();
62          e.printStackTrace();
63      }finally{
64          ConnectionUtils.close(stmt);
65          ConnectionUtils.close(conn);
66      }
67    }
68 }
```

- ✓ n1 = 0：薪水没有被修改
- ✓ n2 = 1：插入一条日志记录
- ✓ sql1 没有执行成功，事务 rollback

3) 测试 3：插入数据成功，记录日志时出现异常

TransactionDemo.java | emp.sql

```
19     * 修改员工表中某个员工的薪水为指定值,同时记录日志
20     */
21    public static void updateSalary(
22                              int empno, double salary)
23      throws SQLException{
24
25      String sql1 = "update emp_xxx set salary = "
26          + salary + " where empno = " + empno;
27
28      String sql2 = "insert into logs_xxx " +
29              "values(1001, " +
30              "user, sysdate, 'update salary')";
31
32      Connection conn =
33          ConnectionUtils.getConnection();
34      Statement stmt = null;
35
36      try{
37          //1.把JDBC的自动提交关闭
38          conn.setAutoCommit(false);
39          stmt = conn.createStatement();
40
```

```
41          //2.执行两条sql语句
42          int n1 = stmt.executeUpdate(sql1);
43          System.out.println("n1 = " + n1);
44          int n2 = stmt.executeUpdate(sql2);
45          System.out.println("n2 = " + n2);
46
47          //3.如果都执行成功了，提交，否则回退
48          if (n1 == 1 && n2 == 1){
49              conn.commit();
50          }
51          else{
52              conn.rollback();
53          }
54
55          //4.恢复默认值
56          conn.setAutoCommit(true);
57
58      }catch(Exception e){
59          System.out.println(
60                  "SQL语句出现了异常！");
61          conn.rollback();
62          e.printStackTrace();
63      }finally{
64          ConnectionUtils.close(stmt);
65          ConnectionUtils.close(conn);
66      }
67    }
68  }
```

```
Console  SQL Results
<terminated> TransactionDemo [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Dec 15, 2
n1 = 1
SQL语句出现了异常！
java.sql.SQLException: ORA-00001: 违反唯一约束条件 (OPENLAB

        at oracle.jdbc.driver.SQLStateMapping.newSQLExcept
        at oracle.jdbc.driver.DatabaseError.newSQLExceptio
```

- ✓ n1 = 1： 修改薪水成功
- ✓ 因为日志表主键(pk)冲突，插入数据时发生异常
- ✓ 出现异常，事务 rollback

5. 批处理(Batch) **

5.1.【案例 5】批处理演示

- **准备数据**

```
create table mytemp_xxx(id number primary key) ;
```

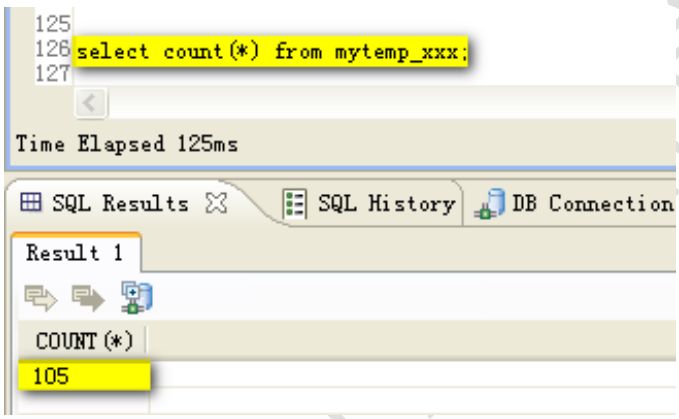
- **代码参考**

BatchDemo.java

```
1  package day02;
2  import java.sql.*;
3  /**
4   * 批处理练习
5   * @author teacher
6   */
7  public class BatchDemo {
8    public static void main(String[] args) {
9      executeSql();
10   }
11
12   public static void executeSql(){
13     String sql = "insert into mytemp_xxx(id) " +
14             "values(?)";
15     Connection conn =
16         ConnectionUtils.getConnection();
17     PreparedStatement stmt = null;
18
19     try {
20         conn.setAutoCommit(false);
21
22         stmt = conn.prepareStatement(sql);
23         for(int i = 1; i <= 105; i++){
24             stmt.setInt(1, i);
25             //1. 把sql语句加入批处理
26             stmt.addBatch();
27             //2. 每10条记录处理一次
28             //   避免批处理中的sql语句太多
29             if(i % 10 == 0){
30                 //3. 执行
31                 stmt.executeBatch();
32                 //4. 清除
33                 stmt.clearBatch();
34             }
35         }
36         //执行最后的5句Sql
37         stmt.executeBatch();
38
39         conn.commit();
```

```
40
41      } catch (SQLException e) {
42          try {
43              conn.rollback();
44          } catch (SQLException e1) {
45              e1.printStackTrace();
46          }
47          e.printStackTrace();
48      } finally{
49          ConnectionUtils.close(stmt);
50          ConnectionUtils.close(conn);
51      }
52    }
53 }
```

- **结果演示**



6. 分页的两种策略

6.1. 基于缓存的分页策略 ***

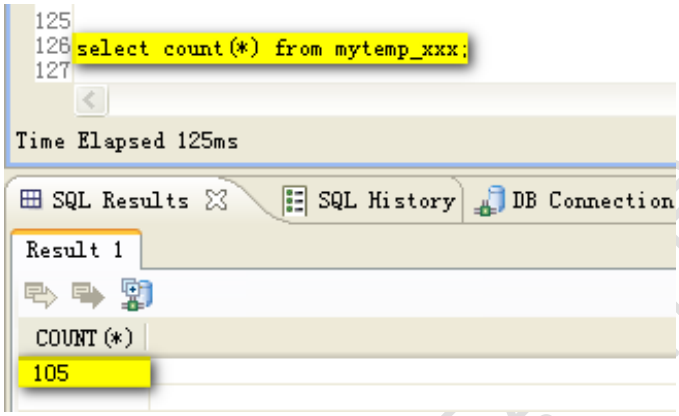
1) 一次性把数据全部取出来放在缓存中，根据用户要看的页数(page)和每页记录数(pageSize)，计算把哪些数据输出显示。
2) 假设每页 10 条(pageSize = 10)
   第 1 页： 1-10
   第 2 页： 11-20
   ....
   第 n 页： [起点] (n-1) * pageSize + 1 -- [终点] 起点 + pageSize - 1
3) **特点**
   ✓ 只访问数据库一次，第一次取数比较慢，以后每页都从缓存中取，比较快

- ✓ 比较适合小数据量，如果数据量大，对内存压力比较大。
- ✓ 一次性将数据库数据读入结果集，每次查看指定的页时，要求结果集的指针能够跳到指定的的行，即指针能够跳到整个结果集的任一位置。

【案例 6】基于缓存的分页策略演示

- **数据准备**

表 mytemp_xxx 中存放数字 1-105 共 105 条数据



- **参考代码**

```
1  package day02;
2  import java.sql.*;
3  /**
4   * 基于缓存的分页策略
5   * @author teacher
6   */
7  public class BufferPageDemo {
8    public static void main(String[] args) {
9      //从第3页开始，打印10条记录
10     getPage(10, 3);  // 21-30
11   }
12
```

```
13  /**
14   * 打印指定页的数据,用mytemp_xxx表做测试
15   *
16   * begin: 在结果集中第page页的记录起点
17   * begin = (page - 1) * pageSize + 1;
18   *
19   * @param pageSize 每页多少条记录
20   * @param page 第几页
21   */
22  public static void getPage(int pageSize, int page){
23    //在结果集中，第page页的记录起点
24    int begin = (page - 1) * pageSize + 1;
25
26    String sql = "select * from mytemp_xxx";
27    Connection conn = ConnectionUtils.getConnection();
28    Statement stmt = null;
29    ResultSet rs = null;
30
31    try {
32        //约定结果集的类型和并发性
33        stmt = conn.createStatement(
34            //1) 设置类型为可滚动的结果集(可跳步的)
35            ResultSet.TYPE_SCROLL_INSENSITIVE,
36            //2) 设置并发性为其他用户只读
37            ResultSet.CONCUR_READ_ONLY);
38
39        //结果集为可滚动的，可进行分页操作
40        rs = stmt.executeQuery(sql);
41
42        /* 演示1 */
43        //1) 定位到绝对位置
44        rs.absolute(85);    //85
45        System.out.println(rs.getInt("id"));
46        //2) 定位到绝对位置
47        rs.relative(5);    //90
48        System.out.println(rs.getInt("id"));
49        //3) 下一条
50        rs.next();         //91
51        System.out.println(rs.getInt("id"));
52        //4) 前一条
53        rs.previous();    //90
54        System.out.println(rs.getInt("id"));
55        //5) 异常：结果集耗尽
56        rs.absolute(106);  //共105条数据
57  //      System.out.println(rs.getInt("id"));
58
59        System.out.println("####################");
60
```

```
61          /* 演示2 */
62          //指针跳到结果集中的起点begin
63          rs.absolute(begin);
64          //循环pageSize次，取数据并打印
65          for (int i = 0; i < pageSize; i++){
66              System.out.println(rs.getInt("id"));
67              if (!rs.next()){
68                  break;
69              }
70          }
71
72      } catch (SQLException e) {
73          e.printStackTrace();
74      } finally{
75          ConnectionUtils.close(rs);
76          ConnectionUtils.close(stmt);
77          ConnectionUtils.close(conn);
78      }
79    }
80 }
81
```

注意：其中的演示 1 只是测试结果集中指针的位置，真正对于分页生效的是演示 2 的内容。

6.2. 基于查询的分页策略

1) 每次只像数据库要求一页的数据量
2) 假设每页 10 条(pageSize = 10)

   第 1 页 : 1-10

   第 2 页 : 11-20

   ....

   第 n 页 : [起点] (n - 1) * pageSize + 1 --- [终点] 起点 + pageSize - 1
3) 特点
   - ✓ 频繁的数据库访问。每次取数据的时间都差不多
   - ✓ 比较适合大数据量
   - ✓ 对内存压力小。