JDBC – some remarks

```java
public Employee getEmployeeById(long id) {
    Connection conn=null;
    PreparedStatement stmt=null;
    ResultSet rs=null;
    try {
        conn = dataSource.getConnection();
        stmt = conn.prepareStatement(
                "select id, firstname, lastname, salary "
              + " from employee where id=?");
        stmt.setLong(1,id);
        rs = stmt.executeQuery();
        if (rs.next()){
            Employee employee = new Employee();
            employee.setId(rs.getLong("id"));
            employee.setFirstName(rs.getString("firstname"));
            employee.setLastName(rs.getString("lastname"));
            employee.setSalary(rs.getBigDecimal("salary"));
        }
        return employee;
    } catch (SQLException e) {
        // Exception handling
    }
```

```
finally {
      if (rs != null) {
          try {
              rs.close();
          } catch (SQLException e){}
       }
       if (stmt != null) {
          try {
              stmt.close();
          } catch (SQLException e){}
       }
      if(conn != null) {
         try {
            conn.close();
         } catch(SQLException e) {}
      }
   }
   return null;
}
```

Statement vs. PreparedStatement

```
stmt = conn.prepareStatement(
              "select id, firstname, lastname, salary "
            + " from employee where id=?");
stmt.setLong(1,id);
rs = stmt.executeQuery();
```

Versus

```
stmt = conn.createStatement();
ts = stmt.executeQuery(
       "select id, firstname, lastname, salary "
     + " from employee where id=" + "id”);
```

Statement vs. PreparedStatement

- **PreparedStatement** – more efficient when query executed repeatedly (possibly with different parameter values)
  - the server creates the execution plan only once.
  - Not always – depends on implementation

- **Statement** – threat of SQL injection

```
ts = stmt.executeQuery(
        "select id, firstname, lastname, salary "
      + " from employee where id="
      + "3; DELETE FROM empkoyee”);
```

```
finally {
      if (rs != null) {
          try {
              rs.close();
          } catch (SQLException e){}
       }
       if (stmt != null) {    // Does it make any sense for
          try {               // PreparedStatement ??????
              stmt.close();
          } catch (SQLException e){}
       }
      if(conn != null) {
         try {
            conn.close();
         } catch(SQLException e) {}
      }
   }
   return null;
}
```

Closing JDBC resources

- **conn.close()** may not close the connection if the connection has some JDBC resources open. Don't rely on assumption that conn.close() should close them !!!

- => rs.close(); stmt.close(); conn.close()

Comparison of access methods

No of queries	Method	Response in ms
100	Statement	82
	PreparedStatement with closing	84
	PreparedStatement w/o closing	43
1000	Statement	435
	PreparedStatement with closing	271
	PreparedStatement w/o closing	128
10000	Statement	1663
	PreparedStatement with closing	1358
	PreparedStatement w/o closing	1027

Connection and ResultSet

- Don't share connection among multiple threads
- ResultSet rs = stmnt.executeQuery();
  - rs may reference an object owned by the JDBC driver => danger of a deadlock
- Another reason why connection shall not be shared:
  - The transaction is bound to the connection