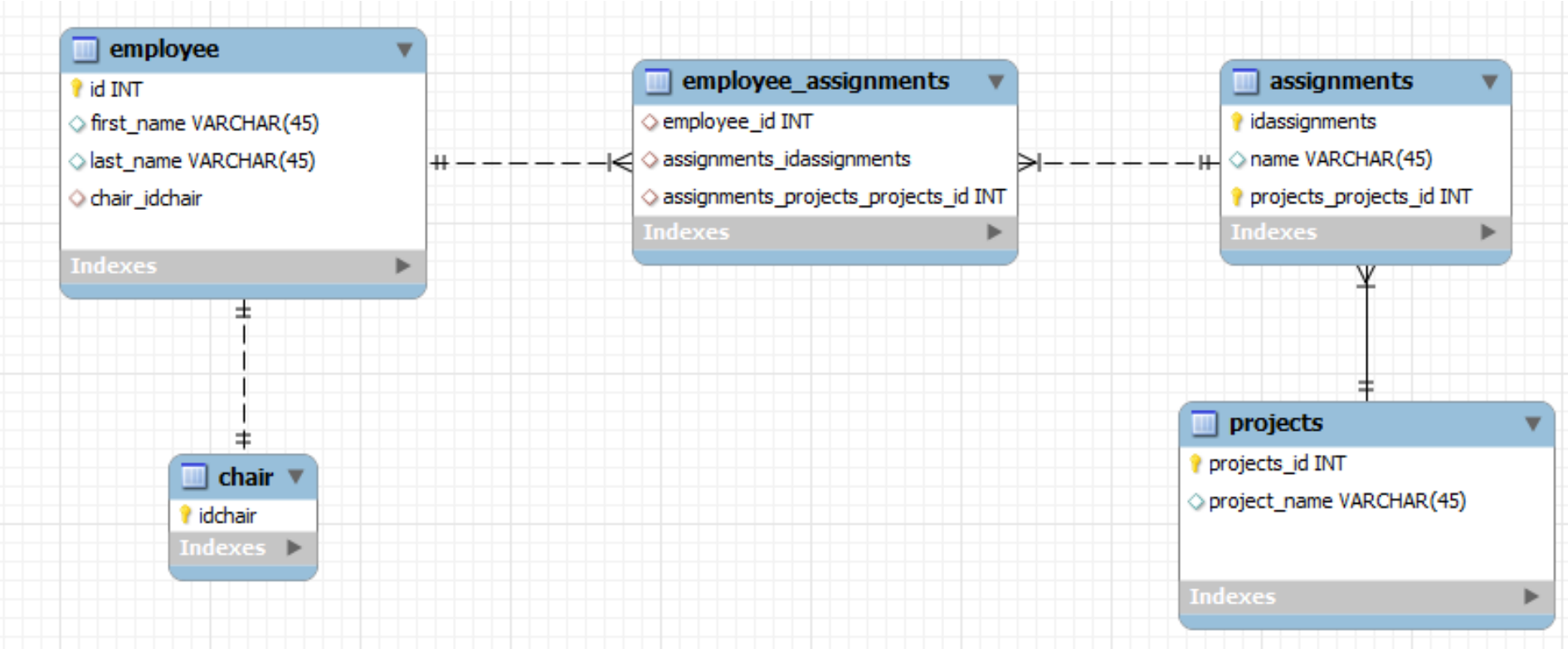
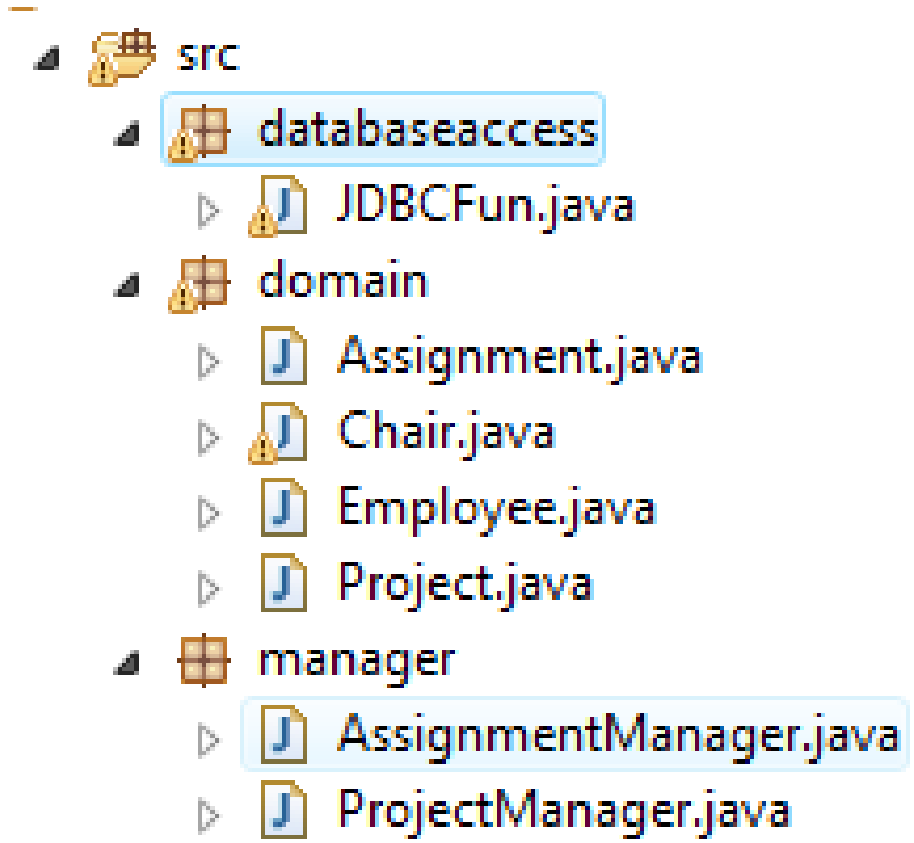
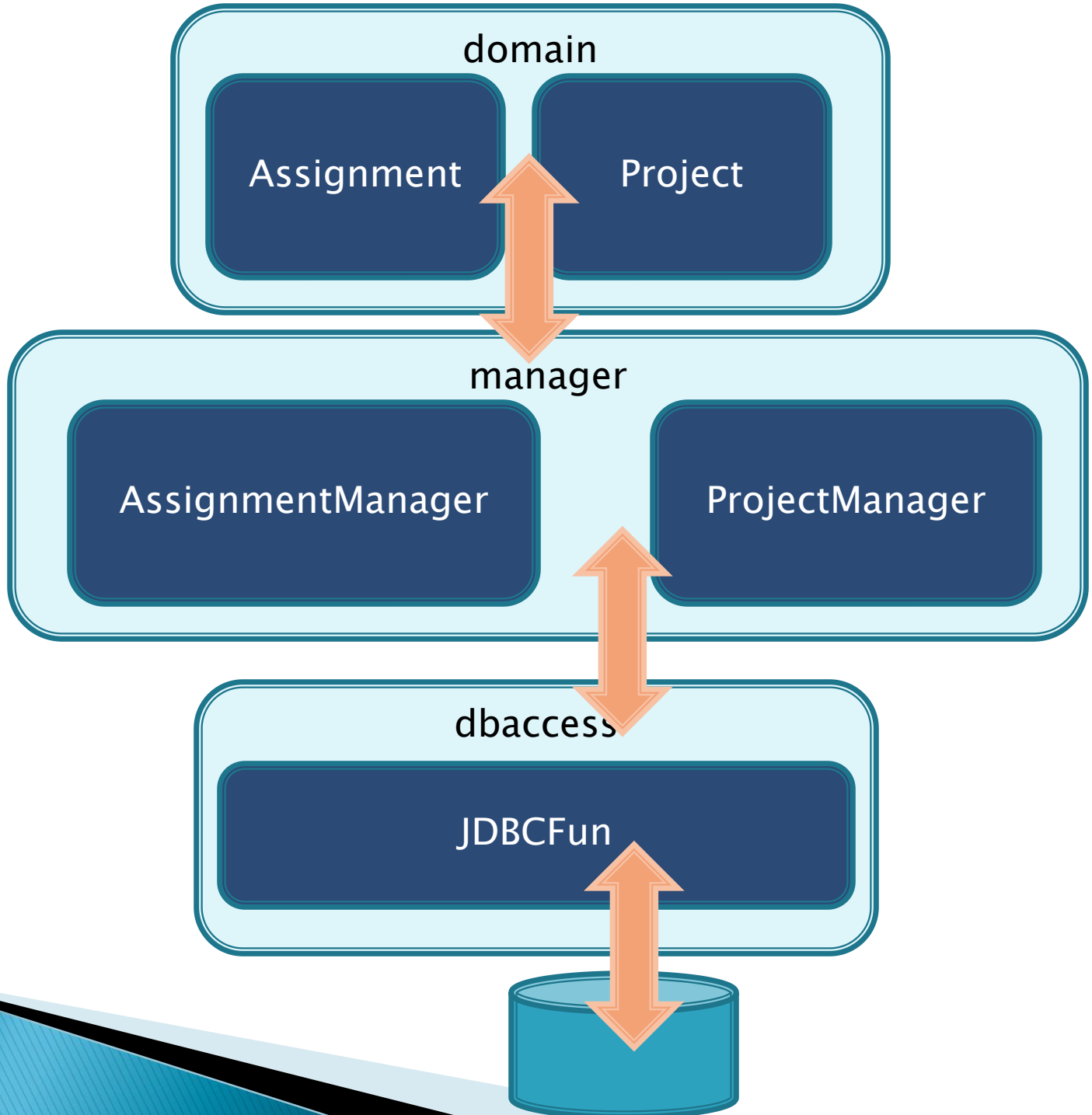
The Data Model



Project Structure




domain
Assignment
Project
manager
AssignmentManager
ProjectManager
dbaccess
JDBCFun

The Employee Class

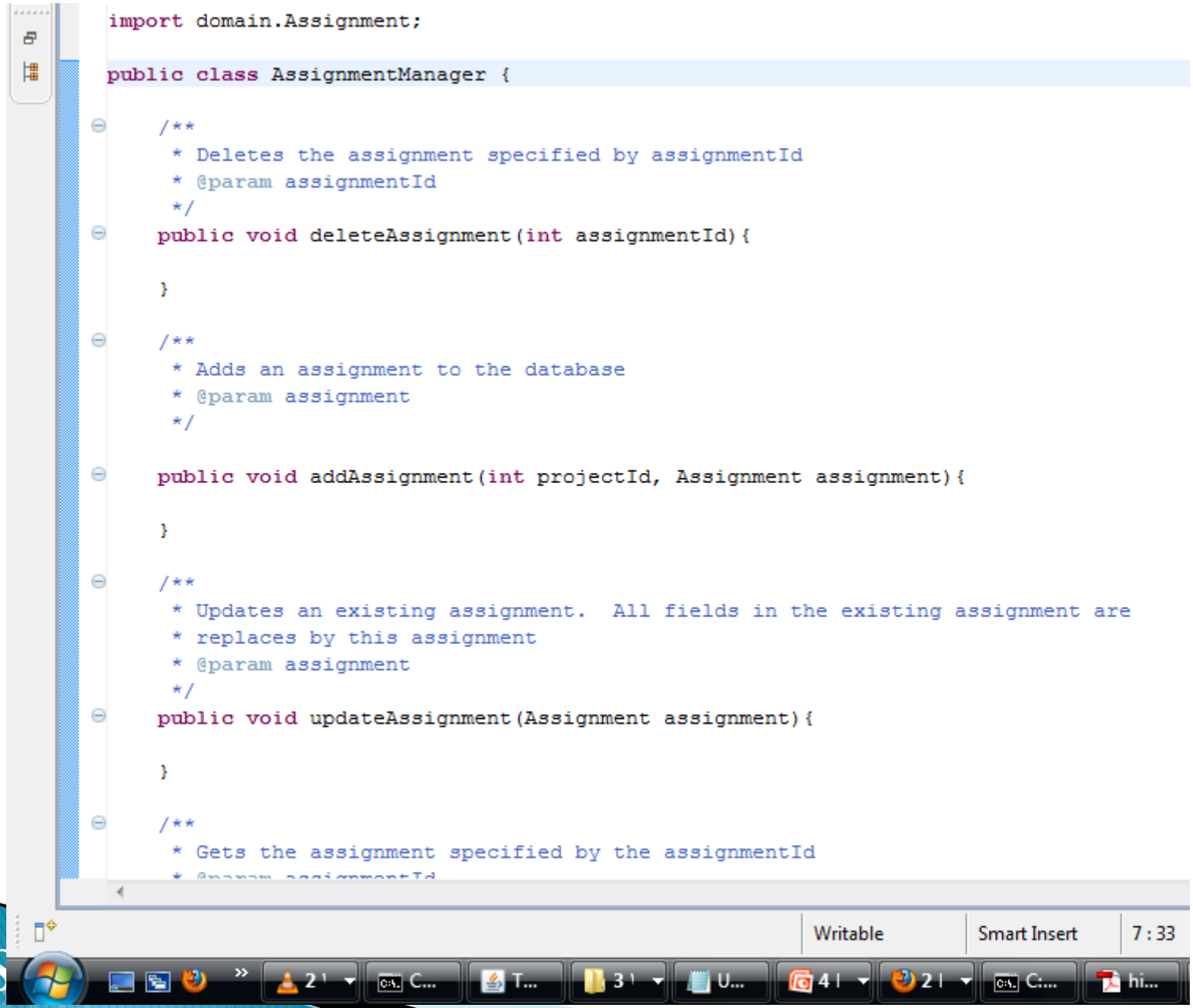
```
public class Employee {
    private int employeeId;
    private Chair chair;
    private List<Assignment> assignments;

    public int getEmployeeId() {
        return employeeId;
    }
    public void setEmployeeId(int employeeId) {
        this.employeeId = employeeId;
    }
    public Chair getChair() {
        return chair;
    }
    public void setChair(Chair chair) {
        this.chair = chair;
    }
    public List<Assignment> getAssignment() {
        return assignments;
    }
    public void setAssignment(List<Assignment> assignment) {
        this.assignments = assignment;
    }

}
```

The Assignment Class

```
public class Assignment {
    private int assignmentId;
    private String name;

    public int getAssignmentId() {
        return assignmentId;
    }
    public void setAssignmentId(int assignmentId) {
        this.assignmentId = assignmentId;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

}
```

The Projects Class

```
public class Project {

    private int projectId;
    private List<Assignment> assignments;

    public int getProjectId() {
        return projectId;
    }
    public void setProjectId(int projectId) {
        this.projectId = projectId;
    }
    public List<Assignment> getAssignments() {
        return assignments;
    }
    public void setAssignments(List<Assignment> assignments) {
        this.assignments = assignments;
    }
}
```


import domain.Assignment;
public class AssignmentManager {
/**
* Deletes the assignment specified by assignmentId
* @param assignmentId
*/
public void deleteAssignment(int assignmentId){
}
/**
* Adds an assignment to the database
* @param assignment
*/
public void addAssignment(int projectId, Assignment assignment){
}
/**
* Updates an existing assignment. All fields in the existing assignment are
* replaces by this assignment
* @param assignment
*/
public void updateAssignment(Assignment assignment){
}
/**
* Gets the assignment specified by the assignmentId
Writable
Smart Insert
7 : 33

EHS
eBuzima

```java
public class ProjectManager {

    /**
     * Deletes the project with the same projectId
     * @param projectId
     */
    public void deleteProject(int projectId){

    }

    /**
     * Updates all the fields of the existing project with the fields in the
     * project object parameter
     * @param project
     */
    public void updateProject(Project project){

    }

    /**
     * Adds a project to the database
     * @param project
     * @param assignment
     */
    public void addProject(Project project){

    }


}
```

Managing database connections

```
/**
 * Creates a connection to the database.
 *
 * @return Connection the connection object.
 */
private Connection getNewConnection() {
    String connectionURL = "jdbc:mysql://localhost:3306/projects";
    Connection connection = null;

    try {
        connection = (Connection) DriverManager.getConnection(
                connectionURL, "devuser", "devpass");

    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return connection;
}

/**
 * Initializes a number of db connections
 * @param sizeOfConnectionPool
 */
public void initializeConnections(int sizeOfConnectionPool){
    int i = 0 ;
    while (i < sizeOfConnectionPool){
        connections.add(getNewConnection());
        i++;
    }
}
```

Managing database connections

```
/**
 *
 * @return
 */
public Connection getAvailableConnection(){
    Connection connectionToReturn = null;

    if(nextIndex == connections.size()){
        nextIndex = 0;
    }

    connectionToReturn = connections.get(nextIndex);

    nextIndex++;

    return connectionToReturn;
}

/**
 * Closes all connections
 */
public void closeConnections(){
    for(Connection conn:connections){
        try {
            if(conn != null){
                conn.close();
            }
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Manager implementation

```
public class ProjectManager {

    public Project getProject(int projectId){
        String sql = "select * from projects where project_id = " + projectId;
        Connection conn = JDBCFun.getInstance().getAvailableConnection();
        StatementResultSetContainer srsc = JDBCFun.getInstance().runQuery(conn, sql);
        ResultSet rs = srsc.getResultSet();
        Project project = null;
        try{
            rs.next();
            project = new Project();
            project.setName(rs.getString("name"));
            project.setProjectId(rs.getInt("project_id"));
        }catch (SQLException sqle){
            sqle.printStackTrace();
        }
        if(srsc != null){
            JDBCFun.getInstance().closeDBResources(srsc.getStatement(), srsc.getResultSet());
        }

        return project;
    }

    /**
     * Deletes the project with the same projectId
     * @param projectId
     */
    public void deleteProject(int projectId){
        String sql = "delete from projects where project_id = " + projectId;
        Connection conn = JDBCFun.getInstance().getAvailableConnection();
        Statement statement = JDBCFun.getInstance().runUpdate(conn, sql);
        JDBCFun.getInstance().closeDBResources(statement, null);

    }
```

JDBC Sinlgeton

```
public class JDBCFun {

    private List<Connection> connections = new ArrayList<Connection>();
    private int nextIndex = 0;

    private static JDBCFun jdbcFun = null;

    public static JDBCFun getInstance(){
        if(jdbcFun == null){
            jdbcFun = new JDBCFun();
        }
        return jdbcFun;
    }

    private JDBCFun(){

    }
```