BIBLIOGRAPHY

[1] ALLA R. KAMMERDINER, "Multidimensional Assignment Problem –MAP MSC2000: 90C10, 90C27 –," in *Encyclopedia of Optimization*, 2009.

[2] B. Derek, "The Assignment Problem and the Hungarian Method".

[3] C. Lewis, Linear Programming: Theory and Applications, 2008.

[4] G. Carpaneto and P. Toth. Algorithm, Solution of the assignment problem. ACM Transactions on Mathematical Software, vol. Vol 6, 1980.

[5] G. V. Shenoy, Linear Programming: Methods and Applications.

[6] H.W.Kuhn, "The Hungarian Method for the Assignment Problem," *Naval Research Logistics Quarterly*, vol. 2, p. 83–97., 1955.

[7] James B. Orlin & Yusin Lee, "QuickMatch: A Very Fast Algorithm for the Assignment Problem," 1993.

[8] L. Fortnow, "The Status of the P versus NP Problem".

[9] Lawler E. & D. E. Wood, "Branch and Bound Methods: A Survey," in *Operations Research*, 1966.

[10] NASA, "Flight Control Optimization," NASA, [Online]. Available: http://www.nasa.gov/centers/ames/research/technology-onepagers/flight-control-optimization.html.

[11] Philip E. GILL, Walter Murray, Dulce B. Poncele and Michael A. Saunders, "Primal- Dual Methods for Linear Programming," *Technical Report SOL*, vol. Revised, no. March, pp. 91-3, 1994.

[12] R. Ahuja, T. Magnanti, J. Orlin, Network Flows, Prentice Hall, 1993.

[13] Rainer Burkard, Mauro Dell'Amico, Silvano Martello, Assignment Problems, Revised Reprint ed., 2008, pp. 35-202,305-318.

[14] R.E. Burkard, B. Klinz, and R. Rudolf , "Perspectives of Monge properties in optimization," 1996.

[15] R.E. Burkard, R. Rudolf, and G.J.Woeginger, "Three-dimensional axial assignments problems with decomposable cost coefficients," *Discr. Appl. Math*, vol. 65, p. 123–139, 1996.

[16] Reuven Cohen Liran Katzir Danny Raz, "An Efficient Approximation for the Generalized Assignment Problem," Department of Computer Science, Technion, Haifa 32000, Israel.

[17] S. Theussl, "CRAN Task View: Optimization and Mathematical Programming," 2013. [Online]. Available: http://cran.r-project.org/web/views/Optimization.html.

[18] W. P. Pierskalla, "The Multi-dimensional Assignment Problem," 1967.

[19] Y. Crama and F.C.R. Spieksma, "Approximation algorithms for three dimensional assignment problems with triangle inequalities," *European Journal of Operations Research*, vol. 60, p. 273– 279, 1992.

[20] T. Tanino, T. Tanaka and M. Inuiguchi, in *Multi-Objective Programming and Goal Programming -Theory and Applications*, Advances in Intelligent and Soft Computing, Vol. 21, 2003.

APPENDIX A- IMPORTANT CODING IN THE TIMETABLE MANAGER

```
package timetablemanager.entity;
// Generated May 27, 2013 8:59:19 PM by Hibernate Tools 3.2.1.GA
/**
 * Period generated by hbm2java
 */
public class Period  implements java.io.Serializable {
     private int periodId;
     private String startTime;
     private String endTime;
    public Period() {
    }
    public Period(int periodId) {
        this.periodId = periodId;
    }
    public Period(int periodId, String startTime, String endTime) {
       this.periodId = periodId;
       this.startTime = startTime;
       this.endTime = endTime;
    }
    public int getPeriodId() {
        return this.periodId;
    }
    public void setPeriodId(int periodId) {
        this.periodId = periodId;
    }
    public String getStartTime() {
```

```
        return this.startTime;
    }

    public void setStartTime(String startTime) {
        this.startTime = startTime;
    }
    public String getEndTime() {
        return this.endTime;
    }

    public void setEndTime(String endTime) {
        this.endTime = endTime;
    }
}
```

```
package timetablemanager.entity;
// Generated May 27, 2013 8:59:19 PM by Hibernate Tools 3.2.1.GA
/**
 * Score generated by hbm2java
 */
public class Score  implements java.io.Serializable {
     private int scoreId;
     private int periodId;
     private int studentGroupId;
     private int teacherId;
     private int score;
     private Integer tempScore;
     private int selected;
     private Integer unusable;
```

```
    private Integer preoccupied;
    private Integer assigned;
    private String scorecol;
public Score() {
   }
   public Score(int scoreId, int periodId, int studentGroupId, int teacherId, int score, int selected) {
       this.scoreId = scoreId;
       this.periodId = periodId;
       this.studentGroupId = studentGroupId;
       this.teacherId = teacherId;
       this.score = score;
       this.selected = selected;
   }
   public Score(int scoreId, int periodId, int studentGroupId, int teacherId, int score, Integer tempScore, int selected, Integer unusable, Integer preoccupied, Integer assigned, String scorecol) {
      this.scoreId = scoreId;
      this.periodId = periodId;
      this.studentGroupId = studentGroupId;
      this.teacherId = teacherId;
      this.score = score;
      this.tempScore = tempScore;
      this.selected = selected;
      this.unusable = unusable;
      this.preoccupied = preoccupied;
      this.assigned = assigned;
      this.scorecol = scorecol;
   }
```

```
public int getScoreId() {
    return this.scoreId;
}
public void setScoreId(int scoreId) {
    this.scoreId = scoreId;
}
public int getPeriodId() {
    return this.periodId;
}
public void setPeriodId(int periodId) {
    this.periodId = periodId;
}
public int getStudentGroupId() {
    return this.studentGroupId;
}
public void setStudentGroupId(int studentGroupId) {
    this.studentGroupId = studentGroupId;
}
public int getTeacherId() {
    return this.teacherId;
}
public void setTeacherId(int teacherId) {
    this.teacherId = teacherId;
}
public int getScore() {
    return this.score;
}
public void setScore(int score) {
    this.score = score;
```

```
}
public Integer getTempScore() {
    return this.tempScore;
}
public void setTempScore(Integer tempScore) {
    this.tempScore = tempScore;
}
public int getSelected() {
    return this.selected;
}
public void setSelected(int selected) {
    this.selected = selected;
}
public Integer getUnusable() {
    return this.unusable;
}
public void setUnusable(Integer unusable) {
    this.unusable = unusable;
}
public Integer getPreoccupied() {
    return this.preoccupied;
}
public void setPreoccupied(Integer preoccupied) {
    this.preoccupied = preoccupied;
}
public Integer getAssigned() {
    return this.assigned;
}
public void setAssigned(Integer assigned) {
```

```
        this.assigned = assigned;
    }
    public String getScorecol() {
        return this.scorecol;
    }
    public void setScorecol(String scorecol) {
        this.scorecol = scorecol;
    }
}
```

```
package timetablemanager.entity;
// Generated May 27, 2013 8:59:19 PM by Hibernate Tools 3.2.1.GA
/**
 * StudentGroup generated by hbm2java
 */
public class StudentGroup  implements java.io.Serializable {
     private int studentGroupId;
     private String groupName;
     private String grade;
     private Integer noOfStudents;
    public StudentGroup() {
    }
    public StudentGroup(int studentGroupId) {
        this.studentGroupId = studentGroupId;
    }
    public StudentGroup(int studentGroupId, String groupName, String grade, Integer noOfStudents) {
        this.studentGroupId = studentGroupId;
        this.groupName = groupName;
        this.grade = grade;
```

```
        this.noOfStudents = noOfStudents;
    }
    public int getStudentGroupId() {
        return this.studentGroupId;
    }
     public void setStudentGroupId(int studentGroupId) {
        this.studentGroupId = studentGroupId;
    }
    public String getGroupName() {
        return this.groupName;
    }
    public void setGroupName(String groupName) {
        this.groupName = groupName;
    }
    public String getGrade() {
        return this.grade;
    }
    public void setGrade(String grade) {
        this.grade = grade;
    }
    public Integer getNoOfStudents() {
        return this.noOfStudents;
    }
    public void setNoOfStudents(Integer noOfStudents) {
        this.noOfStudents = noOfStudents;
    }
}
```

```
package timetablemanager.entity;
```

```
// Generated May 27, 2013 8:59:19 PM by Hibernate Tools 3.2.1.GA
/**
 * Subject generated by hbm2java
 */
public class Subject  implements java.io.Serializable {
     private int subjectId;
     private String subjectName;
     private String grade;
     private Integer periodsPerTerm;
    public Subject() {
    }
    public Subject(int subjectId) {
        this.subjectId = subjectId;
    }
    public Subject(int subjectId, String subjectName, String grade, Integer periodsPerTerm) {
       this.subjectId = subjectId;
       this.subjectName = subjectName;
       this.grade = grade;
       this.periodsPerTerm = periodsPerTerm;
    }
    public int getSubjectId() {
        return this.subjectId;
    }
    public void setSubjectId(int subjectId) {
        this.subjectId = subjectId;
    }
    public String getSubjectName() {
        return this.subjectName;
```

```
    }
    public void setSubjectName(String subjectName) {
        this.subjectName = subjectName;
    }
    public String getGrade() {
        return this.grade;
    }
    public void setGrade(String grade) {
        this.grade = grade;
    }
    public Integer getPeriodsPerTerm() {
        return this.periodsPerTerm;
    }
    public void setPeriodsPerTerm(Integer periodsPerTerm) {
        this.periodsPerTerm = periodsPerTerm;
    }
}
```

```
package timetablemanager.entity;
// Generated May 27, 2013 8:59:19 PM by Hibernate Tools 3.2.1.GA
/**
 * Teacher generated by hbm2java
 */
public class Teacher  implements java.io.Serializable {
     private int teacherId;
     private String name;
     private String address;
```

```
    private String telephone;
    private String email;
    private Boolean password
   public Teacher() {
   }
   public Teacher(int teacherId) {
      this.teacherId = teacherId;
   }
   public Teacher(int teacherId, String name, String address, String telephone,
String email, Boolean password) {
      this.teacherId = teacherId;
      this.name = name;
      this.address = address;
      this.telephone = telephone;
      this.email = email;
      this.password = password;
   }
   public int getTeacherId() {
      return this.teacherId;
   }
   public void setTeacherId(int teacherId) {
      this.teacherId = teacherId;
   }
   public String getName() {
      return this.name;
   }
   public void setName(String name) {
      this.name = name;
   }
```

```
    public String getAddress() {
        return this.address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    public String getTelephone() {
        return this.telephone;
    }
    public void setTelephone(String telephone) {
        this.telephone = telephone;
    }
    public String getEmail() {
        return this.email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public Boolean getPassword() {
        return this.password;
    }
    public void setPassword(Boolean password) {
        this.password = password;
    }
}
```

```
package timetablemanager.entity;
```

```
// Generated May 27, 2013 8:59:19 PM by Hibernate Tools 3.2.1.GA
/**
 * TeacherAssignment generated by hbm2java
 */
public class TeacherAssignment  implements java.io.Serializable {
     private int assignmentId;
     private int teacherId;
     private int studentGroupId;
     private int subjectId;
    public TeacherAssignment() {
    }
    public TeacherAssignment(int assignmentId, int teacherId, int studentGroupId, int subjectId) {
        this.assignmentId = assignmentId;
        this.teacherId = teacherId;
        this.studentGroupId = studentGroupId;
        this.subjectId = subjectId;
    }
    public int getAssignmentId() {
        return this.assignmentId;
    }
    public void setAssignmentId(int assignmentId) {
        this.assignmentId = assignmentId;
    }
    public int getTeacherId() {
        return this.teacherId;
    }
    public void setTeacherId(int teacherId) {
        this.teacherId = teacherId;
```

```
    }
    public int getStudentGroupId() {
        return this.studentGroupId;
    }
    public void setStudentGroupId(int studentGroupId) {
        this.studentGroupId = studentGroupId;
    }
    public int getSubjectId() {
        return this.subjectId;
    }
    public void setSubjectId(int subjectId) {
        this.subjectId = subjectId;
    }
}
```

```
package timetablemanager.entity;
// Generated May 27, 2013 8:59:19 PM by Hibernate Tools 3.2.1.GA
/**
 * Zeros generated by hbm2java
 */
public class Zeros  implements java.io.Serializable {
     private int id;
     private Integer periodId;
     private Integer teacherId;
     private Integer studentGroupId;
     private Integer selected;
     private Integer noOfZeros;
     private Integer noOfZerosTemp;
     private Integer assigned;
```

```
    private Integer unusable;
    private String zeroscol2;
    private String zeroscol3;
   public Zeros() {
   }
   public Zeros(int id) {
      this.id = id;
   }
   public Zeros(int id, Integer periodId, Integer teacherId, Integer studentGroupId, Integer selected, Integer noOfZeros, Integer noOfZerosTemp, Integer assigned, Integer unusable, String zeroscol2, String zeroscol3) {
      this.id = id;
      this.periodId = periodId;
      this.teacherId = teacherId;
      this.studentGroupId = studentGroupId;
      this.selected = selected;
      this.noOfZeros = noOfZeros;
      this.noOfZerosTemp = noOfZerosTemp;
      this.assigned = assigned;
      this.unusable = unusable;
      this.zeroscol2 = zeroscol2;
      this.zeroscol3 = zeroscol3;
   }
   public int getId() {
      return this.id;
   }
   public void setId(int id) {
      this.id = id;
   }
```

```
public Integer getPeriodId() {
    return this.periodId;
}
public void setPeriodId(Integer periodId) {
    this.periodId = periodId;
}
public Integer getTeacherId() {
    return this.teacherId;
}
public void setTeacherId(Integer teacherId) {
    this.teacherId = teacherId;
}
public Integer getStudentGroupId() {
    return this.studentGroupId;
}
public void setStudentGroupId(Integer studentGroupId) {
    this.studentGroupId = studentGroupId;
}
public Integer getSelected() {
    return this.selected;
}
public void setSelected(Integer selected) {
    this.selected = selected;
}
public Integer getNoOfZeros() {
    return this.noOfZeros;
}
public void setNoOfZeros(Integer noOfZeros) {
    this.noOfZeros = noOfZeros;
```

```
}
public Integer getNoOfZerosTemp() {
    return this.noOfZerosTemp;
}
public void setNoOfZerosTemp(Integer noOfZerosTemp) {
    this.noOfZerosTemp = noOfZerosTemp;
}
public Integer getAssigned() {
    return this.assigned;
}
public void setAssigned(Integer assigned) {
    this.assigned = assigned;
}
public Integer getUnusable() {
    return this.unusable;
}
public void setUnusable(Integer unusable) {
    this.unusable = unusable;
}
public String getZeroscol2() {
    return this.zeroscol2;
}
public void setZeroscol2(String zeroscol2) {
    this.zeroscol2 = zeroscol2;
}
public String getZeroscol3() {
    return this.zeroscol3;
}
public void setZeroscol3(String zeroscol3) {
```

```
        this.zeroscol3 = zeroscol3;
    }
}
```

Function Assign

```
public static void Assign() {//Hungarian method of solving is used
ScoreTable.resetTempScores();//delete all entries in zeros table and reset temp scores to values in originally assigned scores
System.out.println("step 1: TempScore=TempScore-MinScore");//comment for debugging
ScoreTable.DeductFirstMinScore();//deducting first min score from all the values. Third step in flow chart
ScoreTable.SetZeroCounter("selected");//store all counts of zeros in NoOfZerosTemp where “selected=0”
ScoreTable.MarkZerosSelected();//Select a zeros row first and identify its scores rows till the last zeros row one by one.
                                //One Score is only considered one time.
if (!AreThereEnoughNumberOfZerosRows()){// if number of independent zeros rows are not enough do again the deduction
System.out.println("step 2: Deducting Second set of Min Scores");
ScoreTable.DeductSecondMinScores();
ScoreTable.ResetOnlyScoreStatusSelected();
ScoreTable.SetZeroCounter("selected");//
 System.out.println("step 2: done");
ScoreTable.MarkZerosSelected();//Select max no of zeros row first and identify its scores rows and do the same till the last zeros row
}
//  after marking all zeros count the items having no_of_zeros more than or equal to 1
//  if marked items are equal to min{teacher.period,student groups.period} set finish for that period and go for final step
```

```
//System.out.println("*** step 10:if count is less than min{no of teacher.period,no of student groups.period} Redusing Min Score-general");
while(!AreThereEnoughNumberOfZerosRows()){
System.out.println("*** step 10:if count is less than min{no of teacher.period,no of student groups.period} Redusing Min Score-general");
ScoreTable.DeductNextMinScore();
ScoreTable.ResetOnlyScoreStatusSelected();
ScoreTable.SetZeroCounter("selected");//Count No of zeros and store all in NoOfZeros/NoOfZerosTemp
ScoreTable.MarkZerosSelected();
}
if (AreThereEnoughNumberOfZerosRows()){
// assign single zeros for student groups and teachers while removing them from the list
System.out.println("Calling Zero Assignment");
//"May need to edit for correct Assignments"
ScoreTable.ResetOnlyScoreStatusAssignedAndUnusable();
ScoreTable.SetZeroCounter("unusable");
ScoreTable.MarkZerosAssigned();
}
//Final step :if no of zeros rows exceed the required assignment points Do the assignment
//if done return assignments
}
```

Function getByTeacher

```
public static void getByTeacher(int teacher_id) {
try {
        Session session = HibernateUtil.getSessionFactory().openSession();
```

```
        session.beginTransaction();
                List<Score> resultList= session.createCriteria(Score.class).add(Restrictions.eq("teacherId", teacher_id)).list();
                for ( Score score : resultList ) {
                System.out.println(score.getScoreId() + "\t\t" + score.getTeacherId()+ "\t\t" + score.getStudentGroupId() + "\t\t"+ score.getPeriodId()+ "\t\t"+score.getScore());
                //score.setScore(score.getScore()-1);
        }
        session.getTransaction().commit();
        session.close();
        } catch (HibernateException he) {
                he.printStackTrace();
        }
        return;
}
```

Function getbyPeriod

```
public static void getbyPeriod(int period_id) {
        try {
                Session session = HibernateUtil.getSessionFactory().openSession();
                session.beginTransaction();
                List<Score> resultList= session.createCriteria(Score.class).add(Restrictions.eq("periodId", period_id)).list();
                for ( Score score : resultList ) {
                        System.out.println(score.getScoreId() + "\t\t" + score.getTeacherId()+ "\t\t" + score.getStudentGroupId() + "\t\t"+ score.getPeriodId()+ "\t\t"+score.getScore());
```

```
            }
            session.getTransaction().commit();
            session.close();
        } catch (HibernateException he) {
            he.printStackTrace();
        }
        return;
}
```

Function getByStudentGroup

```
public static void getByStudentGroup(int student_group_id) {
        try {
            Session session = HibernateUtil.getSessionFactory().openSession();
            session.beginTransaction();
            List<Score> resultList= session.createCriteria(Score.class).add(Restrictions.eq("studentGroupId", student_group_id)).list();
            for ( Score score : resultList ) {
                System.out.println(score.getScoreId() + "\t\t" + score.getTeacherId()+ "\t\t" + score.getStudentGroupId() + "\t\t"+ score.getPeriodId()+ "\t\t"+score.getScore());
            }
            session.getTransaction().commit();
            session.close();
        } catch (HibernateException he) {
            he.printStackTrace();
        }
        return;
}
```

Function generateByTeacher

```
public static void generateByTeacher(int teacher_id) {
      try {
         Session session = HibernateUtil.getSessionFactory().openSession();
         session.beginTransaction();
         List<StudentGroup> studentGroupResultList=
session.createCriteria(StudentGroup.class).list();
         //List<Teacher> teacher=session.createCriteria(Teacher.class).list();
         List<Period> periodResultList=session.createCriteria(Period.class).list();
         for ( StudentGroup studentGroup : studentGroupResultList ) {
            int student_group_id=studentGroup.getStudentGroupId();
            int initial_score=100;
            int initial_selection=0;
            int score_id=0;
            try{

score_id=(Integer)session.createCriteria(Score.class).setProjection(Projections.m
ax("scoreId")).uniqueResult();
               System.out.println(score_id);
            }catch(NullPointerException np){
               np.printStackTrace();
               score_id=0;
            }
            System.out.println("StudentGroupID"+student_group_id );
            for ( Period period : periodResultList ) {
               int period_id=period.getPeriodId();
               score_id++;
               Score score=new Score(score_id, student_group_id, period_id,
initial_score ,teacher_id, initial_selection);
```

```
                session.save(score);
            }
        }
        session.getTransaction().commit();
        session.close();

    } catch (HibernateException he) {
        he.printStackTrace();
    }
    return;
}
```

Function DeductFirstMinScore

```
public static void DeductFirstMinScore(){
    try{
        Session session = HibernateUtil.getSessionFactory().openSession();
    session.beginTransaction();
    int
min_score=(Integer)session.createCriteria(Score.class).setProjection(Projections.
min("tempScore")).uniqueResult();
    List<Score> tempScoreList=session.createCriteria(Score.class).list();
    System.out.println("minscore for all="+min_score);
    for(Score score: tempScoreList){
        score.setTempScore(score.getTempScore()-min_score);
    }
    session.getTransaction().commit();
    session.close();
}
    catch (HibernateException he) {
```

```
            he.printStackTrace();
        }
    }
```

Function clearAll

```
public static void clearAll() {
        try {
            Session session = HibernateUtil.getSessionFactory().openSession();
            session.beginTransaction();
            //List<StudentGroup>                    studentGroupResultList=
session.createCriteria(StudentGroup.class).list();
            session.delete("FROM score");
            session.getTransaction().commit();
            session.close();

        } catch (HibernateException he) {
            he.printStackTrace();
        }
    }
```

Function generateByPeriod

```
    public static void generateByPeriod(int period_id) {
        try {
            Session session = HibernateUtil.getSessionFactory().openSession();
            session.beginTransaction();
            List<StudentGroup>                    studentGroupResultList=
session.createCriteria(StudentGroup.class).list();
            List<Teacher>
teacherResultList=session.createCriteria(Teacher.class).list();
```

```
        for ( StudentGroup studentGroup : studentGroupResultList ) {
            int student_group_id=studentGroup.getStudentGroupId();
            int initial_score=100;
      int initial_selection=0;
            for ( Teacher teacher : teacherResultList ) {
               int score_id=0;
               try{

score_id=(Integer)session.createCriteria(Score.class).setProjection(Projections.max("scoreId")).uniqueResult();
                  System.out.println(score_id);
               }catch(NullPointerException np){
                 np.printStackTrace();
                 score_id=0;
               }
int teacher_id=teacher.getTeacherId();
               score_id++;
               Score score=new Score(score_id, student_group_id, period_id, initial_score ,teacher_id, initial_selection);
               session.save(score);
            }
         }
         session.getTransaction().commit();
         session.close();
      } catch (HibernateException he) {
         he.printStackTrace();
      }
      return;
   }
```

Function clearByTeacher

```
public static void clearByTeacher(int teacher_id) {
      try {
         Session session = HibernateUtil.getSessionFactory().openSession();
         session.beginTransaction();
         session.delete("FROM Score * WHERE teacherId = teacher_id");
         session.getTransaction().commit();
         session.close();
      } catch (HibernateException he) {
         he.printStackTrace();
      }
   }
```

Function clearByStudentGroup

```
      public static void clearByStudentGroup(int student_group_id) {
      try {
         Session session = HibernateUtil.getSessionFactory().openSession();
         session.beginTransaction();
         session.delete("FROM Score * WHERE studentGroupId =
student_group_id");
         session.getTransaction().commit();
         session.close();
      } catch (HibernateException he) {
         he.printStackTrace();
      }
   }
```

Function clearByPeriod

```
public static void clearByPeriod(int period_id) {
try {
Session session = HibernateUtil.getSessionFactory().openSession();
session.beginTransaction();
//List<StudentGroup> studentGroupResultList=
session.createCriteria(StudentGroup.class).list();
session.delete("FROM Score * WHERE periodId = period_id");
session.getTransaction().commit();
session.close();

} catch (HibernateException he) {
he.printStackTrace();
}
}
```

Function clearTemp

```
public static void clearTemp() {
try {
Session session = HibernateUtil.getSessionFactory().openSession();
session.beginTransaction();
List<StudentGroup> studentGroupResultList=
session.createCriteria(StudentGroup.class).list();
session.update("FROM Score tempscore", "100");
session.getTransaction().commit();
session.close();
} catch (HibernateException he) {
he.printStackTrace();
}
```

```
}
```

Function resetTempScores

```
public static void resetTempScores() {
try {
Session session = HibernateUtil.getSessionFactory().openSession();
session.beginTransaction();
List<Score> scoreResultList = session.createCriteria(Score.class).list();
for(Score score : scoreResultList) {
int score_value = score.getScore();
System.out.println("score of score_id =
"+score.getScoreId()+"is"+score_value);
score.setTempScore(score_value);
score.setAssigned(0);
score.setSelected(0);
score.setUnusable(0);
}
session.getTransaction().commit();
session.close();
} catch (HibernateException he) {
he.printStackTrace();
}
return;
}
```

Function ResetOnlyScoreStatusSelected

```
public static void ResetOnlyScoreStatusSelected(){
try {
Session session = HibernateUtil.getSessionFactory().openSession();
```

```
   session.beginTransaction();
        List<Score> scoreResultList = session.createCriteria(Score.class).list();
        for(Score score : scoreResultList) {
           int score_value = score.getScore();
           System.out.println("score of score_id = "+score.getScoreId()+"is"+score_value);
           score.setSelected(0);
        }
        session.getTransaction().commit();
        session.close();
     } catch (HibernateException he) {
        he.printStackTrace();
     }
     return;
   }
```

Function ResetOnlyScoreStatusAssignedAndUnusable

```
     public static void ResetOnlyScoreStatusAssignedAndUnusable(){
     try {
        Session session = HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        System.out.println("ResetOnlyScoreStatusAssigned");
        List<Score> scoreResultList = session.createCriteria(Score.class).list();
        for(Score score : scoreResultList) {
           score.setAssigned(0);
           score.setUnusable(0);
        }
        session.getTransaction().commit();
        session.close();
```

```
        } catch (HibernateException he) {
            he.printStackTrace();
        }
        return;
    }
```

Function getAssignmentTable

```
    public static void getAssignmentTable() {
        //Show the assignments
        try {
            Session session = HibernateUtil.getSessionFactory().openSession();
            session.beginTransaction();
            List<Score> scoreResultList= session.createCriteria(Score.class).add(Restrictions.eq("tempScore", 0)).list();
            for ( Score score : scoreResultList ) {
                System.out.println(score.getTeacherId() + "\t\t" + score.getStudentGroupId() + "\t\t" + score.getPeriodId());
            }
            session.getTransaction().commit();
            session.close();
        } catch (HibernateException he) {
            he.printStackTrace();
        }
    }
```

Function DeductSecondMinScores

```
public static void DeductSecondMinScores(){
try {
```

```
Session session = HibernateUtil.getSessionFactory().openSession();
session.beginTransaction();
System.out.println("***DeductSecondMinScores():select Period.Teacher or peroiod.studentgroup at a time but here it is not considered about teacher.studentgroup ");//step 1.1:select Period.Teacher or peroiod.studentgroup at a time but here it is not considered about teacher.studentgroup assuming that the same teacher's score not needed to be deducted. better consider only peroiod.studentgroup as in all other products same teachers scores are being compettitive to each score
List<Period> periodResultList= session.createCriteria(Period.class).list(); //deducting minscore for a period
for ( Period period : periodResultList ) {
        int period_id= period.getPeriodId();

intmin_period_score=(Integer)session.createCriteria(Score.class).add(Restrictions.eq("periodId", period_id)).setProjection(Projections.min("tempScore")).uniqueResult();
List<Score> tempPeriodScoreList=session.createCriteria(Score.class).add(Restrictions.eq("periodId", period_id)).list();
System.out.println("*** minscore for period "+period_id+"="+min_period_score);
for(Score score: tempPeriodScoreList){
        score.setTempScore(score.getTempScore()-min_period_score);
}
   System.out.println("*** Get the zero points filtered out from the all scores");     //Call Zero Finder here
   ScoreTable.SetZeroCounter("selected");
   System.out.println("*** Get the minimum value entered by each teacher and substract it from all sets of scores of that teacher");//step 2:Get the minimum value entered by each teacher and substract it from all sets of
```

```
scores(with regards all student groups) of that teacher
        List<Teacher> periodTeacherResultList=
session.createCriteria(Teacher.class).list();
        for ( Teacher teacher : periodTeacherResultList ) {
                int teacher_id= teacher.getTeacherId();
                int
min_period_teacher_score=(Integer)session.createCriteria(Score.class).add(Rest
rictions.eq("teacherId",
teacher_id)).setProjection(Projections.min("tempScore")).uniqueResult();
        List<Score>
tempPeriodTeacherScoreList=session.createCriteria(Score.class).add(Restriction
s.eq("periodId", period_id)).add(Restrictions.eq("teacherId", teacher_id)).list();
                System.out.println("*** minscore for period "+period_id+" &
teacher "+teacher_id+"="+min_period_teacher_score);
                for(Score score: tempPeriodTeacherScoreList){
                        score.setTempScore(score.getTempScore()-
min_period_teacher_score);
                }
                System.out.println("*** Get the zero points filtered out from the
all scores");
                ScoreTable.SetZeroCounter("selected");
        }
        System.out.println("*** Get the minimum value derived from step 1 in
each student group and substract it from all sets of scores of that student
group"); //step 3:Get the minimum value derived from step 1 in each student
group and substract it from all sets of scores(with regards to all the teachers) of
that student group
        List<StudentGroup> periodStudentGroupResultList=
session.createCriteria(StudentGroup.class).list();
        for ( StudentGroup studentGroup : periodStudentGroupResultList ) {
                int student_group_id= studentGroup.getStudentGroupId();
```

```
                int
min_period_student_group_score=(Integer)session.createCriteria(Score.class).ad
d(Restrictions.eq("studentGroupId",
student_group_id)).setProjection(Projections.min("tempScore")).uniqueResult()
;
        List<Score>
tempPeriodStudentGroupScoreList=session.createCriteria(Score.class).add(Rest
rictions.eq("periodId",        period_id)).add(Restrictions.eq("studentGroupId",
student_group_id)).list();
                System.out.println("*** minscore for period "+period_id+" &
student group "+student_group_id+"="+min_period_student_group_score);
                for(Score score: tempPeriodStudentGroupScoreList){
                        score.setTempScore(score.getTempScore()-
min_period_student_group_score);
                }//session.getTransaction().commit();


                System.out.println("*** step 4:Get the zero points filtered out
from the all scores"); //step 4:Get the zero points filtered out from the all scores
                ScoreTable.SetZeroCounter("selected");
                }
        }
        session.getTransaction().commit();
        session.close();
}catch (HibernateException he) {
        he.printStackTrace();
}
}
```

Function generateByTeacher

```
  public static void generateByTeacher(int teacher_id) {
```

```
   }
public static void DeductNextMinScore(){

      try{

         Session session = HibernateUtil.getSessionFactory().openSession();
         session.beginTransaction();
         //  In Uncovered find the minimum score
         int uncovered_min_score=0;
         try{

uncovered_min_score=(Integer)session.createCriteria(Score.class).add(Restrictio
ns.eq("selected",
0)).setProjection(Projections.min("tempScore")).uniqueResult();
            System.out.println("***
uncovered_min_score="+uncovered_min_score);
         }
         catch (NullPointerException np) {
         np.printStackTrace();
         uncovered_min_score=0;
         }
         List<Score> tempScoreList=session.createCriteria(Score.class).list();
         System.out.println("*** uncovered minscore for
all="+uncovered_min_score);
         //  reduce that value from all uncovered elements and add that values to
cross( double coverd )scores or if triple covered add minimum scorex2
         for(Score score: tempScoreList){
            score.setTempScore(score.getTempScore()-
uncovered_min_score+uncovered_min_score*(score.getSelected()));
         }
```

```
        session.getTransaction().commit();
        session.close();
    }
        catch (HibernateException he) {
        he.printStackTrace();
    }
  }
```

Function AreThereEnoughNumberOfZerosRows

```
   public static boolean AreThereEnoughNumberOfZerosRows(){
      boolean enough_number_of_zeros_rows=false;
      try{
         Session session = HibernateUtil.getSessionFactory().openSession();
         session.beginTransaction();
         int
no_of_zeros_rows=session.createCriteria(Zeros.class).add(Restrictions.eq("selec
ted", 1)).list().size();
         int no_of_teachers=session.createCriteria(Teacher.class).list().size();
         int
no_of_student_groups=session.createCriteria(Teacher.class).list().size();
         int no_of_periods=session.createCriteria(Period.class).list().size();
         System.out.println("*** No of Zeros Rows which are selected="+
no_of_zeros_rows);
         int a=no_of_periods*no_of_teachers;
         int b=no_of_periods*no_of_student_groups;
         session.getTransaction().commit();
         session.close();
         if ((no_of_zeros_rows>=a)||(no_of_zeros_rows>=b)){
            enough_number_of_zeros_rows=true;
```

```
        }
      }
      catch (HibernateException he) {
        he.printStackTrace();
      }
   return enough_number_of_zeros_rows;
   }
```

Function SetZeroCounter

```
   public static void SetZeroCounter(String column){
      ScoreTable.DeleteZeroCounter();//Delete all entries in the Zeros list
      try{
         Session session = HibernateUtil.getSessionFactory().openSession();
         session.beginTransaction();
         List<Teacher>
teacherResultList=session.createCriteria(Teacher.class).list();
         List<StudentGroup>
studentGroupResultList=session.createCriteria(StudentGroup.class).list();
         List<Period> periodResultList=session.createCriteria(Period.class).list();
         int id=1;
         System.out.println("***zeros id" +id+" ");
         //step 5:Get zeros calculated for a teacher in the perticular period
         for (Period period: periodResultList){
            int period_id=period.getPeriodId();
            for (Teacher teacher: teacherResultList){
               int teacher_id=teacher.getTeacherId();
               Criterion crit1 = Restrictions.eq("tempScore",0);
               Criterion crit2 = Restrictions.eq("periodId",period_id);
               Criterion crit3 = Restrictions.eq("teacherId",teacher_id);
```

```
                Criterion crit4 = Restrictions.eq(column,0);
                Criterion crit5 = Restrictions.and(crit1, crit2);
                Criterion crit6 = Restrictions.and(crit3, crit4);
                Criterion crit7 = Restrictions.and(crit5, crit6);
                int                                                    zero_count
=session.createCriteria(Score.class).add(crit7).list().size();
                Zeros zeros=new Zeros();
                zeros.setId(id);
                zeros.setNoOfZeros(zero_count);
                zeros.setNoOfZerosTemp(zero_count);
                zeros.setPeriodId(period_id);
                zeros.setTeacherId(teacher_id);
                zeros.setStudentGroupId(0);
                zeros.setSelected(0);
                zeros.setAssigned(0);
                zeros.setUnusable(0);
                session.save(zeros);
                System.out.println("*** Updating zerosId="+id+" zero score for
period="+period_id+" & teacher="+teacher_id+"no of zeros="+zero_count);
                id++;
             }
          }
          //step 6:Get zeros calculated for a student group in the perticular period
          for (Period period: periodResultList){
             int period_id=period.getPeriodId();
             for (StudentGroup studentGroup: studentGroupResultList){
                int student_group_id=studentGroup.getStudentGroupId();

                Criterion crit1 = Restrictions.eq("tempScore",0);
```

```
            Criterion crit2 = Restrictions.eq("periodId",period_id);
            Criterion crit3 = Restrictions.eq("studentGroupId",student_group_id);
            Criterion crit4 = Restrictions.eq(column,0);
            Criterion crit5 = Restrictions.and(crit1, crit2);
            Criterion crit6 = Restrictions.and(crit3, crit4);
            Criterion crit7 = Restrictions.and(crit5, crit6);
            int zero_count =session.createCriteria(Score.class).add(crit7).list().size();
            Zeros zeros=new Zeros();
            zeros.setId(id);
            zeros.setNoOfZeros(zero_count);
            zeros.setNoOfZerosTemp(zero_count);
            zeros.setPeriodId(period_id);
            zeros.setStudentGroupId(student_group_id);
            zeros.setTeacherId(0);
            zeros.setSelected(0);
            zeros.setAssigned(0);
            zeros.setUnusable(0);
        session.save(zeros);
            System.out.println("*** Updating zerosId="+id+" zero score for period="+period_id+" & student group="+student_group_id+"no of zeros="+zero_count);
            id++;
          }
        }
        for (Teacher teacher: teacherResultList){
          int teacher_id=teacher.getTeacherId();
          for (StudentGroup studentGroup: studentGroupResultList){
```

```
            int student_group_id=studentGroup.getStudentGroupId();
            Criterion crit1 = Restrictions.eq("tempScore",0);
            Criterion crit2 = Restrictions.eq("teacherId",teacher_id);
            Criterion crit3 = Restrictions.eq("studentGroupId",student_group_id);
            Criterion crit4 = Restrictions.eq(column,0);
            Criterion crit5 = Restrictions.and(crit1, crit2);
            Criterion crit6 = Restrictions.and(crit3, crit4);
            Criterion crit7 = Restrictions.and(crit5, crit6);
            int zero_count =session.createCriteria(Score.class).add(crit7).list().size();
            Zeros zeros=new Zeros();
            zeros.setId(id);
            zeros.setNoOfZeros(zero_count);
            zeros.setNoOfZerosTemp(zero_count);
            zeros.setTeacherId(teacher_id);
            zeros.setStudentGroupId(student_group_id);
            zeros.setPeriodId(0);
            zeros.setSelected(0);
            zeros.setAssigned(0);
            zeros.setUnusable(0);
            session.save(zeros);
            System.out.println("*** Updating zerosId="+id+" zero score for teacher="+teacher_id+" & student group="+student_group_id+"no of zeros="+zero_count);
            id++;
          }
        }
        session.getTransaction().commit();
```

```
            session.close();
        }
    catch (HibernateException he) {
            he.printStackTrace();
        }
    }
    //update zero counter without taking selected items from score table
```

Function UpdateZeroCounter

```
public static void UpdateZeroCounter(String column){
        //Counting Number of Zeros for each teacher.student group,teacher.period,period.student group
        try{
            Session session = HibernateUtil.getSessionFactory().openSession();
            session.beginTransaction();
            List<Zeros> zerosResultList=session.createCriteria(Zeros.class).list();
            for(Zeros zeros:zerosResultList){
                int period_id=zeros.getPeriodId();
                int teacher_id=zeros.getTeacherId();
                int student_group_id=zeros.getStudentGroupId();
                //"May need to edit for multiple periods"
                if(period_id==0){
                    Criterion crit1 = Restrictions.eq("tempScore",0);
                    Criterion crit2 = Restrictions.eq("teacherId",teacher_id);
                    Criterion crit3 = Restrictions.eq("studentGroupId",student_group_id);
                    Criterion crit4 = Restrictions.eq(column,0);
                    Criterion crit5 = Restrictions.and(crit1, crit2);
                    Criterion crit6 = Restrictions.and(crit3, crit4);
```

```
            Criterion crit7 = Restrictions.and(crit5, crit6);
            int zero_count
=session.createCriteria(Score.class).add(crit7).list().size();
            System.out.println("*** zero score for teacher="+teacher_id+" &
student group="+student_group_id+"no of zeros="+zero_count);
            zeros.setNoOfZerosTemp(zero_count);
            }
          if(teacher_id==0){
            Criterion crit1 = Restrictions.eq("tempScore",0);
            Criterion crit2 = Restrictions.eq("periodId",period_id);
            Criterion crit3 =
Restrictions.eq("studentGroupId",student_group_id);
            Criterion crit4 = Restrictions.eq(column,0);
            Criterion crit5 = Restrictions.and(crit1, crit2);
            Criterion crit6 = Restrictions.and(crit3, crit4);
            Criterion crit7 = Restrictions.and(crit5, crit6);
            int zero_count
=session.createCriteria(Score.class).add(crit7).list().size();
            System.out.println(" zero score for period="+period_id+" & student
group="+student_group_id+"no of zeros="+zero_count);
            zeros.setNoOfZerosTemp(zero_count);
            }
          if(student_group_id==0){
            Criterion crit1 = Restrictions.eq("tempScore",0);
            Criterion crit2 = Restrictions.eq("periodId",period_id);
            Criterion crit3 = Restrictions.eq("teacherId",teacher_id);
            Criterion crit4 = Restrictions.eq(column,0);
            Criterion crit5 = Restrictions.and(crit1, crit2);
            Criterion crit6 = Restrictions.and(crit3, crit4);
            Criterion crit7 = Restrictions.and(crit5, crit6);
```

```
                int zero_count
=session.createCriteria(Score.class).add(crit7).list().size();
                System.out.println(" zero score for period="+period_id+" &
teacher="+teacher_id+"no of zeros="+zero_count);
                zeros.setNoOfZerosTemp(zero_count);
                }
            session.getTransaction().commit();
            session.beginTransaction();
         }
         session.close();
      }
   catch (HibernateException he) {
         he.printStackTrace();
      }
   }
   //Delete zero counter results /zeros table
   public static void DeleteZeroCounter(){
      Session session = HibernateUtil.getSessionFactory().openSession();
      session.beginTransaction();
      List<Zeros> zerosResultset=session.createCriteria(Zeros.class).list();
      for(Zeros zeros: zerosResultset){
            session.delete(zeros);
      }
      session.getTransaction().commit();
      session.close();
   }
   //Printing out Zeros table in degrading order
```

Function ArrangeZerosInDegradingOrder

```
public static void ArrangeZerosInDegradingOrder(){
    // for each time needed the list have to be called in the required order
    Session session = HibernateUtil.getSessionFactory().openSession();
    session.beginTransaction();
    List<Zeros> zerosResultset=session.createCriteria(Zeros.class).addOrder(
Order.desc("noOfZeros") ).list();

    for(Zeros zeros: zerosResultset){
        System.out.println("*** Id "+zeros.getId()+" No of
Zeros"+zeros.getNoOfZeros());
    }
    session.getTransaction().commit();
  }
```

Function MarkZerosSelected

```
public static void MarkZerosSelected(){
    //arrange those above(both student groups and teachers) in degrading
order and set selected while removing the selected zeros from no of zeros column
by calling ScoreTable.UpdateZeroCounter();
    System.out.println("*** Starting Method:MarkZerosSelected");
    ScoreTable.ResetOnlyScoreStatusSelected();
    int no_of_unselected_no_of_zeros_nonzero_rows=1;
    while(no_of_unselected_no_of_zeros_nonzero_rows>0){
      try{
        Session session = HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        Criterion crit1 =Restrictions.eq("selected",0);
        Criterion crit2 =Restrictions.ne("noOfZerosTemp", 0);
```

```
            List<Zeros>
zerosResultset=session.createCriteria(Zeros.class).add(Restrictions.and(crit1,
crit2)).addOrder( Order.desc("noOfZerosTemp") ).list();
            no_of_unselected_no_of_zeros_nonzero_rows=zerosResultset.size();
            System.out.println("***
no_of_unselected_no_of_zeros_nonzero_rows="+no_of_unselected_no_of_zeros_
nonzero_rows);
            //get first 'zeros'
            if(no_of_unselected_no_of_zeros_nonzero_rows==0){
               break;
            }
            Zeros zeros=zerosResultset.get(0);
            System.out.println("*** Getting Zeros Selected Id "+zeros.getId()+"
No of Zeros"+zeros.getNoOfZeros());
            //mark selected for that 'zeros'
            zeros.setSelected(1);
            //zeros.setZeroscol1(1);
            session.getTransaction().commit();
            //mark selected for scores that were covered by the perticular zeros.
            ScoreTable.MarkScoreSelected(zeros);
            //recount for maximum no of covered/uncovered zero scores
            //not making any adjustment to SELECTED column in Zeros table and
update noOfZerosTemp in Zeros table by counting score tempScore=0 and
selected=0 in Scores table
            ScoreTable.UpdateZeroCounter("selected");// refreshing the
Zerocounter to detect the changes done
            session.close();
            no_of_unselected_no_of_zeros_nonzero_rows--;
            //go to next zeros`
         }
```

```
        catch (HibernateException he) {
            he.printStackTrace();
        }
    }
    System.out.println("*** Terminating MarkZerosSelected");
  }
  //"Store No of Zeros" in "No of Zeros Temp"
```

Function StoreFirstZeros

```
public static void StoreFirstZeros(){
    Session session = HibernateUtil.getSessionFactory().openSession();
    session.beginTransaction();
    List<Zeros>  zerosResultset=session.createCriteria(Zeros.class).addOrder(
Order.desc("noOfZeros") ).list();
    //"Store No of Zeros" in "No of Zeros Temp"
    for(Zeros zeros: zerosResultset){
        zeros.setNoOfZerosTemp(zeros.getNoOfZeros());
    }
    session.getTransaction().commit();
  }
  //For a selected zero mark corresponding scores selected
```

Function MarkScoreSelected

```
public static void MarkScoreSelected(Zeros zeros){
    //make selected=1 for the given 'Zeros' or if already selected make
selected=selected+1
    Session session = HibernateUtil.getSessionFactory().openSession();
    session.beginTransaction();
    int period_id=zeros.getPeriodId();
```

```
        int teacher_id=zeros.getTeacherId();
        int student_group_id=zeros.getStudentGroupId();
        Criterion crit1 = Restrictions.eq("periodId",period_id);
        Criterion crit2 = Restrictions.eq("teacherId",teacher_id);
        Criterion crit3 = Restrictions.eq("studentGroupId",student_group_id);
        //remove all periods related to a teacher and a student group
        if (period_id==0){//"May need to edit for multiple periods"
           List<Score>
selectedScoreResultset=session.createCriteria(Score.class).add(Restrictions.and(
crit3, crit2)).list();
           for(Score scores: selectedScoreResultset){
               scores.setSelected(scores.getSelected()+1);
           }
        }
        //remove all teachers related to a period and a student group
        if (teacher_id==0){
           List<Score>
selectedScoreResultset=session.createCriteria(Score.class).add(Restrictions.and(
crit1, crit3)).list();
           for(Score scores: selectedScoreResultset){
               scores.setSelected(scores.getSelected()+1);
           }
        }
        //remove all student groups related to a period and a teacher
        if (student_group_id==0){
           List<Score>
selectedScoreResultset=session.createCriteria(Score.class).add(Restrictions.and(
crit1, crit2)).list();
           for(Score scores: selectedScoreResultset){
               scores.setSelected(scores.getSelected()+1);
```

```
        }
    }
    session.getTransaction().commit();
  }
```

Function MarkScoreAssigned

```
  public static void MarkScoreAssigned(Zeros zeros){
    //make assigned=1  for one score and make unusable=1 for others for the given 'Zeros' or if already unusable make unusable=unusable+1
    Session session = HibernateUtil.getSessionFactory().openSession();
  session.beginTransaction();
    int period_id=zeros.getPeriodId();
    int teacher_id=zeros.getTeacherId();
    int student_group_id=zeros.getStudentGroupId();
    Criterion crit1 = Restrictions.eq("periodId",period_id);
    Criterion crit2 = Restrictions.eq("teacherId",teacher_id);
    Criterion crit3 = Restrictions.eq("studentGroupId",student_group_id);
    Criterion crit4 = Restrictions.eq("tempScore",0);
    Criterion crit5 = Restrictions.eq("assigned",0);
    Criterion crit6 = Restrictions.eq("unusable",0);
    Criterion crit7 = Restrictions.and(crit4, crit5);
    Criterion crit8 = Restrictions.and(crit6, crit7);
    //remove all periods related to a teacher and a student group
    if (period_id==0){
      Criterion crit9 = Restrictions.and(crit2, crit3);
      List<Score> selectedScoreResultset=session.createCriteria(Score.class).add(Restrictions.and(crit8, crit9)).addOrder(Order.asc("selected")).list();
      if (!(selectedScoreResultset.isEmpty())){
```

```
            Score scores=selectedScoreResultset.get(0);
            scores.setAssigned(1);
            session.getTransaction().commit();
            session.close();
            ScoreTable.MarkOtherScoreUnassignable(scores);
         }
      }
      //remove all teachers related to a period and a student group
      if (teacher_id==0){
         Criterion crit9 = Restrictions.and(crit1, crit3);
         List<Score>
selectedScoreResultset=session.createCriteria(Score.class).add(Restrictions.and(
crit8, crit9)).addOrder(Order.asc("selected")).list();
         if (!(selectedScoreResultset.isEmpty())){
            Score scores=selectedScoreResultset.get(0);
            scores.setAssigned(1);
            session.getTransaction().commit();
            session.close();
            ScoreTable.MarkOtherScoreUnassignable(scores);
         }
      }
      //remove all student groups related to a period and a teacher
      if (student_group_id==0){
         Criterion crit9 = Restrictions.and(crit1, crit2);
         List<Score>
selectedScoreResultset=session.createCriteria(Score.class).add(Restrictions.and(
crit8, crit9)).addOrder(Order.asc("selected")).list();
         if (!(selectedScoreResultset.isEmpty())){
            Score scores=selectedScoreResultset.get(0);
```

```
            scores.setAssigned(1);
            session.getTransaction().commit();
            session.close();
            ScoreTable.MarkOtherScoreUnassignable(scores);
        }
    }
  }
```

Function MarkZerosAssigned

```
public static void MarkZerosAssigned(){
    //arrange those  above(both student groups and teachers)  in degrading order and set selected while removing the selected zeros from no of zeros column by calling ScoreTable.UpdateZeroCounter();
        System.out.println("*** Starting Method:MarkZerosSelected");
    int no_of_unassigned_and_usable_zeros_rows=1;
    Session session = HibernateUtil.getSessionFactory().openSession();
    while(no_of_unassigned_and_usable_zeros_rows>0){
        try{
            session.beginTransaction();
            Criterion crit1 =Restrictions.eq("assigned",0);
            Criterion crit2 =Restrictions.ne("noOfZerosTemp", 0);
            Criterion crit3 =Restrictions.eq("unusable", 0);
            Criterion crit4 =Restrictions.and(crit1, crit2);
            Criterion crit5 =Restrictions.and(crit3, crit4);
            List<Zeros>
zerosResultset=session.createCriteria(Zeros.class).add(crit5).addOrder(
Order.asc("noOfZerosTemp") ).list();
            no_of_unassigned_and_usable_zeros_rows=zerosResultset.size();
            System.out.println("***
```

```
no_of_unassigned_no_of_zeros_nonzero_rows="+no_of_unassigned_and_usable
_zeros_rows);
            //get first 'zeros'
            if(no_of_unassigned_and_usable_zeros_rows==0){
               break;
            }
            Zeros zeros=zerosResultset.get(0);
            System.out.println("*** Getting Zeros Selected Id "+zeros.getId()+"
No of Zeros"+zeros.getNoOfZeros());
            //mark selected for that 'zeros'
            zeros.setAssigned(1);//set assigned=1
            zeros.setUnusable(1);//set unusable=1
            //"May need to edit for correct Updates"
            session.getTransaction().commit();
            //mark selected for scores that were covered by the perticular zeros.
            ScoreTable.MarkScoreAssigned(zeros);
            //recount for maximum no of covered/uncovered zero scores
            //not making any adjustment to SELECTED column in Zeros table and
update noOfZerosTemp in Zeros table by counting score tempScore=0 and
selected=0 in Scores table
            ScoreTable.UpdateZeroCounter("unusable");// refreshing the
Zerocounter to detect the changes done
            no_of_unassigned_and_usable_zeros_rows--;          //go to next
zeros`
         }
         catch (HibernateException he) {
            he.printStackTrace();
         }
      }
      session.close();
```

```
    System.out.println("*** Terminating MarkZerosSelected");
  }
```

Function MarkOtherScoreUnassignable

```
  public static void MarkOtherScoreUnassignable(Score score){
        Session session = HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        Criterion crit1 = Restrictions.eq("teacherId",score.getTeacherId());
        Criterion crit2 = Restrictions.eq("studentGroupId",score.getStudentGroupId());
         Criterion crit3 = Restrictions.eq("periodId",score.getPeriodId());
         Criterion crit4 = Restrictions.and(crit1, crit3);
        Criterion crit5 = Restrictions.and(crit2, crit3);
         Criterion crit6 = Restrictions.or(crit4, crit5);
         List<Score> selectedScoreResultset=session.createCriteria(Score.class).add(crit6).list();
         for(Score otherScore: selectedScoreResultset){
                otherScore.setUnusable(otherScore.getUnusable()+1);
//"May need to edit for correct Updates"
        }
        session.getTransaction().commit();
        session.getTransaction().begin();
        crit4 = Restrictions.and(crit1, crit2);       //"May need to edit for multiple periods"
        selectedScoreResultset=session.createCriteria(Score.class).add(crit4).list();
        for(Score otherScore: selectedScoreResultset){
                otherScore.setUnusable(otherScore.getUnusable()+1);
        }
```

```
        session.getTransaction().commit();
        session.close();
}
```

APPENDIX B-SOFTWARE IMPLEMENTATIONS FOR THE HUNGARIAN ALGORITHM

C implementation

https://github.com/maandree/hungarian-algorithm-n3/blob/master/hungarian.c
https://launchpad.net/lib-bipartite-match

Java/JavaScript/Java Applet implementation

https://github.com/KevinStern/software-and-algorithms/blob/master/src/main/java/blogspot/software_and_algorithms/stern_library/optimization/HungarianAlgorithm.java
http://konstantinosnedas.com/dev/soft/munkres.htm
http://web.axelero.hu/szilardandras/gaps.html
http://timefinder.svn.sourceforge.net/viewvc/timefinder/trunk/timefinder-algo/src/main/java/de/timefinder/algo/roomassignment/
http://twofourone.blogspot.com/2009/01/hungarian-algorithm-in-javascript.html

Python implementation

http://software.clapper.org/munkres/
https://github.com/xtof-durr/makeSimple/blob/master/Munkres/kuhnMunkres.py

C# implementation

http://noldorin.com/blog/2009/09/hungarian-algorithm-in-csharp/

Implementation in Matlab

http://www.mathworks.com/matlabcentral/fileexchange/11609

C++ implementation (multi-functional bipartite graph version)

http://students.cse.tamu.edu/lantao/codes/codes.php
http://saebyn.info/2007/05/22/munkres-code-v2/
http://dlib.net/optimization.html#max_cost_assignment
http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=hungarianAlgorithm