ITM-411
Intermediate Software Development
Final Project

Project Summary:

A Java SE Netbeans project is created which is comprised of a graphical user interface components from the Swing and AWT api's for performing the functionalities such as creating the database table, inserting records into the table using csv file data given, retrieve the records, update the record, delete the record and also perform serialization, deserialization, compute data analytics and write the data to a file. When user perform any functionalities the result of the functionality performed is displayed on GUI.

Installation, Compile and Run Time requirements:

1. NetBeans IDE 7.2.1
2. Java 1.7.0_11, Java Hotspot(TM) 64-Bit Server VM 23.6-b04
3. Windows 7 version 6.1 running on amd64
4. MySQL – and the jar "mysql-connector-java5.1.22-bin.jar"

Insights, Expected results, and Challenges:

Insights:

The project helped me get hands on creating graphical user interface using swing components and AWT APIs and providing back end code to perform the required operation selected by the user using GUI.

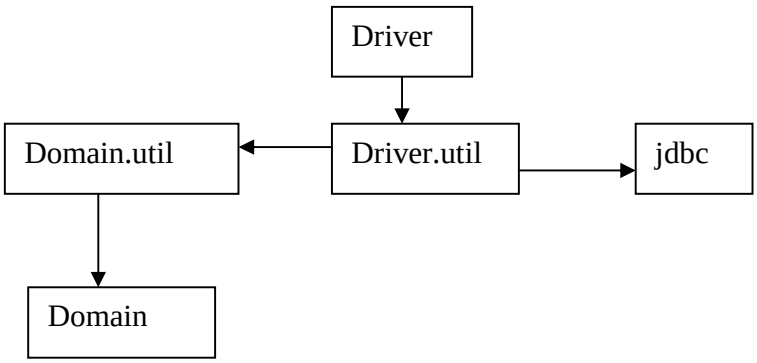
To perform the required functionalities I have built the graphical user interface which comprises of Jframe, Jbutton, Jtable, JLabel, JtextField, Jpanel etc. For user to select the required functionality to perform I have provided the JButton naming the functionality on that. So, when user clicks the required button respective code is triggered in the back and required operation is performed and status of the operation performed using JLabel.

When the user retrieves the records from the database the result is displayed using JTable. When the data analytics is performed the result of those are displayed on GUI using JTextField. JPanel is used to provide the look and feel to the GUI and aggregate operations in one block. All these components are built on JFrame which provides the user option to close the application whenever required and also to minimize and maximize the GUI which in turn adjusts the inner components accordingly.

The packages used in this project are:

1. domain
2. domain.util
3. driver.util
4. jdbc
5. driver

The data flow diagram for these packages is as shown below:



domain package consists of the following classes:

i. Record
ii. DaylightRecord
iii. PersistentObject
iv. DataAnalyticsResult
v. WinterSolstice
vi. SummerSolstice
vii. VernalandAutumnalEquinox

i. Record Superclass:

This class is an abstract super class. This is the parent class and it performs the following Operations:

· Import the required packages.
· Declaration of variables.
· Provide No-arguments constructor with default values.
· Provide Full-arguments constructor.
· Provide accessors and mutators for the variables declared.
· Provide toString() method.
· Implement Serializable.

ii. DaylightRecord Subclass:

This class is a subclass which does the following operations:

· Import the required package
· Inherit the parent class i.e., the abstract super class 'Record'

iii. DataAnalyticsResult :

This class is used to store data after various data analytics computation and does the following operations:

· Import the required packages.
· Declaration of variables.

iv. WinterSolstice :

This class is used for winter solstice computation as comparator and does the following functions:

· Implements comparator.
· Compares the values of datafields in the given objects data.

v. SummerSolstice:
This class is used for the summer solstice computation as comparator and does the following functions:
· Implements comparator.
· Compares the values of datafields in the given objects data.

vi. VernalandAutumnalEquinox:
This class is used for the spring and fall equinox computation as comparator and does the following functions:
· Implements comparator.
· Compares the values of datafields in the given objects data.

2) **domain.util** package consists of the following class:

i. Utilities :

This class has methods to read ".csv" file, serialization, deserialization, create ".csv" file, write ".csv" file back to the system, compute data analytics to retrieve shortest day in winter , longest day summer solstice and equal day and night in equinox. The functionalities of all the methods are as described below:

□ *readCSVfile() :*

In this method, I create the instance of DaylightRecord and add the data into an arraylist named daylightRecords.

□ *serializeListObject(PersistantObject pobj):*

In this method, persistent object in which date and arraylist daylightRecords encapsulated are serialized to daylight-record.ser.

□ *deserializeListObject():*

In this method, the serialized data is deserialized back into app.

□ *createCSVfile():*

In this method, deserialized data is read, formatted (ie to insert ',','\n') as required and stored accordingly into a stringbuilder..

□ *writeCSVfile(StringBuilder sb):*

In this method, the stringbuilder data which is the result of createCSVfile() method is read and a file deserializedDaylightRecords.csv is created.

□ *findCorrespondingData(List daylightRecords, String startDate, String endDate):*

This method is used to compute data analytics for winter and summer solstice months. It takes daylightRecords, startDate and endDate as parameters. startDate will be the winter Solstice's or Summer Solstice's starting date and endDate will be the winter solstice or summer solstice ending dates respectively. The function returns the list of all computed result of all the days in winter and summer.

□ *findRelevantData(List daylightRecords, String startDate, String endDate):*

This method is used to compute data analytics for spring and fall equinox months. It take daylightRecords and startDate and endDate as parameters. startDate will be the spring Equinox or Fall Equinox starting date and endDate will be the Sping Equinox's or Fall Equinox's ending dates respectively. The function returns the list of all computed result for all the days in spring and fall

□ *computeSolsticeDataAnalytics(Object o, String startDayofSolstice, String endDayofSolstice):*

This method is mainly used to find the difference between sunrise and sunset for the days in winter and summer solstice. This method is called from findCorrespondingData(List day…..,String …, String ..), and takes object, startDate and endDate as parameter.

- *computeEquinoxDataAnalytics(Object o1, Object o2, String equinoxStartDate, String equinoxEndDate)*:

This method is mainly used to find the difference between sunrise and sunset of a day, and then subtract it from 24hrs, for all days coming in fall and spring equinox. This method is called from findRelevantData(List day…..,String …, String ..), and takes object, startDate and endDate as parameter.

- *retrieveDataFromComputation(List newList, List dataList):*

This method is used to retrieve the final results of all the data analytics computed ie. the shortest day of winter, longest day of summer and equal day and night for spring and fall equinox.

3) **jdbc** package consists of the following class:

i. JDBCUtilities :

This class contains all the functions related to database. The functions implemented and its functionalities in this class are as follows:

- *getConnection() :*

This function establishes the connection with the database itm411db. This function is called before performing any required CRUD operation with the database.
initByCreatingandPopulatingTableInDatabase(…) :
This function is called to create table in the database and insert all the daylight records read from the CSV file into the created table. This function is called everytime the application is launched.

- *CreateRecord(..):*

This method is called whenever the user wants to insert an daylight record into database. This function is invoked when button ‘Ok’ is pressed from the popped up Jframe which will be created and displayed after pressing Create button from MainGUI. All the daylight record data entered to insert in the database should be of required data type otherwise the appropriate error message is returned. On success inserts the record into database and success message is returned.

- *retrieveRecordForDay (..):*

This method is called whenever the user wants to retrieve a required day record. This function is invoked when button ‘Retrieve record by day’ is pressed from GUI. The user must enter the day value in the textfield provided and an optional month value. The day value is then passed to this function. If the entered day value is valid then respective day light record data is retrieved and returned.

- *retrieveRecordBetweenDayRange(…):*

This method is called whenever the user wants to retrieve a required day record between two day ranges. This function is invoked when ‘Retrieve record by range’ button is pressed from GUI. The user must enter the begin day and end day value in the respective textfield provided and an optional month value. The days value is then

passed to this function. If the entered days value is valid then respective daylight records data is retrieved and returned.

- *retrieveAllRecords() :*

This method is used to retrieve all the daylights' records from the database. This function is invoked when button 'Retrieve all' is pressed from the GUI. If the database table is not empty then all day light record present in the database is retrieved and returned.

- *updateRecord():*

This method is called whenever the user wants to update a daylight record in database. This function is invoked when button 'Ok' is pressed from the popped up Jframe which is displayed after pressing 'Update' button from MainGUI. All the daylight record data entered to update record in the database should be of required data type otherwise the appropriate error message is returned. On success record is updated into the database and success message is returned.

- *deleteRecord(..):*

This method is called whenever the user wants to delete a required daylight record. This function is invoked when 'Delete' button is pressed from the GUI. The user should either enter the day value or month value or both according to requirement to delete the record. If the entered values are valid then respective record is deleted and successful message is returned else error message is returned.

4) **driver.util** package consists of the following class:

i. GUIUtilities:

This class is used to implement methods which will be called from driver class. This class helps in minimizing driver class methods and in understanding the main class in a better and easy way. The class contains the following functions:

- *createRecordJFrame(..):*

This function is called when 'Create' button is pressed from the GUI. Function creates the JFrame and components Jlabel(for day, month, sunrise and sunset), JTextfield(for day month, sunrise and sunset) to accept values for the colums present in the database. If the entered values are valid then CreateRecord(..) function is called to insert values into the database. Upon success or failure it displays the message on the status label.

- *updateRecordJFrame(…):*

This function is called when 'Update' button is pressed from the GUI. Function creates the JFrame and components Jlabel(for day, month, sunrise and sunset), JTextfield(for day month, sunrise and sunset) to accept values for the colums present in the database. If the entered values are valid then updateRecord (..) function is called to update values into the database. Upon success or failure it displays the message on the status label.

- *serializeandDisplaytheStatusOnGUI(..):*

This function is called when 'Serialize' button is pressed from the GUI. Function serializes the daylight records and current date into persistent object and displays the success or failure message on the status label.

- *writeDeserializedDatatofile(..):*

This function is called when 'Deserialize' button is pressed from GUI. After deserializing the object,this function is called to write deserialized data into the file. If any error occurs respective message is displayed on the status label.

- *computeDataAnalytics(…):*

This function is called when 'Execute' button is pressed from GUI. Function computes the data analytics and displays the result in the respective data analytics text field (Winter, Summer, Vernal, Autumn text fields).

- *retrieveAllDatabaseRecords(…):*

This function is called when 'Retrieve All' button is pressed from GUI. Function invokes the retrieveAllRecords(..) which retrieves the data and the obtained result is displayed on JTable. If success or any error occurs then respective error message is displayed in status label.

- *retrieveRecordByDay(…):*

This function is called when 'Retrieve By Day' button is pressed from GUI. Function gets the day value, month value and invokes the retrieveRecordForDay (..) which retrieves the data and obtained result is displayed on JTable. If success or any error occurs then the respective message is displayed in status label.

- *retrieveRecordByDayRange(…):*

This function is called when 'Retrieve By Range' button is pressed from GUI. Function gets the day value, month value and invokes the retrieveRecordBetweenDayRange (..) which retrieves the data between two day ranges and returns the same, obtained result is displayed on JTable. If success or any error occurs then the respective message is displayed in status label.

5) **driver** package consists of the following class:

i. MainFrame:

This is the main class where the program execution starts and displays the GUI. In this class the GUI components events are created. The events are triggered when respective component is selected from GUI. The class contains the following functions:

- WriteToFileButtonActionPerformed(…):

This function is triggered when 'Write To File' button is pressed from the displayed GUI.

- readCsvButtonActionPerformed(…):

This function is triggered when 'Read CSV' button is pressed from the displayed GUI.

- serializeButtonActionPerformed(…):

This function is triggered when 'Serialize' button is pressed from the displayed GUI.

- deserializeButtonActionPerformed(…):

This function is triggered when 'Deserialize' button is pressed from the displayed GUI.

- executeButtonActionPerformed(…):

This function is triggered when 'Execute' button is pressed from the displayed GUI.

- CreateRecordButtonActionPerformed(…):

This function is triggered when 'Create' button is pressed from the displayed GUI.

- RetrieveAllRecordButtonActionPerformed(…):

This function is triggered when 'Retrieve All' button is pressed from the displayed GUI.

- RetrieveRecordByDayButtonActionPerformed(…):

This function is triggered when 'Retrieve By Day' button is pressed from the displayed GUI.

- RetrieveRecordByRangeButtonActionPerformed(…):

This function is triggered when 'Retrieve By Range' button is pressed from the displayed GUI.

- UpdateRecordButtonActionPerformed(…):

This function is triggered when 'Update' button is pressed from the displayed GUI.

- DeleteRecordButtonActionPerformed(…):

This function is triggered when 'Delete' button is pressed from the displayed GUI.

- displayTimeButtonActionPerformed(…):

This function is triggered when 'Della Time' button is pressed from the displayed GUI.

Expected Results:

Expected Results	*Test Result*
1. Use a MySQL database called itm411DB for all of the record data.	**Pass**
2. Create the single database table with fields from the CSV file	**Pass**
3. Provide the capability of re-initializing the table data if the user wishes to start over.	**Pass**
4. Provide GUI components to trigger read csv file,serialization and deserialization, data analytics and to write output to file(MP2 functions)	**Pass**
5. To display status on GUI for all user interaction performed by the user	**Pass**
6. Manipulate DayLightRecord objects from the GUI using database CRUD (Create, Retrieve, Update and Delete) JDBC operations.	**Pass**
7. Display operation results in either tabular, text field or text area formats on GUI	**Pass**

Note:

1. Database created in MySQL - itm411db,

UserName – root

Password – admin.

2. While entering input values for sunrise and sunset to create and update the record the format should be in HHmm format and length should be of 4. Ex. If 456 is written then error message is displayed. User has to give input as 0456. If nothing is given as input to sunrise and sunset values while creating or updating then 0000 and 0000 values are inserted or updated to respective record in the database.

Challenges Faced:
While developing the user interface, the preview of the user would be correct but when the project is initiated it appeared differently from the built one. However, it was resolved when I adjusted the GUI for the proper display.

While computing data analytics I was confused whether to consider csv data or database data. Then finally I provided the feature for both options. The user has to decide on which data is needed the data analytics result.

Program Simulation Steps:

The following steps are to be followed to get proper simulation results. This gives an idea about the program simulation to the user regarding the input fields and what inputs the user must provide:

1. Every time when the program execution starts it creates the table daylightrecord in database itm411db with day, month, sunrise and sunset as columns where day and month are primary keys and inserts data from given CSV file.
2. **Read CSV** Button: This button is used to read csv data and store it in a list. Before reading the file if user try to perform persistent operations (serialization, deserialization, display time difference between serialization and deserialization) and data analytics operations then error message is displayed.
3. **Serialize Button**: This button is used to serialize the persistent object as an aggregate.
4. **Deserialize Button**: This button is used to deserialize the serialized data back into app. If user tries to perform deserialize before serialization then the error message is displayed.
5. **Delta Time Button**: This button is used to display the time difference between serialization and deserialization. If the user tries to perform this before doing serialization and deserialization then error message is displayed.
6. **Execute Button** with two **radio buttons option** to chose date source either from CSV or from database: This button is used to perform and display the data analytics result. If the data from csv option is selected then data analytics operations are performed for csv file data. If database option is selected then data analytics is performed for database data. All the computed data analytics results are displayed in appropriate text fields provided.
7. **Write Data to file**: This button is used to write the read daylightrecords to a file. If the user tries to perform this before read csv file then error message is displayed.
8. **Create** Button: This button is used to create a daylightrecord and insert into the database. When button is clicked a window pops up asking for the input values which is used to create record. Since the data is for the year 2013 the validation is used for not to insert record for 29th , 30th, 31st of February, 30th of April, June, September , and November. The values entered by the user is validated and then inserted into database. If the connection to MySQL sever is lost then appropriate message is displayed and record is not inserted.

9. **Retrieve All** Button: This button is used to retrieve all the records present in the database and Displayed on JTable. If the connection to MySQL sever is lost then appropriate message is displayed and records are not retrieved.
10. **Retrieve By Day** Button: This button is used to retrieve particular record(s) from the database. This has two input fields:
    - Day: Day value is mandatory. User has to input the day value to retrieve record if not error message will be displayed. If user enters a particular day with no month selected then records of day entered from all months are displayed.
    - Month: Month is optional. User can leave this non-selected. If selected with day input then a day record of that particular month is displayed.

    If the connection to MySQL sever is lost then appropriate message is displayed and record is not retrieved.
11. **Retrieve By Range** Button: This button is used to retrieve a particular record(s) from the database between two day ranges. This has three input fields:
    - Begin Day:
    - End Day: Begin Day and End value is mandatory. User has to input the Begin day and End day value to retrieve record between two day ranges otherwise error message will be displayed. If user enters a begin and end day with no month selected then records between the days enterted from all months are displayed.
    - Month: Month is optional. User can leave this non-selected. If selected with begin and end day input then records between those two days of particular month are displayed.

    If the connection to MySQL sever is lost then appropriate message is displayed and record is not retrieved.
12. **Update Button**: This button is used to update a daylightrecord present in the database. When button is clicked a window pops asking for the input values which is used to update record. Since the data is for the year 2013 the validation is used for not to update record for 29th , 30th, 31st of February, 30th of April, June, September , and November because those record doesn't exist in database. The values entered by the user are validated and then respective record is updated in database. If the connection to MySQL sever is lost then appropriate message is displayed and record is not inserted.
13. **Delete Button**: This button is used to delete a particular record(s) in the database. This has two input fields:
    - Day: If only day value is entered then all the records for that particular day from all months are deleted.
    - Month: If only month value is entered then all the records for that particular month is deleted.
    - If both month and day is selected then only one record with day value of particular month is deleted.

    If the connection to MySQL sever is lost then appropriate message is displayed and record is not retrieved.
14. **Reset All Input Fields** Button: This button is used to reset all the input fields present in the GUI. This is used to help user to erase all the input values he entered to perform required operation saving his time.

15. **Reload Database** Button: After performing CRUD operations, if the user wants the original values in the database, clicking this button would load the original data from CSV file into the database. Anyhow, if the user exits the application and restart it then the database is automatically loaded with CSV file data.
16. **Status** Label: This is used to display all the user operational messages .

Please refer to the Screenshots (output section) below for some of the simulation examples.

Screenshots:

1. Record superclass:

Snapshot 1.1:

```
 */
public abstract class Record implements Serializable {


    private String day;
    private Date sunRise;
    private Date sunSet;

    //No argument Constructor
    public Record() {
        day = "noday";
        sunRise = null;
        sunSet = null;
    }

    //full argument Constructor
    public Record(String day, Date sunRise, Date sunSet) {
        this.day = day;
        this.sunRise = sunRise;
        this.sunSet = sunSet;
    }

    //Accessors and Mutators
    public String getDay() {
        return day;
```

Record superclass which is made serializable and have fields for storing data read from data files. no-arg and full-arg constructors are implemented for fields

Snapshot 1.2

```
public void setDay(String day) {
    this.day = day;
}

public Date getSunRise() {
    return sunRise;
}

public void setSunRise(Date sunRise) {
    this.sunRise = sunRise;
}

public Date getSunSet() {
    return sunSet;
}

public void setSunSet(Date sunSet) {
    this.sunSet = sunSet;
}

@Override
public String toString() {
    return "Record{" + "day=" + day + ", sunRise=" + df.format(sunRise) + ", sunSet=" + df.format(sunSet) + '}';
}
```

Record superclass getters, setters, and toStringmethod()

2. DaylightRecord subclass

Snapshot 2.1:

```
5   package domain;
6
7   import java.io.Serializable;
8   import java.util.Date;
9
10  /**
11   * @author Meghashree M Ramachandra
12   */
13  public class DaylightRecord extends Record implements Serializable {
14
15      //No argument Constructor
16      public DaylightRecord() {
17          super();
18      }
19
20      //full argument constructor
21      public DaylightRecord(String day, Date sunRise, Date sunSet) {
22          super(day, sunRise, sunSet);
23      }
24
25      @Override
26      public String toString() {
27          return "DayLightRecord{" + super.toString() + '}';
28      }
29  }
```

DaylightRecord subclass with no-arg and full-arg contructors, accessors, mutators and toString() method

3. PersistentObject class

Snapshot 3.1:

```
public class PersistentObject implements Serializable {
    private Date dateObj;
    private List<DaylightRecord> dayLightRecord;

    //No argument constructor
    public PersistentObject() {
        dateObj = null;
        dayLightRecord = null;
    }

    //Full argument constructor
    public PersistentObject(Date dateObj, List<DaylightRecord> dayLightRecord) {
        this.dateObj = dateObj;
        this.dayLightRecord = dayLightRecord;

    }

    public Date getDateObj() {
        return dateObj;
    }

    public void setDateObj(Date dateObj) {
        this.dateObj = dateObj;
    }

    public List<DaylightRecord> getDayLightRecord() {
        return dayLightRecord;
```

PersistentObject class with Date and arraylist objects encapsulated, which is made serializable. no-arg, full-arg constructors, getter and setter method implemented

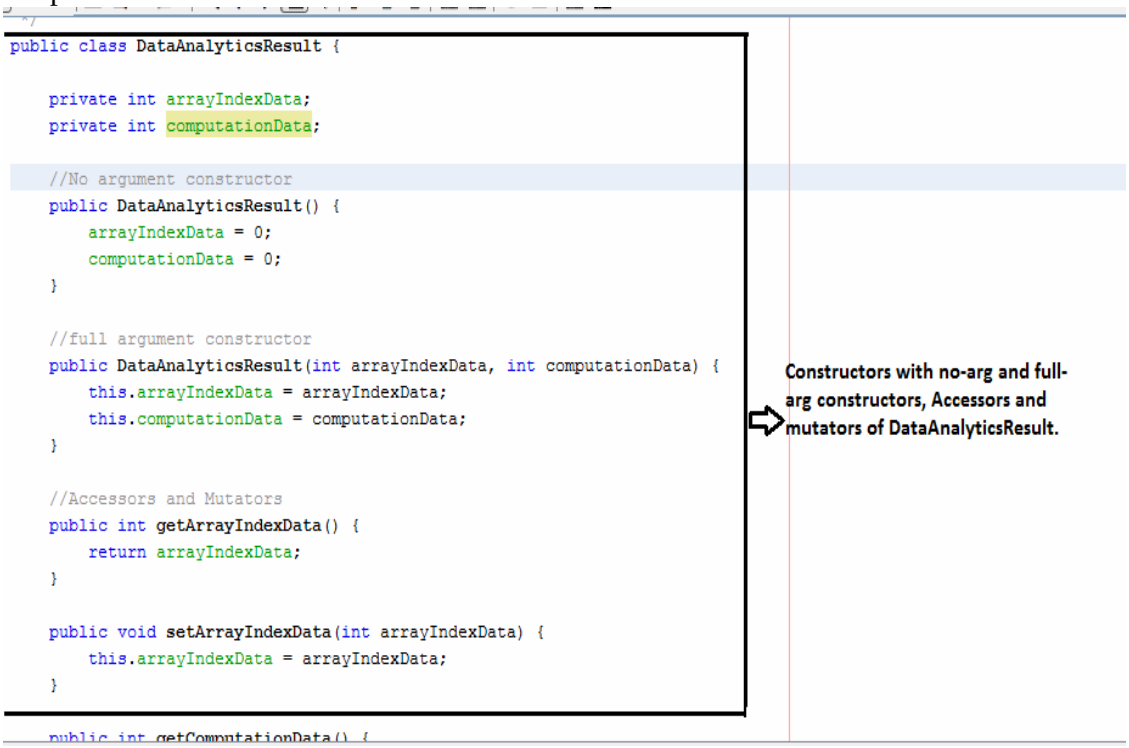
Snapshot 3.2:

```
    public void setDateObj(Date dateObj) {
        this.dateObj = dateObj;
    }

    public List<DaylightRecord> getDayLightRecord() {
        return dayLightRecord;
    }

    public void setDayLightRecord(List<DaylightRecord> dayLightRecord) {
        this.dayLightRecord = dayLightRecord;
    }

    @Override
    public String toString() {
        return "PersistentObject{" + "dateObj=" + dateObj + ", dayLightRecord=" + dayLightRecord + '}';
    }
}
```

PersistentObject class setters, getters and toString() method.

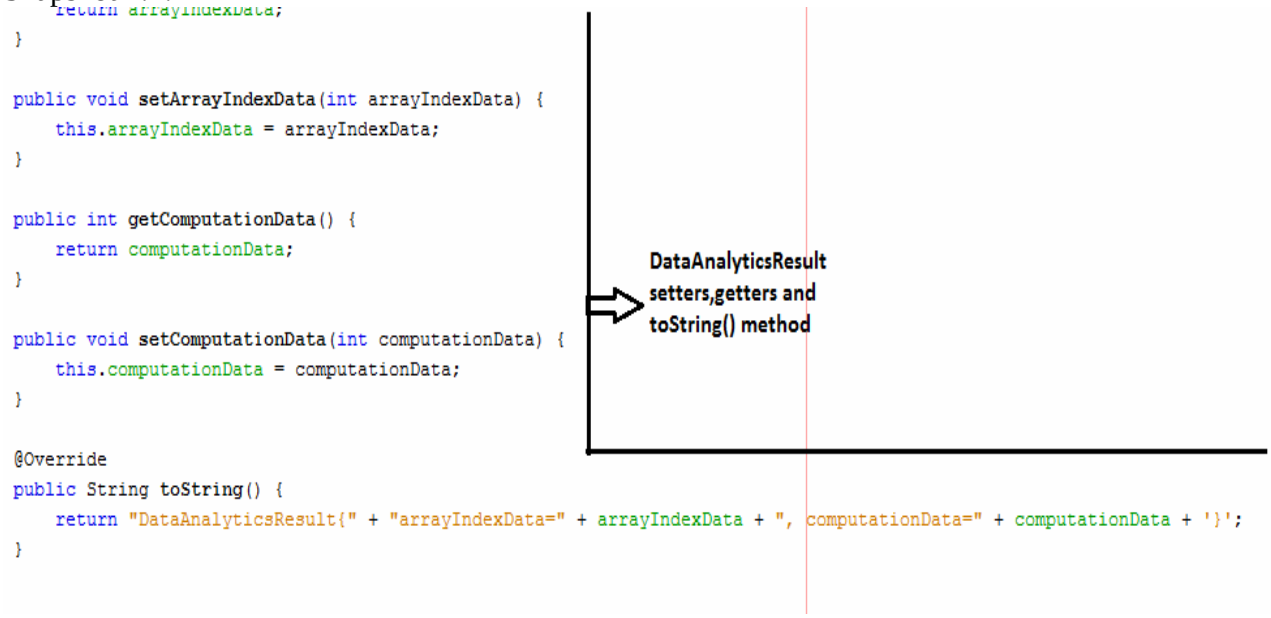
4. DataAnalyticsResult class:

Snapshot 4.1:




```
public class DataAnalyticsResult {

    private int arrayIndexData;
    private int computationData;

    //No argument constructor
    public DataAnalyticsResult() {
        arrayIndexData = 0;
        computationData = 0;
    }

    //full argument constructor
    public DataAnalyticsResult(int arrayIndexData, int computationData) {
        this.arrayIndexData = arrayIndexData;
        this.computationData = computationData;
    }

    //Accessors and Mutators
    public int getArrayIndexData() {
        return arrayIndexData;
    }

    public void setArrayIndexData(int arrayIndexData) {
        this.arrayIndexData = arrayIndexData;
    }

    public int getComputationData() {
```

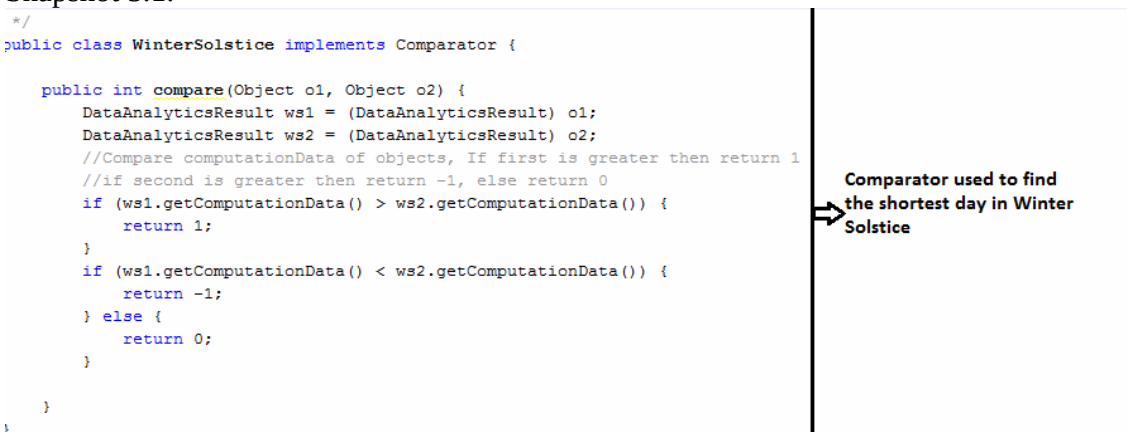
Snapshot 4.2:




```
}

public void setArrayIndexData(int arrayIndexData) {
    this.arrayIndexData = arrayIndexData;
}

public int getComputationData() {
    return computationData;
}

public void setComputationData(int computationData) {
    this.computationData = computationData;
}

@Override
public String toString() {
    return "DataAnalyticsResult{" + "arrayIndexData=" + arrayIndexData + ", computationData=" + computationData + '}';
}
```

4. WinterSolstice Comparator

Snapshot 5.1:

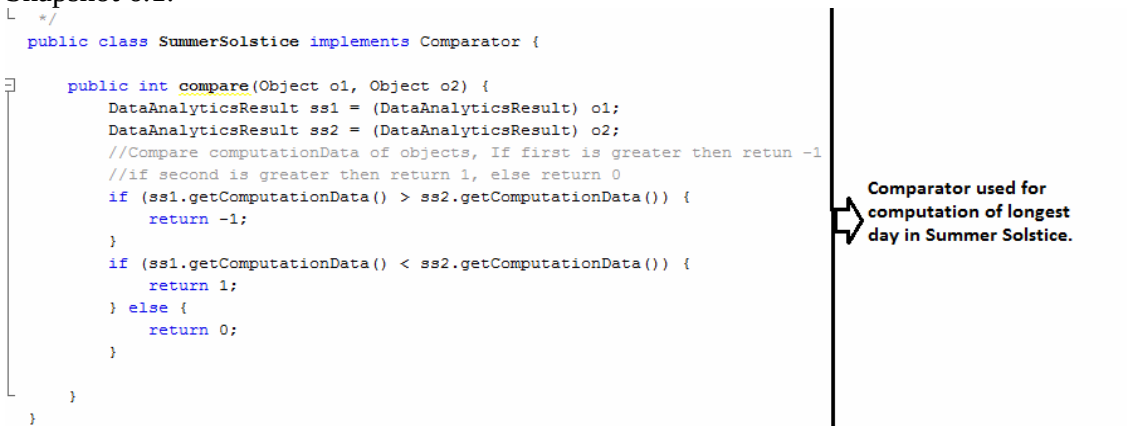


```
*/
public class WinterSolstice implements Comparator {

    public int compare(Object o1, Object o2) {
        DataAnalyticsResult ws1 = (DataAnalyticsResult) o1;
        DataAnalyticsResult ws2 = (DataAnalyticsResult) o2;
        //Compare computationData of objects, If first is greater then return 1
        //if second is greater then return -1, else return 0
        if (ws1.getComputationData() > ws2.getComputationData()) {
            return 1;
        }
        if (ws1.getComputationData() < ws2.getComputationData()) {
            return -1;
        } else {
            return 0;
        }

    }
```

Comparator used to find the shortest day in Winter Solstice

6. SummerSolstice Comparator

Snapshot 6.1:

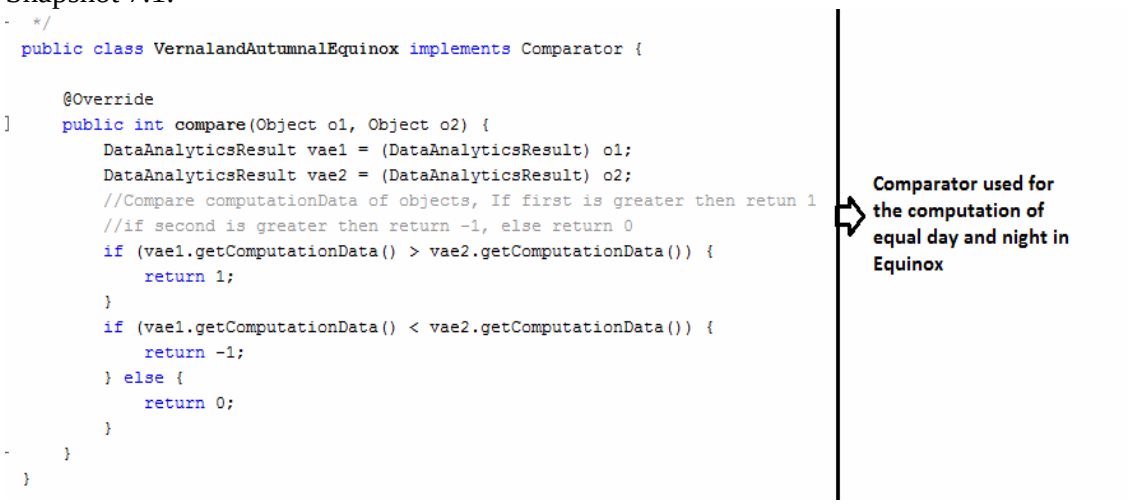


```
*/
public class SummerSolstice implements Comparator {

    public int compare(Object o1, Object o2) {
        DataAnalyticsResult ss1 = (DataAnalyticsResult) o1;
        DataAnalyticsResult ss2 = (DataAnalyticsResult) o2;
        //Compare computationData of objects, If first is greater then retun -1
        //if second is greater then return 1, else return 0
        if (ss1.getComputationData() > ss2.getComputationData()) {
            return -1;
        }
        if (ss1.getComputationData() < ss2.getComputationData()) {
            return 1;
        } else {
            return 0;
        }

    }
}
```

Comparator used for computation of longest day in Summer Solstice.

8. VernalandAutumnal Comparator

Snapshot 7.1:



```
*/
public class VernalandAutumnalEquinox implements Comparator {

    @Override
    public int compare(Object o1, Object o2) {
        DataAnalyticsResult vae1 = (DataAnalyticsResult) o1;
        DataAnalyticsResult vae2 = (DataAnalyticsResult) o2;
        //Compare computationData of objects, If first is greater then retun 1
        //if second is greater then return -1, else return 0
        if (vae1.getComputationData() > vae2.getComputationData()) {
            return 1;
        }
        if (vae1.getComputationData() < vae2.getComputationData()) {
            return -1;
        } else {
            return 0;
        }
    }
}
```

Comparator used for the computation of equal day and night in Equinox

9. Utilities class:

Snapshot 8.1:

```
 *
 * @author Meghashree M Ramachandra
 */
public class Utilities {

    //to read data from file
    public static List readCSVfile() {
        BufferedReader br;
        String line;
        DaylightRecord dlrData;
        DateFormat format = new SimpleDateFormat("MM/dd/yyyy HHmm");
        List daylightRecords = new ArrayList<DaylightRecord>();
        int month;

        try {
            // Read data in from CSV file...
            br = new BufferedReader(new FileReader("data/sunrise-sunset.csv"));
            String dateSunrise;
            String dateSunset;
            while ((line = br.readLine()) != null) {
                // Extract tokens from CSV line with comma delimiter...
                month = 1;
                StringTokenizer st = new StringTokenizer(line, ",");
```

Function to read data from files and store in arraylist

Snapshot 8.2:

```
            String sunRiseData = st.nextToken();
            String sunSetData = st.nextToken();
            if (sunRiseData.length() == 3) {
                sunRiseData = "0" + sunRiseData;
            }
            if (sunSetData.length() == 3) {
                sunSetData = "0" + sunSetData;
            }
            dateSunrise = Integer.toString(month) + "/" + day + "/2013 " + sunRiseData;
            dateSunset = Integer.toString(month) + "/" + day + "/2013 " + sunSetData;
            if ((day.equals("29") || day.equals("30")) && month == 2) {
                //check for month which don't have day 29 and 30 in the year 2013.
                daylightRecords.add(null);
                dateSunrise = Integer.toString(++month) + "/" + day + "/2013 " + sunRiseData;
                dateSunset = Integer.toString(month) + "/" + day + "/2013 " + sunSetData;
                dlrData = new DaylightRecord(day, months[++i], format.parse(dateSunrise), format.parse(dateSunset));
                daylightRecords.add(dlrData);

            } else if ((day.equals("31")) && (month == 2 || month == 4 || month == 6 || month == 9 || month == 11)) {
                //check for months which don't have day 31 in the year 2013.
                daylightRecords.add(null);
                dateSunrise = Integer.toString(++month) + "/" + day + "/2013 " + sunRiseData;
                dateSunset = Integer.toString(month) + "/" + day + "/2013 " + sunSetData;
                dlrData = new DaylightRecord(day, months[++i], format.parse(dateSunrise), format.parse(dateSunset));
                daylightRecords.add(dlrData);

            } else {
                //create daylight record object and insert into list.
                dlrData = new DaylightRecord(day, months[i], format.parse(dateSunrise), format.parse(dateSunset));
```

code snippet to read data from file and store in an arraylist

Snapshot 8.3:

```
//to get day light records data from the database.
public static List databaseDataRecords() {
    DaylightRecord dlrData;
    DateFormat format = new SimpleDateFormat("MM/dd/yyyy HHmm");
    DateFormat df = new SimpleDateFormat("HHmm");
    List<String> months = Arrays.asList("January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November",
    List daylightRecords = null;
    String dateSunrise;
    String dateSunset;
    // Retrieves all records from database stored in list.
    List<DaylightRecord> retrievedRecordList = JDBCUtilities.retrieveAllRecords();
    // if retrieved data present in the list.
    if (retrievedRecordList.size() > 0 && retrievedRecordList != null) {
        daylightRecords = new ArrayList<DaylightRecord>();
        for (DaylightRecord dlr : retrievedRecordList) {
            String sunRiseData = df.format(dlr.getSunRise());
            String sunSetData = df.format(dlr.getSunSet());
            if (sunRiseData.length() == 3) {
                sunRiseData = "0" + sunRiseData;
            }
            if (sunSetData.length() == 3) {
                sunSetData = "0" + sunSetData;
            }
            dateSunrise = (months.indexOf(dlr.getMonth()) + 1) + "/" + dlr.getDay() + "/2013 " + sunRiseData;
            dateSunset = (months.indexOf(dlr.getMonth()) + 1) + "/" + dlr.getDay() + "/2013 " + sunSetData;
```

code snippet to read data from database to perform data analytics.

Snapshot 8.4:

```
            dateSunrise = (months.indexOf(dlr.getMonth()) + 1) + "/" + dlr.getDay() + "/2013 " + sunRiseData;
            dateSunset = (months.indexOf(dlr.getMonth()) + 1) + "/" + dlr.getDay() + "/2013 " + sunSetData;
            try {
                dlrData = new DaylightRecord(dlr.getDay(), dlr.getMonth(), format.parse(dateSunrise), format.parse(dateSunset));
                daylightRecords.add(dlrData);
            } catch (ParseException ex) {
                Logger.getLogger(Utilities.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
    return daylightRecords;
}
```

code to retrieve data from database continued...

Snapshot 8.5:

```
//to serialize
public static void serializeListObject(PersistentObject pObject) {
    ObjectOutputStream oos;
    try {
        oos = new ObjectOutputStream(new FileOutputStream("data/daylight-record.ser"));
        oos.writeObject(pObject);
    } catch (IOException ex) {
        Logger.getLogger(Utilities.class.getName()).log(Level.SEVERE, null, ex);
    }
}

//to deserialize
public static PersistentObject deserializeListObject() {
    ObjectInputStream ois = null;
    PersistentObject pObject = new PersistentObject();
    try {
        // Serialize records as a aggregate...
        ois = new ObjectInputStream(new FileInputStream("data/daylight-record.ser"));
        pObject = (PersistentObject) ois.readObject();
    } catch (ClassNotFoundException | IOException ex) {
        Logger.getLogger(Utilities.class.getName()).log(Level.SEVERE, null, ex);
    }
```

code snippet to serialize and deserialize the PersistentObject

Snapshot 8.6:

```
//make app sleep for 10 seconds
public static void makeAppSleep() {
    try {
        System.out.println("Waiting 10 seconds...");
        Thread.sleep(10000);
    } catch (InterruptedException ex) {
        Logger.getLogger(Mp2.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Code snippet to make app sleep for 10seconds

Snapshot 8.7:

```
public static StringBuilder createCSVfile(PersistentObject pob) {
    DateFormat df = new SimpleDateFormat("HHmm");
    StringBuilder sb = new StringBuilder();
    int day = 1;
    DaylightRecord d;
    for (Object r : pob.getDayLightRecord()) {

        d = (DaylightRecord) r;
        // sb.append(pob.getDateObj() + ",");
        if (d != null) {
            if (Integer.toString(day).equals(d.getDay().toString())) {
                if (day != 1) {
                    sb.append("\n");
                }
                sb.append(d.getDay().toString() + ",");
                sb.append(df.format(d.getSunRise()) + ",");
                sb.append(df.format(d.getSunSet()) + ",");
                day++;
            } else {
                sb.append(df.format(d.getSunRise()) + ",");
                sb.append(df.format(d.getSunSet()) + ",");
            }
```

code snippet to create '.csv' file after deserialization of the PersistentObject

Snapshot 8.8:

```
public static void writeCSVfile(StringBuilder sb) {
    try {
        FileWriter fw;
        fw = new FileWriter("data/deserializedDaylightRecords.csv");
        fw.write(sb.toString(), 0, sb.length());
        fw.flush();
    } catch (IOException ex) {
        Logger.getLogger(Utilities.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Write deserialized data into deserializedDaylightRecords.csv file after creating it from createcsvfile() method

Snapshot 8.9:

```
//to compute data analytics for winter and summer
public static List findCorrespondingData(List daylightRecords, String startDate, String endDate, boolean checkForSolstice
    List computationList = new ArrayList<DataAnalyticsResult>();
    int value;
    for (int index = 0; index < daylightRecords.size();index++) {

        if (daylightRecords.get(index) == null) {
            index++;
            continue;
        } else {
            value = Utilities.computeSolsticeDataAnalytics(daylightRecords.get(index), startDate, endDate, checkForSolsti
            if (value != 0) {
                computationList.add(new DataAnalyticsResult(index, value));
            }
        }
    }
    if (checkForSolstice == true) {
        Collections.sort(computationList, new WinterSolstice());
    } else {
        Collections.sort(computationList, new SummerSolstice());
    }
    return computationList;
}
```

Compute and collects the data of day length into the arraylist for winter and summer solstice

Snapshot 8.10:

```
//to compute data analytics for spring and fall
public static List findRelevantData(List daylightRecords, String startDate, String endDate) {
    List computationListEqualDayNight = new ArrayList<DataAnalyticsResult>();
    int nextDaySunriseDataIndex;
    int valueResult;
    for (int index = 0; index < daylightRecords.size();index++) {

        if (daylightRecords.get(index) == null) {
            index++;
            continue;
        } else {
            nextDaySunriseDataIndex = index + 12;
            if (nextDaySunriseDataIndex >= daylightRecords.size()) {
                nextDaySunriseDataIndex = 0;
            } else if (daylightRecords.get(nextDaySunriseDataIndex) == null) {
                nextDaySunriseDataIndex++;
            } else {
                valueResult = Utilities.computeEquinoxDataAnalytics(daylightRecords.get(index), daylightRecords.get(nextDaySunriseDataIndex), sta
                if (valueResult != -1) {
                    computationListEqualDayNight.add(new DataAnalyticsResult(index, valueResult));
                }
            }

        }
    }
    Collections.sort(computationListEqualDayNight, new VernalandAutumnalEquinox());
    return computationListEqualDayNight;
}
```

computes and collects data of difference between day and night length to an arraylist for data analytics of equal day night in Equinox

Snapshot 8.11:

```
//compute data analytics for the specified date range and aggregate the data(for summer and winter)
public static int computeSolsticeDataAnalytics(Object o, String startDayofSolstice, String endDayofSolstice, boolean checkForSolstice) {
    int finalValue = 0;
    DaylightRecord dr1 = (DaylightRecord) o;

    String winterSolsticeStart = startDayofSolstice;
    String winterSolsticeEnd = endDayofSolstice;
    DateFormat df1 = new SimpleDateFormat("MM/dd/yyyy");
    try {
        if (checkForSolstice == true) {
            if (dr1.getSunRise().compareTo(df1.parse(winterSolsticeStart)) >= 0 || dr1.getSunSet().compareTo(df1.parse(winterSolsticeEnd)) <= 0) {
                finalValue = Integer.parseInt(findDayLength(dr1));
            }
        } else if (checkForSolstice == false) {
            if ((dr1.getSunRise().compareTo(df1.parse(winterSolsticeStart)) >= 0 && dr1.getSunSet().compareTo(df1.parse(winterSolsticeEnd)) <= 0)) {
                finalValue = Integer.parseInt(findDayLength(dr1));
            }
        }
    } catch (ParseException ex) {
        Logger.getLogger(Utilities.class.getName()).log(Level.SEVERE, null, ex);
```

compute and retrieve the daylength for all the dates for winter and summer solstice

Snapshot 8.12:

```
//compute day length
public static String findDayLength(DaylightRecord dr1) {
    long d;

    d = Math.abs(dr1.getSunRise().getTime() - dr1.getSunSet().getTime());
    int Hours = (int) ((d / (1000 * 60 * 60)) % 24);
    int Minutes = (int) (int) ((d / (1000 * 60)) % 60);
    String s;
    if (Integer.toString(Minutes).length() == 1) {
        s = "0" + Integer.toString(Minutes);
        s = Integer.toString(Hours) + s;
    } else {
        s = Integer.toString(Hours) + Integer.toString(Minutes);
    }
    return s;
}
```

to find the daylength for a particular date.

Snapshot 8.13:

```
//compute data analytics for the specified date range and aggregate the data(for spring and fall)
public static int computeEquinoxDataAnalytics(Object o1, Object o2, String equinoxStartDate, String equinoxEndDate) {

    int finalValue = -1;
    DaylightRecord dr1 = (DaylightRecord) o1;
    DaylightRecord dr2 = (DaylightRecord) o2;
    long d1;
    String winterSolsticeStart = equinoxStartDate;
    String winterSolsticeEnd = equinoxEndDate;
    DateFormat df1 = new SimpleDateFormat("MM/dd/yyyy");
    try {
        if (dr1.getSunRise().compareTo(df1.parse(winterSolsticeStart)) >= 0 && dr1.getSunSet().compareTo(df1.parse(winterSolsticeEnd)) <= 0 && dr2.getSu
            d1 = Math.abs(dr1.getSunRise().getTime() - dr1.getSunSet().getTime());
            int dayHours = (int) ((d1 / (1000 * 60 * 60)) % 24);
            int dayMinutes = (int) ((d1 / (1000 * 60)) % 60);

            int timeHoursDifference = 0;
            int timeMinutesDifference = 0;
            String s;
            if (dayHours >= 12 && dayMinutes >= 0) {
                if (dayHours == 12 && dayMinutes == 0) {
```

Compute and retrieve the difference between day length and night length which can be used to decide the equal day and night in equinox.

Snapshot 8.14:

```
                    s = "0" + Integer.toString(timeMinutesDifference);
                    s = Integer.toString(timeHoursDifference) + s;
                } else {
                    s = Integer.toString(timeHoursDifference) + Integer.toString(timeMinutesDifference);
                }
                finalValue = Integer.parseInt(s);
            }
        } else {
            timeHoursDifference = 12 - dayHours;
            timeMinutesDifference = 00 - dayMinutes;
            if (timeMinutesDifference < 0) {
                timeHoursDifference = timeHoursDifference - 1;
                timeMinutesDifference = timeMinutesDifference + 60;
            }
            if (Integer.toString(timeMinutesDifference).length() == 1) {
                s = "0" + Integer.toString(timeMinutesDifference);
                s = Integer.toString(timeHoursDifference) + s;
            } else {
                s = Integer.toString(timeHoursDifference) + Integer.toString(timeMinutesDifference);
            }
            finalValue = Integer.parseInt(s);
        }
```

continued code snippet of computeEquinoxDataAnalytics().

Snapshot 8.15:

```
//retrieve required data for display
public static ArrayList retrieveDataFromComputation(List newList, List dataList) {
    DateFormat df = new SimpleDateFormat("dd MMMM yyyy");
    ArrayList finalValues = new ArrayList();
    DataAnalyticsResult dar;
    DaylightRecord dlr;
    int firstTime = 0;
    int data = 0;
    int repeat = 1;
    for (Object o : newList) {
        dar = (DataAnalyticsResult) o;
        dlr = (DaylightRecord) dataList.get(dar.getArrayIndexData());

        if (firstTime == 0) {
            data = dar.getComputationData();
            firstTime = 1;
            finalValues.add(df.format(dlr.getSunRise()).toString());
        } else {
            if (data == dar.getComputationData()) {
                repeat++;
                finalValues.add(df.format(dlr.getSunRise()).toString());
            }
        }
    }
    finalValues.add(repeat);
    return finalValues;
}
```

Retrieve required data to display from all the computation of data analytics results

Snapshot 8.16:

```
//display computed data
public static void printData(ArrayList computationData) {
    int repeatData = computationData.size();
    for (int m = 0; m < repeatData - 1; m++) {
        System.out.print(computationData.get(m) + ",");
    }
}
//write daylightrecords to file
public static void writeListtoFile(List daylightRecords) {
    try {
        BufferedWriter out = null;
        File file = new File("output/daylight-records.txt");
        out = new BufferedWriter(
                new FileWriter(file));
        for (Object s : daylightRecords) {
            // Write line to file.
            if (s != null) {
                DaylightRecord d = (DaylightRecord) s;
                out.write(d.toString());
            } else {
                out.write("null");
            }
            // Write newLine with BufferedReader method.
            out.newLine();
        }
        out.close();
    } catch (IOException ex) {
        Logger.getLogger(Mp2.class.getName()).log(Level.SEVERE, null, ex);
    }
```

Code snippet to print the final data for all data analytics and to write the daylightRecords list data into daylight-records.txt

9. JDBCUtilities class:

Snapshot 9.1:

```
                    rec = new DaylightRecord(recordValues.get(0).toString(), recordValues.get(1).toString(), df.parse(recordValues.
                }
            } catch (ParseException ex) {
                Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
            }
            //Execute theSQL insert statement to insert the data into the database.
            //On success set result message.
            stmt.execute("INSERT INTO DAYLIGHTRECORD ("
                    + "day, month, sunRise, sunSet)"
                    + " VALUES("
                    + Integer.parseInt(rec.getDay()) + ",'"
                    + rec.getMonth() + "','"
                    + df.format(rec.getSunRise()) + "','"
                    + df.format(rec.getSunSet()) + "');");
            result = "Record created";
        } else {
            //If record exists display respective data.
            result = "Record already exists.";
        }
    } catch (NumberFormatException nfe) {
        result = "Error in entered data type. Please follow the datatype while entering the inputs";
    } //catch exception if occurs while inserting the data.
    catch (SQLException ex) {
        result = "Error while inserting data into database.";
    } finally {
        try {
```

code snippet to create record in database using the input given by the user continued which shows executing the query to insert record into database and stores respective success or failure message in a string.

Snapshot 9.2:

```
//If the inputs are correct the record is inserted with the values.

public static String createRecord(List recordValues) {
    DateFormat df = new SimpleDateFormat("HHmm");
    String result = "";
    //connect to database.
    Connection conn = getConnection();
    //checks if connection is established.
    if (conn != null) {
        try {
            //creates a statement object by sending SQL statements to the database.
            Statement stmt = conn.createStatement();
            //Executes the SQL statement. This is to check whether the daylight record for the day and month value
            //already exists. This check is done to avoid duplicate entry.
            ResultSet rs = stmt.executeQuery("SELECT * FROM DAYLIGHTRECORD WHERE day=" + Integer.parseInt(recordValues.get(0).toString()) + " and month
            //if the record not exists then insert the record.
            if (!rs.next()) {
                DaylightRecord rec = null;
                try {
                    //create the respective daylight record checking for sunrise and sunset time.
                    if (recordValues.get(2).toString().equals("") && recordValues.get(3).toString().equals("")) {
                        rec = new DaylightRecord(recordValues.get(0).toString(), recordValues.get(1).toString(), df.parse("0000"), df.parse("0000"));
                    } else if (!recordValues.get(2).toString().equals("") && recordValues.get(3).toString().equals("")) {
                        rec = new DaylightRecord(recordValues.get(0).toString(), recordValues.get(1).toString(), df.parse(recordValues.get(2).toString
                    } else if (recordValues.get(2).toString().equals("") && !recordValues.get(3).toString().equals("")) {
                        rec = new DaylightRecord(recordValues.get(0).toString(), recordValues.get(1).toString(), df.parse("0000"), df.parse(recordValue
```

code snippet to create record in database using the input given by the user.

Snapshot 9.3:

```
//This function is called to connect to the database itm411db.
private static Connection getConnection() {
    Connection conn = null;
    try {
        //Establishes a connection to the given database URL.
        conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/itm411DB",
    }//catches exception if error in connection to database occurs.
    catch (SQLException ex) {
    }
    return conn;
}
```

This function is used to establish connection to the database to perform all CRUD operations.

Snapshot 9.4:

```
//This function is called whenever the user wants to retrieve an daylight record for a day or month or for both.
//day value and month value taken as input from the user is passed to this function. If the entered day and month value
//is valid appropriate record is retrieved and displayed.
public static List retrieveRecordForDay(String dayValue, String monthValue) {
    DateFormat df = new SimpleDateFormat("HHmm");
    //connect to database.
    Connection conn = null;
    //to store the list of daylight records.
    List<DaylightRecord> recList = null;
    try {
        //connect to database.
        conn = getConnection();
        if (conn != null) {
            recList = new ArrayList();
            //creates a statement object by sending SQL statements to the database.
            Statement stmt = conn.createStatement();
            ResultSet rs = null;
            if (monthValue.equals("None")) {
                //Executes the SQL statement to retrieve the respective daylight record.
                rs = stmt.executeQuery("SELECT * FROM DAYLIGHTRECORD WHERE day = " + Integer.parseInt(dayValue));
            } else {
                //Executes the SQL statement to retrieve the respective daylight record.
                rs = stmt.executeQuery("SELECT * FROM DAYLIGHTRECORD WHERE day =" + Integer.parseInt(dayValue) + " and month='" + monthValue + "';");
            }
```

code snippet to retrive record from database for the user input day and month value. day Value is must where as month value is optional according to user requirement.

Snapshot 9.5:

```
            while (rs.next()) {
                //get the value from the result set obtained from database.
                String day = Integer.toString(rs.getInt("day"));
                String month = rs.getString("month");
                String sunRise = rs.getString("sunRise");
                String sunSet = rs.getString("sunSet");
                try {
                    //create record and add into the list.
                    DaylightRecord rec = new DaylightRecord(day, month, df.parse(sunRise), df.parse(sunSet));
                    recList.add(rec);
                } catch (ParseException ex) {
                    Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        }
    } //catch exception if occurs while retrieving the data.
    catch (SQLException ex) {
        //Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        try {
            //Close the connection with database at last when the required operation is finished.
            if (conn != null) {
                conn.close();
            }
        }//catches exception if error occurs while closing the database connection.
        catch (SQLException ex) {
            // Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

code snippet to retrieve a record for a day value continued.

Snapshot 9.6:

```
        //retrieved record value is returned
        return recList;
    }

    //This function is called whenever the user wants to retrieve daylight record between two day or month or for both range values.
    //day value and month value taken as input from the user is passed to this function. If the entered day and month value
    //is valid appropriate record is retrieved and displayed.
    public static List retrieveRecordBetweenDayRange(String beginDayValue, String endDayValue, String monthValue) {
        DateFormat df = new SimpleDateFormat("HHmm");
        //connect to database.
        Connection conn = null;
        //to store the list of daylight records.
        List<DaylightRecord> recList = null;
        try {
            //connect to database.
            conn = getConnection();
            if (conn != null) {
                recList = new ArrayList();
                //creates a statement object by sending SQL statements to the database.
                Statement stmt = conn.createStatement();
                ResultSet rs = null;
                if (monthValue.equals("None")) {
                    //Executes the SQL statement to retrieve the respective daylight record.
                    rs = stmt.executeQuery("SELECT * FROM DAYLIGHTRECORD WHERE day >= " + Integer.parseInt(beginDayValue) + " and day <= " + Integer.parse
                } else {
                    //Executes the SQL statement to retrieve the respective daylight record.
                    rs = stmt.executeQuery("SELECT * FROM DAYLIGHTRECORD WHERE day >= " + Integer.parseInt(beginDayValue) + " and day <= " + Integer.parse
                }
```

code snippet to retrieve records between two day ranges specified by the user as per the requirement. Month value is optional if user needs records for particular month between two day ranges.

Snapshot 9.7:

```
                // Retrieved record values are wrapped into an record object
                while (rs.next()) {
                    //get the value from the result set obtained from database.
                    String day = Integer.toString(rs.getInt("day"));
                    String month = rs.getString("month");
                    String sunRise = rs.getString("sunRise");
                    String sunSet = rs.getString("sunSet");
                    try {
                        //create record and add into the list.
                        DaylightRecord rec = new DaylightRecord(day, month, df.parse(sunRise), df.parse(sunSet));
                        recList.add(rec);
                    } catch (ParseException ex) {
                        Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
            }

        } //catch exception if occurs while retrieving the data.
        catch (SQLException ex) {
            //Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            try {
                //Close the connection with database at last when the required operation is finished.
                if (conn != null) {
                    conn.close();
                }
            }//catches exception if error occurs while closing the database connection.
            catch (SQLException ex) {
```

To retrieve records between two day ranges continued...

Snapshot 9.8:

```
        return recList;
    }

    //This function is called whenever the user wants to retrieve all the daylight records from the database.
    public static List retrieveAllRecords() {
        DateFormat df = new SimpleDateFormat("HHmm");
        //connect to database.
        Connection conn = getConnection();
        List<DaylightRecord> recList = null;
        if (conn != null) {
            try {
                recList = new ArrayList();
                //creates a statement object by sending SQL statements to the database.
                Statement stmt = conn.createStatement();
                //Executes the SQL statement to retrieve the all daylight records present in the database.
                ResultSet rs = stmt.executeQuery("SELECT * FROM DAYLIGHTRECORD;");
                // Retrieved record values are wrapped into an record objects and stored in the arraylist.
                while (rs.next()) {
                    //get the value from the result set obtained from database.
                    String day = Integer.toString(rs.getInt("day"));
                    String month = rs.getString("month");
                    String sunRise = rs.getString("sunRise");
                    String sunSet = rs.getString("sunSet");
                    DaylightRecord rec;
                    try {
                        //create record and add into the list.
                        rec = new DaylightRecord(day, month, df.parse(sunRise), df.parse(sunSet));
```

code snippet to retireve all records present from the database. Appropriate message displayed if any error occurs.

Snapshot 9.9:

```
            }//catch exception if occurs while retrieving the data.
            catch (SQLException ex) {
                Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
            } finally {
                try {
                    //Close the connection with database at last when the required operation is finished.
                    if (conn != null) {
                        conn.close();
                    }
                } catch (SQLException ex) {
                    Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        }
        //retrieved record values arraylist is returned.
        return recList;
    }
```

code to retrieve all records from database continued...

Snapshot 9.10:

```
//This function is called whenever the user wants to update an daylight record
//All the input of the user is passed to this function to update the record.
//If the inputs are correct the database is updated with the values.
public static String updateRecord(List updateRecordValues) {
    DateFormat df = new SimpleDateFormat("HHmm");
    Connection conn = null;
    String result = "";
    try {
        //connect to database.
        conn = getConnection();
        if (conn != null) {
            //creates a statement object by sending SQL statements to the database.
            Statement stmt = conn.createStatement();
            //Executes the SQL statement. This is to check whether the daylight record for the day and month value
            //already exists.
            ResultSet rs = stmt.executeQuery("SELECT * FROM DAYLIGHTRECORD WHERE day=" + Integer.parseInt(updateRecordValues.get(0).toString()) + " and
            if (rs.next()) {
                //updates daylight record using the values entered by the user.
                DaylightRecord rec = null;
                try {
                    //create the respective daylight record checking for sunrise and sunset time
                    if (updateRecordValues.get(2).toString().equals("") && updateRecordValues.get(3).toString().equals("")) {
                        rec = new DaylightRecord(updateRecordValues.get(0).toString(), updateRecordValues.get(1).toString(), df.parse("0000"), df.parse(
                    } else if (!updateRecordValues.get(2).toString().equals("") && updateRecordValues.get(3).toString().equals("")) {
                        rec = new DaylightRecord(updateRecordValues.get(0).toString(), updateRecordValues.get(1).toString(), df.parse(updateRecordValues
                    } else if (updateRecordValues.get(2).toString().equals("") && !updateRecordValues.get(3).toString().equals("")) {
```

code snippet to update a particular record in the database. The record is updated using input values given by the user. Errors are handled and appropriate messages are displayed.

Snapshot 9.11:

```
                        rec = new DaylightRecord(updateRecordValues.get(0).toString(), updateRecordValues.get(1).toString(), df.parse("0000"), df.parse(
                    } else {
                        rec = new DaylightRecord(updateRecordValues.get(0).toString(), updateRecordValues.get(1).toString(), df.parse(updateRecordValues
                    }
                } catch (ParseException ex) {
                    Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
                }
                // update the respective employee record with the changed data.
                stmt.execute("UPDATE DAYLIGHTRECORD SET "
                        + "day=" + Integer.parseInt(rec.getDay()) + ", "
                        + "month='" + rec.getMonth() + "', "
                        + "sunRise='" + df.format(rec.getSunRise()) + "', "
                        + "sunSet='" + df.format(rec.getSunSet())
                        + "' WHERE day=" + Integer.parseInt(rec.getDay()) + " and month='" + rec.getMonth() + "';");
                result = "Record Updated Succesfully";

            } else {
                //If record exists display respective message.
                result = "Record for the day " + updateRecordValues.get(0) + " and month " + updateRecordValues.get(1) + " does not exists. Record not u
            }
        } else {
            result = "Error while connecting to database. Please check the MySQL server connection.";
        }
    } //catch exception if occurs while updating the data.
    catch (SQLException ex) {
        result = "Error while inserting data into database.";
    } finally {
        try {
```

update of particualr record code continued...

Snapshot 9.12:

```
            catch (SQLException ex) {
                Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
        //return message.
        return result;
    }

    //This function is called whenever the user wants to delete daylight record(s)
    //day value and month value of the record is asked as input by the user.
    //If the entered day and month value is valid appropriate record is deleted.
    public static String deleteRecord(String day, String month) {
        //connect to database.
        Connection conn = null;
        String result = "";
        boolean isValidday = false;
        try {
            //check for valid day value.
            if (!day.equals("")) {
                Integer.parseInt(day);
                isValidday = true;
            } else {
                isValidday = true;
            }
        } catch (NumberFormatException nfe) {
            result = "Error in entered day data type.";
```

code to delete a particular record from database. The day and month value is required to delete a single record. Only day value is enough to delete a particular day from all months. only month value is required to delete all daya present in the month. Errors are handeled and appropriate messages are displayed.

Snapshot 9.13:

```
if (isValidday) {
    conn = getConnection();
    if (conn != null) {
        try {
            isValidday = false;
            //creates a statement object by sending SQL statements to the database.
            Statement stmt = conn.createStatement();
            if (!day.equals("") && !month.equals("None")) {
                //Executes the SQL statement. This is to check whether the daylight record
                //exists. This check is done to avoid deleting the data which is not present.
                ResultSet rs = stmt.executeQuery("SELECT * FROM DAYLIGHTRECORD WHERE day = " + Integer.parseInt(day) + " and month='" + month + "
                //After checking data whether present or not do the appropriate task.
                if (rs.next()) {
                    //Executes the SQL statement to delete the respective daylight record.
                    stmt.execute("DELETE FROM DAYLIGHTRECORD WHERE day = " + Integer.parseInt(day) + " and month='" + month + "';");
                    isValidday = true;
                } else {
                    result = "Record for day " + day + " and month " + month + " not exists.";
                }
            } else if (day.equals("") && !month.equals("None")) {

                ResultSet rs = stmt.executeQuery("SELECT * FROM DAYLIGHTRECORD WHERE month='" + month + "';");
                //After checking data whether present or not do the appropriate task.
                if (rs.next()) {
                    //Executes the SQL statement to delete the respective employee record using employee number ID.
                    stmt.execute("DELETE FROM DAYLIGHTRECORD WHERE month='" + month + "';");
                    isValidday = true;
                } else {
```

code snippet to delete a record is continued...

Snapshot 9.14:

```
                    //Executes the SQL statement to delete the respective employee record using employee number ID.
                    stmt.execute("DELETE FROM DAYLIGHTRECORD WHERE month='" + month + "';");
                    isValidday = true;
                } else {
                    result = "Record for month " + month + " not exists.";
                }
            } else if (!day.equals("") && month.equals("None")) {
                ResultSet rs = stmt.executeQuery("SELECT * FROM DAYLIGHTRECORD WHERE day = " + Integer.parseInt(day) + ";");
                //After checking data whether present or not do the appropriate task.
                if (rs.next()) {
                    //Executes the SQL statement to delete the respective employee record using employee number ID.
                    stmt.execute("DELETE FROM DAYLIGHTRECORD WHERE day = " + Integer.parseInt(day) + ";");
                    isValidday = true;
                } else {
                    result = "Record for day " + day + " not exists.";
                }
            }
        } catch (SQLException ex) {
            Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
        }
        //if the delete is done enter into the loop.
        if (isValidday) {
            ResultSet rs = null;
            try {
                //creates a statement object by sending SQL statements to the database.
                Statement stmt = conn.createStatement();
                if (!day.equals("") && !month.equals("None")) {
```

code snippet to delete a record continued in which query is executed to delete the respective record and errors are handeled is shown.

Snapshot 9.15:

```
                    rs = stmt.executeQuery("SELECT * FROM DAYLIGHTRECORD WHERE day = " + Integer.parseInt(day) + " and month='" + month + "';");
                } else if (day.equals("") && !month.equals("None")) {
                    //Executes the SQL statement. This is to check whether the data is deleted successfully or not.
                    rs = stmt.executeQuery("SELECT * FROM DAYLIGHTRECORD WHERE month='" + month + "';");
                } else if (!day.equals("") && month.equals("None")) {
                    //Executes the SQL statement. This is to check whether the data is deleted successfully or not.
                    rs = stmt.executeQuery("SELECT * FROM DAYLIGHTRECORD WHERE day=" + Integer.parseInt(day) + ";");
                }
            }//catches exception if error occurs while executing the query.
            catch (SQLException ex) {
                Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
            }
            try {
                //if data is deleted from database then set successful result message else failure message.
                if (!rs.next()) {
                    result = "Record Deletion Successful.";
                } else {
                    result = "Record not deleted.";
                }
            }//catches exception while reading data from the result returned from the database query .
            catch (SQLException ex) {
                Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
            } finally {
                try {
                    //Close the connection with database at last when the required operation is finished.
                    if (conn != null) {
                        conn.close();
```

code snippet to delete a record continued in which checking whether record is deleted or not is performed after deletion is shown.

Snapshot 9.16:

```
                try {
                    //Close the connection with database at last when the required operation is fi
                    if (conn != null) {
                        conn.close();
                    }
                }//catches exception if error occurs while closing the database connection.
                catch (SQLException ex) {
                    Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        } else {
            try {
                //Close the connection with database at last when the required operation is finish
                if (conn != null) {
                    conn.close();
                }
            }//catches exception if error occurs while closing the database connection.
            catch (SQLException ex) {
                Logger.getLogger(JDBCUtilities.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    } else {
        result = "Error while connecting to database. Please check the MySQL server connection.";
    }
}// returns message result.
return result;
}
```

deletion of a particular code snippet continued.

Snapshot 9.17:

```
//this function is called at the start up to create the database table
//and insert csv file data into database at startup.
public static boolean initByCreatingandPopulatingTableInDatabase(List<DaylightRecord> dayLightRecords) {
    //connect to database.
    Connection conn = null;
    DateFormat df = new SimpleDateFormat("HHmm");
    boolean result = false;
    try {
        conn = getConnection();
        //if connection successful then only continue
        if (conn != null) {
            //creates a statement object by sending SQL statements to the database.
            Statement stmt = conn.createStatement();
            try {
                //drop the table if any record exists.
                stmt.execute("DROP TABLE DAYLIGHTRECORD;");
            } catch (SQLException ex) {
                System.out.println("No such table exists... ");
            }
            //creates table in the database with columns mentioned.
            stmt.execute("CREATE TABLE DAYLIGHTRECORD (day integer,month VARCHAR(10),sunRise VARCHAR(4),sunSet VARCHAR(4), PRIMARY KEY(day,month));");
            for (DaylightRecord dlr : dayLightRecords) {
                if (dlr != null) {
                    //inserts daylight records into database.
                    stmt.execute("INSERT INTO DAYLIGHTRECORD ("
```

code snippet to create table in the database and insert all the daylight records by retrieving data from csv file and store in the database.

Snapshot 9.18:

```
                    stmt.execute("INSERT INTO DAYLIGHTRECORD ("
                            + "day, month, sunRise, sunSet)"
                            + " VALUES('"
                            + Integer.parseInt(dlr.getDay()) + "','"
                            + dlr.getMonth() + "','"
                            + df.format(dlr.getSunRise()) + "','"
                            + df.format(dlr.getSunSet()) + "');");
                }
            }
            result = true;
        }
    } catch (SQLException ex) {
        System.out.println("SQL statement execution fail !");
        System.out.println(ex.getMessage());
    } finally {
        try {
            //Close the connection with database at last when the required op
            if (conn != null) {
                conn.close();
            }
        }//catches exception if error occurs while closing the database connection.
        catch (SQLException ex) {
            Logger.getLogger(JDBCUtilities.class
                    .getName()).log(Level.SEVERE, null, ex);
        }
    }
    //returns boolean result.
    return result;
}
```

code snippet to create table and insert record into it is continued.

10. GUIUtilities class:

Snapshot 10.1:

```
//This class is used to minimize the main class,
//which makes main class look neat and easy to understand
public class GUIUtilities {

    //This function is called when user want to create record into database
    //ie, when create button is pressed from GUI.This function creates the Jframe
    //whic have fields to accept input required to create record.
    public static void createRecordJFrame(final MainFrame mf, final JLabel OperationStatusDisplayValue) {
        final List data = new ArrayList();
        //creates jframe window.
        final JFrame cjf = new JFrame("Enter the data to create record.");
        cjf.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        //listener when button pressed on window.
        cjf.addWindowListener(new WindowAdapter() {
            public void windowClosed(WindowEvent e) {
                mf.setEnabled(true);
            }
        });
        //creates panel.
        JPanel cjp = new JPanel();
        //sets border for panel.
        cjp.setBorder(BorderFactory.createEmptyBorder(0, 10, 10, 10));
```

code snippet to create Jframe with components to accept values from user when 'Create' button is pressed by the user.

Snapshot 10.2:

```
cjp.setBorder(BorderFactory.createEmptyBorder(0, 10, 10, 10));
// this sets the location on the screen for this window to display.
cjf.setLocation(mf.getSize().width / 2 - cjf.getSize().width / 2, mf.getSize().height / 2 - cjf.getSize().height / 2);
//creates label.
JLabel cj1_day = new JLabel("Enter day(Integer data type):");
//Textfield to accept input for day.
final JTextField cj1_dayTextField = new JTextField();
JLabel cj1_month = new JLabel("Select Month: ");
//combobox create to display the list of months out of
//which user has to select one to insert record for respective month.
String[] comboBoxContents = {"January", "February", "March", "April", "May", "June", "July", "August", "September", "October",
// creates combo box with comboBoxContents.
final JComboBox cj1_monthComboBox = new JComboBox(comboBoxContents);
JLabel cj1_sunRise = new JLabel("Enter sun rise time(HHmm format): ");
//Textfield to accept the sunRise value.
final JTextField cj1_sunRiseTextField = new JTextField(4);
JLabel cj1_sunSet = new JLabel("Enter sun set time(HHmm format): ");
//Textfield to accept the sunset value
final JTextField cj1_sunSetTextField = new JTextField(4);
// create buttons for accpting the values or cancelling the task.
JButton cok_Button = new JButton("Ok");
JButton ccancel_Button = new JButton("Cancel");
//set layout size.
cjp.setLayout(new GridLayout(5, 2));
//add all the created components into jframe.
```

code snippet to create Jfame with components to accept values from user continued

Snapshot 10.3:

```
    cjp.add(cj1_dayTextField);
    cjp.add(cj1_month);
    cjp.add(cj1_monthComboBox);
    cjp.add(cj1_sunRise);
    cjp.add(cj1_sunRiseTextField);
    cjp.add(cj1_sunSet);
    cjp.add(cj1_sunSetTextField);
    cjp.add(cok_Button);
    cjp.add(ccancel_Button);
    cjf.setVisible(true);
    cjf.setSize(300, 200);
    cjf.pack();
    //when this jframe is popped up disable the main window to avoid performing other operations
    mf.setEnabled(false);
    // when ok button is pressen from jframe window popped up.
    cok_Button.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {

            data.add(cj1_dayTextField.getText());
            data.add((String) cj1_monthComboBox.getSelectedItem());
            data.add(cj1_sunRiseTextField.getText());
            data.add(cj1_sunSetTextField.getText());
            cjf.setVisible(false);
            if (!data.get(0).equals("")) {
```

code to create Jframe to accept values from user is continued. when 'ok' button is pressed in the window then the validation of entered values take place and create record method is invoked to complete the operation.

Snapshot 10.4:

```
        public void actionPerformed(ActionEvent e) {

            data.add(cj1_dayTextField.getText());
            data.add((String) cj1_monthComboBox.getSelectedItem());
            data.add(cj1_sunRiseTextField.getText());
            data.add(cj1_sunSetTextField.getText());
            cjf.setVisible(false);
            if (!data.get(0).equals("")) {

                if ((data.get(0).equals("29") || data.get(0).equals("30") || data.get(0).equals("31")) && (data.get(1).equals("Febru
                    OperationStatusDisplayValue.setText("Day " + data.get(0) + " for " + data.get(1) + " doesnt exist for year 2013.
                } else {
                    boolean result = checkIsInteger(data.get(0).toString(), OperationStatusDisplayValue, "Error in entered day data
                    if (result) {
                        if ((data.get(2).toString().equals("")) && (data.get(3).toString().equals(""))
                                || (!data.get(2).toString().equals("")) && (data.get(3).toString().equals(""))
                                || (data.get(2).toString().equals("")) && (!data.get(3).toString().equals(""))) {
                            String message = JDBCUtilities.createRecord(data);
                            OperationStatusDisplayValue.setText(message);
                        } else if (!(data.get(2).toString().length() < 3) || !(data.get(2).toString().length() > 4)) {
                            result = false;
                            result = checkIsInteger(data.get(2).toString(), OperationStatusDisplayValue, "Error in entered sunRise v
                            if (result) {
                                result = false;
                                if (!(data.get(3).toString().length() < 3) || !(data.get(3).toString().length() > 4)) {
```

code to create Jframe to insert record continued.

Snapshot 10.5:

```
                                if (!(data.get(3).toString().length() < 3) || !(data.get(3).toString().length() > 4)) {
                                    result = checkIsInteger(data.get(3).toString(), OperationStatusDisplayValue, "Error in entered
                                    if (result) {
                                        String message = JDBCUtilities.createRecord(data);
                                        OperationStatusDisplayValue.setText(message);
                                    }
                                } else {
                                    OperationStatusDisplayValue.setText("Error in entered sunSet format type. Please follow the dat
                                }
                            }
                        } else {
                            OperationStatusDisplayValue.setText("Error in entered sunRise format type. Please follow the datatype w
                        }
                    }
                }
            } else {
                OperationStatusDisplayValue.setText("Empty day value. Record not created. ");
            }
            mf.setEnabled(true);
        }
    });
    // when cancel button is pressen from jframe window popped up.
    ccancel_Button.addActionListener(new ActionListener() {
        @Override
```

code to create Jframe to insert record continued.

Snapshot 10.6:

```
    });
    // when cancel button is pressen from jframe window popped up.
    ccancel_Button.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            cjf.setVisible(false);
            OperationStatusDisplayValue.setText("Create task cancelled.")
            mf.setEnabled(true);
        }
    });
}
```

code to create Jframe to insert record continued. when cancel button is pressed is popped window then no operation is performed.

Snapshot 10.7:

```
//This function is called when user want to update record into database
//ie, when update button is pressed from GUI.This function creates the Jframe
//which have fields to accept input required to update record.
public static void updateRecordJFrame(final MainFrame mf, final JLabel OperationStatusDisplayValue) {
    final List data = new ArrayList();
    //creates jframe window.
    final JFrame cjf = new JFrame("Enter the data to update record.");
    cjf.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    //listener when button pressed on window.
    cjf.addWindowListener(new WindowAdapter() {
        public void windowClosed(WindowEvent e) {
            mf.setEnabled(true);
        }
    });
    //sets border for panel.
    JPanel cjp = new JPanel();
    //sets border for panel.
    cjp.setBorder(BorderFactory.createEmptyBorder(0, 10, 10, 10));
    // this sets the location on the screen for this window to display.
    cjf.setLocation(mf.getSize().width / 2 - cjf.getSize().width / 2, mf.getSize().height / 2 - cjf.getSize().height / 2);
    //creates label.
    JLabel cjl_day = new JLabel("Enter day(Integer data type):");
    //Textfield to accept input for day.
    final JTextField cj1_dayTextField = new JTextField();
    JLabel cjl_month = new JLabel("Select Month: ");
```

code snippet to create Jframe with components to accept values from user when Update button is pressed by the user.

Snapshot 10.8:

```
//combobox create to display the list of months out of
//which user has to select one to insert record for respective month.
String[] comboBoxContents = {"January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "Nov
// creates combo box with comboBoxContents.
final JComboBox cj1_monthComboBox = new JComboBox(comboBoxContents);
JLabel cjl_sunRise = new JLabel("Enter sun rise time(HHmm format):");
//Textfield to accept the sunRise value.
final JTextField cj1_sunRiseTextField = new JTextField(4);
JLabel cjl_sunSet = new JLabel("Enter sun set time(HHmm format): ");
//Textfield to accept the sunset value
final JTextField cj1_sunSetTextField = new JTextField(4);
// create buttons for accpting the values or cancelling the task.
JButton cok_Button = new JButton("Ok");
JButton ccancel_Button = new JButton("Cancel");
//set layout size.
cjp.setLayout(new GridLayout(5, 2));
//add all the created components into jframe.
cjf.add(cjp);
cjp.add(cjl_day);
cjp.add(cj1_dayTextField);
cjp.add(cjl_month);
cjp.add(cj1_monthComboBox);
cjp.add(cjl_sunRise);
cjp.add(cj1_sunRiseTextField);
```

code snippet to create Jfame with components to accept values from user continued

Snapshot 10.9:

```
cjf.setSize(300, 200);
cjf.pack();
//when this jframe is popped up disable the main window to avoid performing other operations.
mf.setEnabled(false);
// when ok button is pressen from jframe window popped up.
cok_Button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        data.add(cj1_dayTextField.getText());
        data.add((String) cj1_monthComboBox.getSelectedItem());
        data.add(cj1_sunRiseTextField.getText());
        data.add(cj1_sunSetTextField.getText());
        cjf.setVisible(false);
        if (!data.get(0).equals("")) {
            boolean result = checkIsInteger(data.get(0).toString(), OperationStatusDisplayVali
            if (result) {
                if ((data.get(2).toString().equals("")) && (data.get(3).toString().equals(""))
                        || (!data.get(2).toString().equals("")) && (data.get(3).toString().equ
                        || (data.get(2).toString().equals("")) && (!data.get(3).toString().equ
                    String message = JDBCUtilities.updateRecord(data);
                    OperationStatusDisplayValue.setText(message);
                } else if (!(data.get(2).toString().length() < 3) || !(data.get(2).toString().
```

code to create Jframe to accept values from user is continued. when 'ok' button is pressed in the window then the validation of entered values take place and update record method is invoked to complete the operation.

Snapshot 10.10:

```
                } else if (!(data.get(2).toString().length() < 3) || !(data.get(2).toString().length() > 4)) {
                    result = false;
                    result = checkIsInteger(data.get(2).toString(), OperationStatusDisplayValue, "Error in entered sunRise value
                    if (result) {
                        result = false;
                        if (!(data.get(3).toString().length() < 3) || !(data.get(3).toString().length() > 4)) {
                            result = checkIsInteger(data.get(3).toString(), OperationStatusDisplayValue, "Error in entered sunSe
                            if (result) {
                                String message = JDBCUtilities.updateRecord(data);
                                OperationStatusDisplayValue.setText(message);
                            }
                        } else {
                            OperationStatusDisplayValue.setText("Error in entered sunSet format type. Please follow the datatype
                        }
                    }
                } else {
                    OperationStatusDisplayValue.setText("Error in entered sunRise format type. Please follow the datatype while
                }
            }
        } else {
            OperationStatusDisplayValue.setText("Empty day value. Record not updated. ");
        }
        mf.setEnabled(true);
    }
});
```

code to create Jframe to update record continued.

Snapshot 10.11:

```
            }
            mf.setEnabled(true);
        }
    });
    // when cancel button is pressen from jframe window popped up.
    ccancel_Button.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            cjf.setVisible(false);
            OperationStatusDisplayValue.setText("Update task cancelled.");
            mf.setEnabled(true);
        }
    });
}
```

code to create Jframe to update record continued. when cancel button is pressed is popped window then no operation is performed.

Snapshot 10.12:

```
//This function is used to check whether the string is integer or not.
public static boolean checkIsInteger(String value, JLabel OperationStatusDisplayValue, String message) {
    boolean result = false;
    try {
        //Parse the string to convert it to int.
        Integer.parseInt(value);
        result = true;
    } catch (NumberFormatException nfe) {
        OperationStatusDisplayValue.setText(message);
    }
    return result;
}
```

code snippet to check whther the string is inteder or not.

Snapshot 10.13:

```
//This function is called to serialize the daylightrecords and date of persistent object.
//This method is invoked when serialize button is pressed from GUI.
public static boolean serializeandDisplaytheStatusOnGUI(List<DaylightRecord> dayLightRecords, JLabel OperationStatusDisplayValue) {
    boolean result = false;
    //creates persistent object which will be serialized.
    PersistentObject pObj = new PersistentObject(new Date(), dayLightRecords);
    result = Utilities.serializeListObject(pObj);
    //If result is success or failure display respective message.
    if (result) {
        OperationStatusDisplayValue.setText("Serialization successful.");
    } else {
        OperationStatusDisplayValue.setText("Serialization failed.");

    }
    return result;
}
```

code snippet to serialize the persistent object and on success or failure displays the appropriate message. This function is invoked when'Serialize' button is pressed in GUI.

Snapshot 10.14:

```
//This function is called to write the deserialized persistent object into a file.
//This method is invoked when deserialize button is pressed from GUI.
public static boolean writeDeserializedDatatofile(PersistentObject deserializePObj, JLabel OperationStatusDisplayValue) {
    boolean result = false;
    //this function is invoked to write dederialized object data into csv file
    StringBuilder sb = Utilities.createCSVfile(deserializePObj);
    result = Utilities.writeCSVfile(sb);
    //if error occurs then respective error message is displayed.

    return result;
}
```

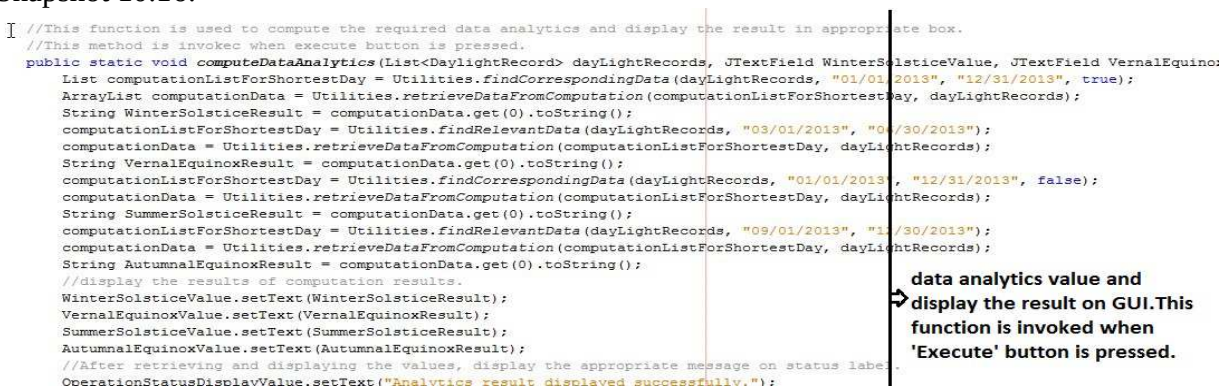
This function is invoked to write the deserialized object to file.

Snapshot 10.15:

```
//This function is used to compute the time difference between serialization and deserialization and displays it.
//This method is invoked when diplay delta time button is pressed from GUI.
public static void timeDiffSerandDeser(PersistentObject deserializedObj, JLabel OperationStatusDisplayValue, JLabel timeValue, long c
    if (deserializedObj != null) {
        long diff = currentTime - deserializedObj.getDateObj().getTime();
        //Compute hours, minutes, seconds  of the total simulation time executed
        long Hours = diff / (60 * 60 * 1000);
        long Minutes = diff / (60 * 1000) % 60;
        long Seconds = diff / 1000 % 60;
        timeValue.setText(Hours + "hr " + Minutes + "min " + Seconds + "sec.");
        //if success or failure then respective message is displayed.
        OperationStatusDisplayValue.setText("Time difference between serialization and deserialization successfully displayed.");
    } else {
        OperationStatusDisplayValue.setText("Perform deserialization and then try to retrieve the time difference.");
    }
```

code snippet to calculate and display the time between serialization and deserialization. The time result is displayed after calculation.

Snapshot 10.16:



```
//This function is used to compute the required data analytics and display the result in appropriate box.
//This method is invokec when execute button is pressed.
public static void computeDataAnalytics(List<DaylightRecord> dayLightRecords, JTextField WinterSolsticeValue, JTextField VernalEquinox
    List computationListForShortestDay = Utilities.findCorrespondingData(dayLightRecords, "01/01/2013", "12/31/2013", true);
    ArrayList computationData = Utilities.retrieveDataFromComputation(computationListForShortestDay, dayLightRecords);
    String WinterSolsticeResult = computationData.get(0).toString();
    computationListForShortestDay = Utilities.findRelevantData(dayLightRecords, "03/01/2013", "06/30/2013");
    computationData = Utilities.retrieveDataFromComputation(computationListForShortestDay, dayLightRecords);
    String VernalEquinoxResult = computationData.get(0).toString();
    computationListForShortestDay = Utilities.findCorrespondingData(dayLightRecords, "01/01/2013", "12/31/2013", false);
    computationData = Utilities.retrieveDataFromComputation(computationListForShortestDay, dayLightRecords);
    String SummerSolsticeResult = computationData.get(0).toString();
    computationListForShortestDay = Utilities.findRelevantData(dayLightRecords, "09/01/2013", "12/30/2013");
    computationData = Utilities.retrieveDataFromComputation(computationListForShortestDay, dayLightRecords);
    String AutumnalEquinoxResult = computationData.get(0).toString();
    //display the results of computation results.
    WinterSolsticeValue.setText(WinterSolsticeResult);
    VernalEquinoxValue.setText(VernalEquinoxResult);
    SummerSolsticeValue.setText(SummerSolsticeResult);
    AutumnalEquinoxValue.setText(AutumnalEquinoxResult);
    //After retrieving and displaying the values, display the appropriate message on status label.
    OperationStatusDisplayValue.setText("Analytics result displayed successfully.");
```

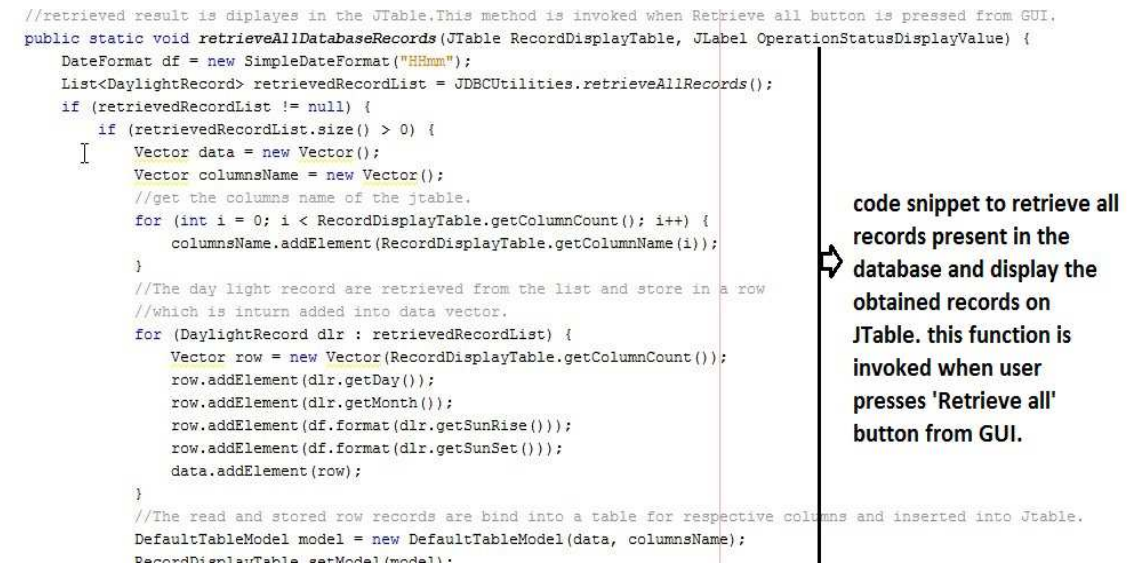
data analytics value and display the result on GUI.This function is invoked when 'Execute' button is pressed.

Snapshot 10.17:

```
//This function is used to reset all input text fields on GUI.
//This method is invoked when Reselt all input fields button is pressed.
public static void refreshTextFields(JTextField WinterSolsticeValue, JTextField SummerSolsticeValue, JTextField VernalEquino
    WinterSolsticeValue.setText("");
    SummerSolsticeValue.setText("");
    VernalEquinoxValue.setText("");
    AutumnalEquinoxValue.setText("");
    DayInputTextField.setText("");
    DeleteDayInputTextField.setText("");
    BeginDayInputTextField.setText("");
    EndDayInputTextField.setText("");
    MonthForDayComboValue.setSelectedItem(MonthForDayComboValue.getItemAt(0));
    MonthforRangeComboValue.setSelectedItem(MonthforRangeComboValue.getItemAt(0));
    DeleteMonthForDayComboValue.setSelectedItem(DeleteMonthForDayComboValue.getItemAt(0));
}
```

code snippet to reset all input fields. This function is invoked when used clicks 'Reset All Input Fields' button is pressed from GUI.

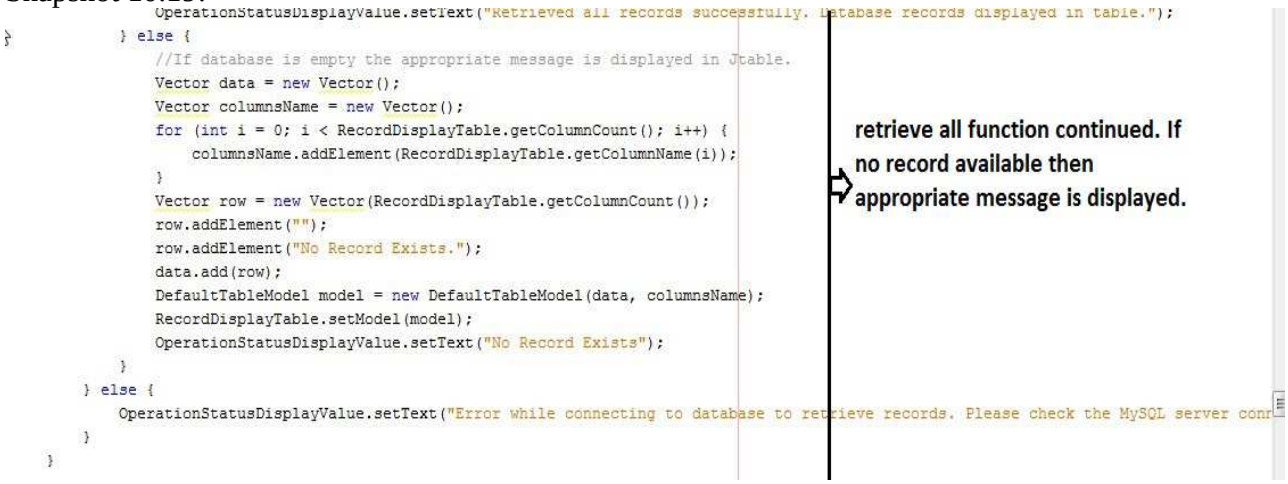
Snapshot 10.18:



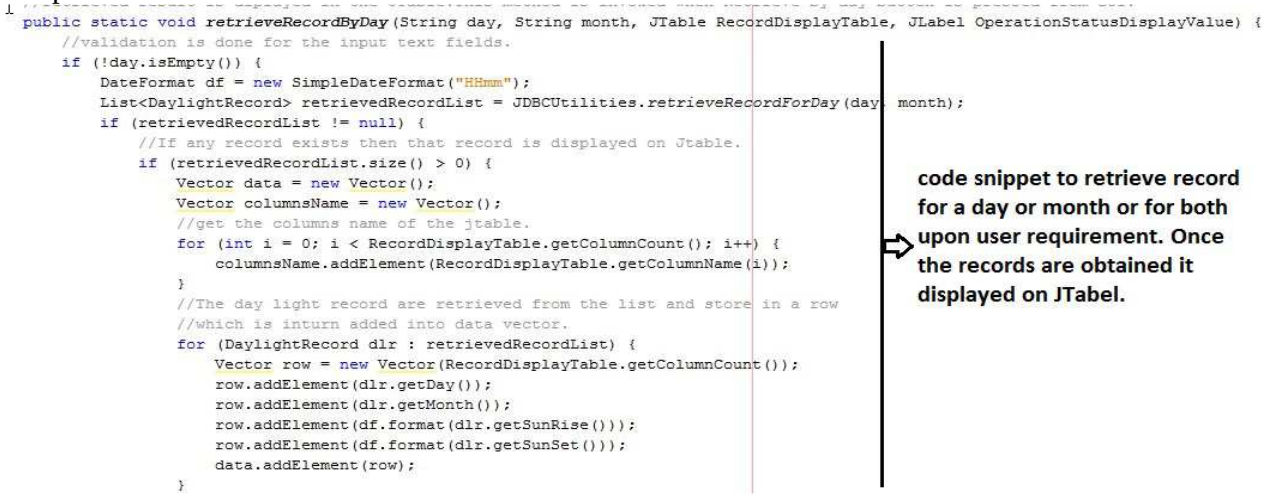
```
//retrieved result is diplayes in the JTable.This method is invoked when Retrieve all button is pressed from GUI.
public static void retrieveAllDatabaseRecords(JTable RecordDisplayTable, JLabel OperationStatusDisplayValue) {
    DateFormat df = new SimpleDateFormat("HHmm");
    List<DaylightRecord> retrievedRecordList = JDBCUtilities.retrieveAllRecords();
    if (retrievedRecordList != null) {
        if (retrievedRecordList.size() > 0) {
            Vector data = new Vector();
            Vector columnsName = new Vector();
            //get the columns name of the jtable.
            for (int i = 0; i < RecordDisplayTable.getColumnCount(); i++) {
                columnsName.addElement(RecordDisplayTable.getColumnName(i));
            }
            //The day light record are retrieved from the list and store in a row
            //which is inturn added into data vector.
            for (DaylightRecord dlr : retrievedRecordList) {
                Vector row = new Vector(RecordDisplayTable.getColumnCount());
                row.addElement(dlr.getDay());
                row.addElement(dlr.getMonth());
                row.addElement(df.format(dlr.getSunRise()));
                row.addElement(df.format(dlr.getSunSet()));
                data.addElement(row);
            }
            //The read and stored row records are bind into a table for respective columns and inserted into Jtable.
            DefaultTableModel model = new DefaultTableModel(data, columnsName);
            RecordDisplayTable.setModel(model);
```

code snippet to retrieve all records present in the database and display the obtained records on JTable. this function is invoked when user presses 'Retrieve all' button from GUI.

Snapshot 10.19:

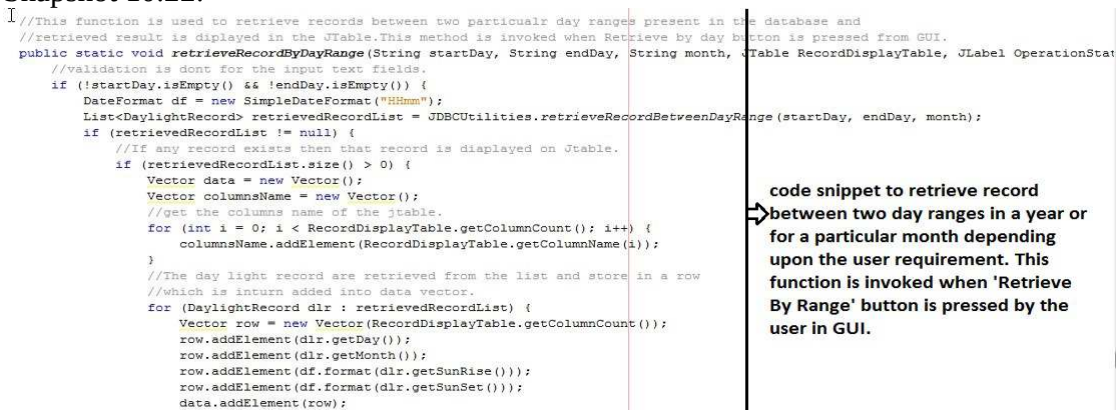


Snapshot 10.20:



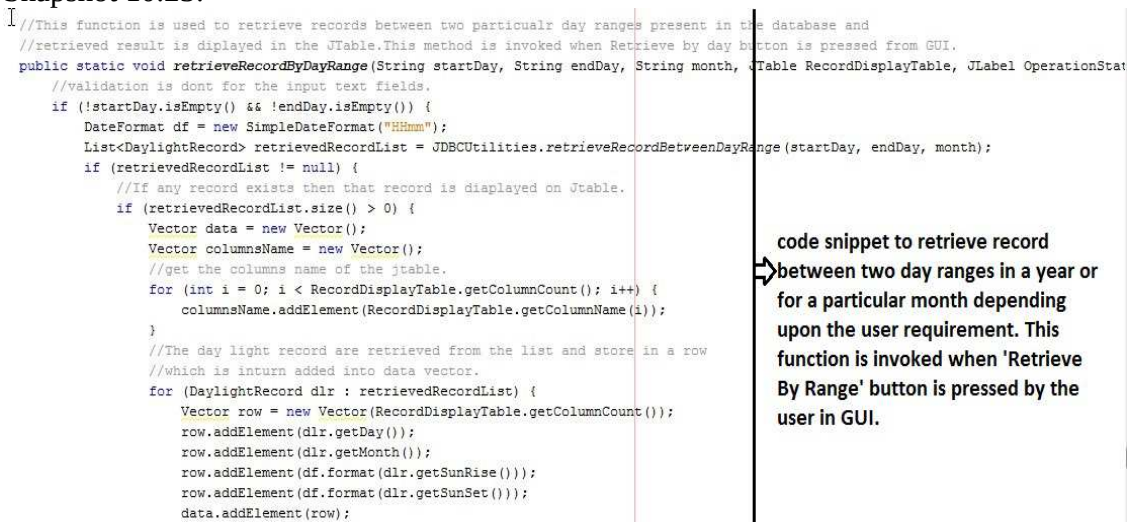
Snapshot 10.21:

Snapshot 10.22:




```
//This function is used to retrieve records between two particualr day ranges present in the database and
//retrieved result is diplayed in the JTable.This method is invoked when Retrieve by day button is pressed from GUI.
public static void retrieveRecordByDayRange(String startDay, String endDay, String month, JTable RecordDisplayTable, JLabel OperationStat
    //validation is dont for the input text fields.
    if (!startDay.isEmpty() && !endDay.isEmpty()) {
        DateFormat df = new SimpleDateFormat("HHmm");
        List<DaylightRecord> retrievedRecordList = JDBCUtilities.retrieveRecordBetweenDayRange(startDay, endDay, month);
        if (retrievedRecordList != null) {
            //If any record exists then that record is diaplayed on Jtable.
            if (retrievedRecordList.size() > 0) {
                Vector data = new Vector();
                Vector columnsName = new Vector();
                //get the columns name of the jtable.
                for (int i = 0; i < RecordDisplayTable.getColumnCount(); i++) {
                    columnsName.addElement(RecordDisplayTable.getColumnName(i));
                }
                //The day light record are retrieved from the list and store in a row
                //which is inturn added into data vector.
                for (DaylightRecord dlr : retrievedRecordList) {
                    Vector row = new Vector(RecordDisplayTable.getColumnCount());
                    row.addElement(dlr.getDay());
                    row.addElement(dlr.getMonth());
                    row.addElement(df.format(dlr.getSunRise()));
                    row.addElement(df.format(dlr.getSunSet()));
                    data.addElement(row);
```

Snapshot 10.23:




```
//This function is used to retrieve records between two particualr day ranges present in the database and
//retrieved result is diplayed in the JTable.This method is invoked when Retrieve by day button is pressed from GUI.
public static void retrieveRecordByDayRange(String startDay, String endDay, String month, JTable RecordDisplayTable, JLabel OperationStat
    //validation is dont for the input text fields.
    if (!startDay.isEmpty() && !endDay.isEmpty()) {
        DateFormat df = new SimpleDateFormat("HHmm");
        List<DaylightRecord> retrievedRecordList = JDBCUtilities.retrieveRecordBetweenDayRange(startDay, endDay, month);
        if (retrievedRecordList != null) {
            //If any record exists then that record is diaplayed on Jtable.
            if (retrievedRecordList.size() > 0) {
                Vector data = new Vector();
                Vector columnsName = new Vector();
                //get the columns name of the jtable.
                for (int i = 0; i < RecordDisplayTable.getColumnCount(); i++) {
                    columnsName.addElement(RecordDisplayTable.getColumnName(i));
                }
                //The day light record are retrieved from the list and store in a row
                //which is inturn added into data vector.
                for (DaylightRecord dlr : retrievedRecordList) {
                    Vector row = new Vector(RecordDisplayTable.getColumnCount());
                    row.addElement(dlr.getDay());
                    row.addElement(dlr.getMonth());
                    row.addElement(df.format(dlr.getSunRise()));
                    row.addElement(df.format(dlr.getSunSet()));
                    data.addElement(row);
```

11. Driver (MainFrame) class:

Snapshot 11.1:


```
// Variables declaration - do not modify
    private javax.swing.JTextField AutumnalEquinoxValue;
    private javax.swing.JTextField BeginDayInputTextField;
    private javax.swing.JLabel BeginDayLabel;
    private javax.swing.JButton CreateRecordButton;
    private javax.swing.JTextField DayInputTextField;
    private javax.swing.JLabel DayLabel;
    private javax.swing.JLabel DayLabel1;
    private javax.swing.JTextField DeleteDayInputTextField;
    private javax.swing.JComboBox DeleteMonthForDayComboValue;
    private javax.swing.JButton DeleteRecordButton;
    private javax.swing.JTextField EndDayInputTextField;
    private javax.swing.JLabel EndDayLabel;
    private javax.swing.JComboBox MonthForDayComboValue;
    private javax.swing.JLabel MonthLabel;
    private javax.swing.JLabel MonthLabel1;
    private javax.swing.JLabel MonthLabel2;
    private javax.swing.JComboBox MonthforRangeComboValue;
    private javax.swing.JLabel OperationStatusDisplay;
    private static javax.swing.JLabel OperationStatusDisplayValue;
    private static javax.swing.JTable RecordDisplayTable;
    private javax.swing.JButton RefreshDatabase;
    private javax.swing.JButton RefreshTextFields;
    private javax.swing.JButton RetrieveAllRecordButton;
```

Snapshot 11.2:

```
/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)
    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            //create database table and insert all daylight records into database.
            List<DaylightRecord> dlr = Utilities.readCSVfile();
            boolean value = JDBCUtilities.initByCreatingandPopulatingTableInDatabase(dlr);
            //if insert successful or failure then start the mainframe with appropriate message in it.
            if (value) {
                new MainFrame().setVisible(true);
                GUIUtilities.retrieveAllDatabaseRecords(RecordDisplayTable, OperationStatusDisplayValue);
                OperationStatusDisplayValue.setText("Database table daylightrecord created and records inserted su
            } else {
                new MainFrame().setVisible(true);
                OperationStatusDisplayValue.setText("Database table daylightrecord not created. Please check the M
            }
        }
    });
```

code snippet from where execution of the program starts. When the program is launched it creates the database table and insert data into the table from given CSV file and displayes GUI displaying succes or failure message related to table creation and insetion of records.

Snapshot 11.3:

```
//This event is triggered when Refresh All Text field button is pressed.
private void RefreshTextFieldsActionPerformed(java.awt.event.ActionEvent evt) {
    OperationStatusDisplayValue.setText("Processing.....");
    //This to reset all text fields after user entering the input data.
    GUIUtilities.refreshTextFields(WinterSolsticeValue, SummerSolsticeValue, VernalEquinoxValue, AutumnalEquinoxVa
    OperationStatusDisplayValue.setText("Refreshed Input Text Fields Succesful.");
}

/**
```

Event triggered when Refresh Input Fields' button is pressed. Resets all input fields of GUI.

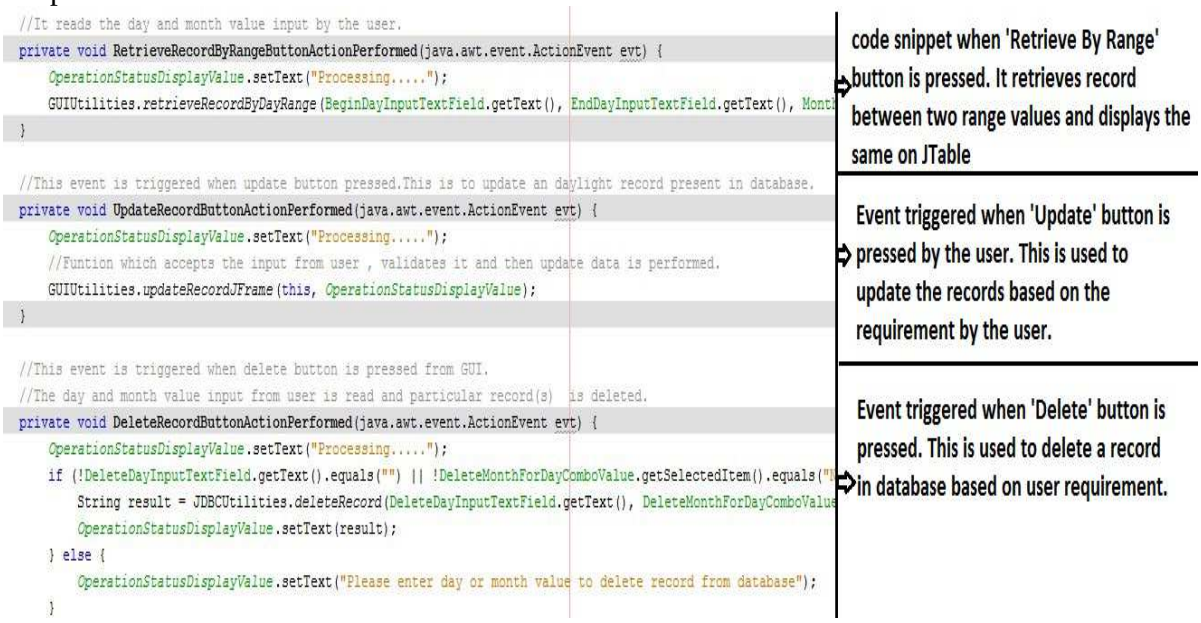
Snapshot 11.4:

```
//This event is triggred when didplay delta time is pressed.
//displays the time difference between serialization and deserialization.
private void displayTimeButtonActionPerformed(java.awt.event.ActionEvent evt) {
    OperationStatusDisplayValue.setText("Processing.....");
    GUIUtilities.timeDiffSerandDeser(deserializePObj, OperationStatusDisplayValue, t
}

//This event is triggered when Refresh database button is pressed.
private void RefreshDatabaseActionPerformed(java.awt.event.ActionEvent evt) {
    OperationStatusDisplayValue.setText("Processing.....");
    //Reload database with read CSV file after performing any insertion and deletion
    //original database value.
    List<DaylightRecord> dlr = Utilities.readCSVfile();
    boolean value = JDBCUtilities.initByCreatingandPopulatingTableInDatabase(dlr);
    if (value) {
        OperationStatusDisplayValue.setText("Successfully reloaded the database u
    } else {
        OperationStatusDisplayValue.setText("Database reload failed.Check MySQL s
    }
}
```

Event triggered when Display Delta time' is pressed. It calculates and displayes the time difference between serialization and deserialization performed.

Event triggered when 'Reload Database' button is pressed. Loads database with data from CSV file.

Snapshot 11.5:



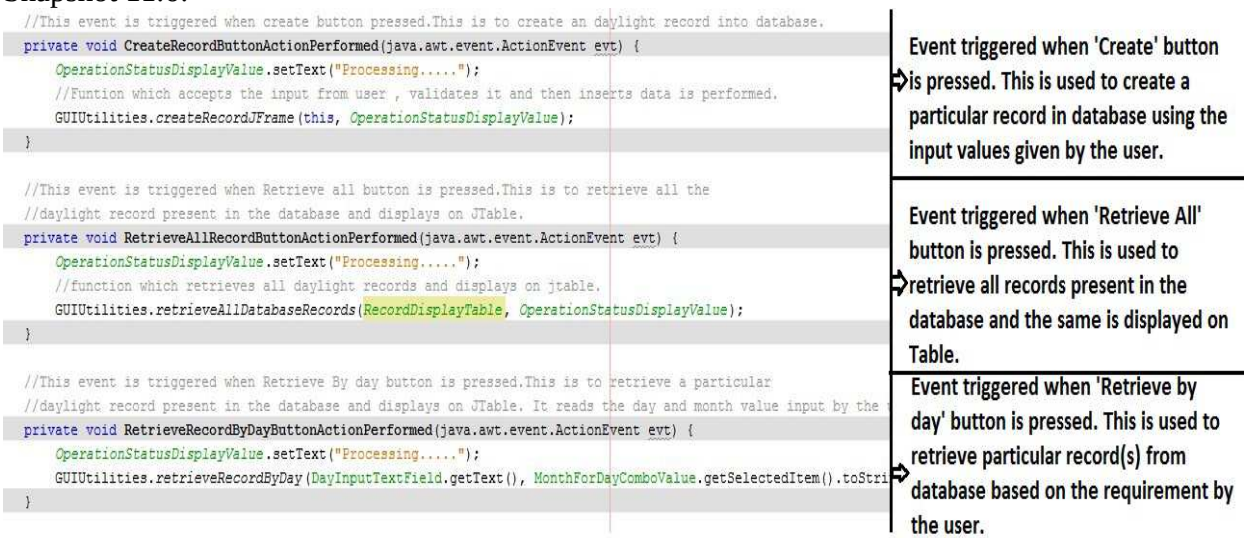
```
//It reads the day and month value input by the user.
private void RetrieveRecordByRangeButtonActionPerformed(java.awt.event.ActionEvent evt) {
    OperationStatusDisplayValue.setText("Processing.....");
    GUIUtilities.retrieveRecordByDayRange(BeginDayInputTextField.getText(), EndDayInputTextField.getText(), Month
}

//This event is triggered when update button pressed.This is to update an daylight record present in database.
private void UpdateRecordButtonActionPerformed(java.awt.event.ActionEvent evt) {
    OperationStatusDisplayValue.setText("Processing.....");
    //Funtion which accepts the input from user , validates it and then update data is performed.
    GUIUtilities.updateRecordJFrame(this, OperationStatusDisplayValue);
}

//This event is triggered when delete button is pressed from GUI.
//The day and month value input from user is read and particular record(s)  is deleted.
private void DeleteRecordButtonActionPerformed(java.awt.event.ActionEvent evt) {
    OperationStatusDisplayValue.setText("Processing.....");
    if (!DeleteDayInputTextField.getText().equals("") || !DeleteMonthForDayComboValue.getSelectedItem().equals("
        String result = JDBCUtilities.deleteRecord(DeleteDayInputTextField.getText(), DeleteMonthForDayComboValue
        OperationStatusDisplayValue.setText(result);
    } else {
        OperationStatusDisplayValue.setText("Please enter day or month value to delete record from database");
    }
```

code snippet when 'Retrieve By Range' button is pressed. It retrieves record between two range values and displays the same on JTable

Event triggered when 'Update' button is pressed by the user. This is used to update the records based on the requirement by the user.

Event triggered when 'Delete' button is pressed. This is used to delete a record in database based on user requirement.

Snapshot 11.6:



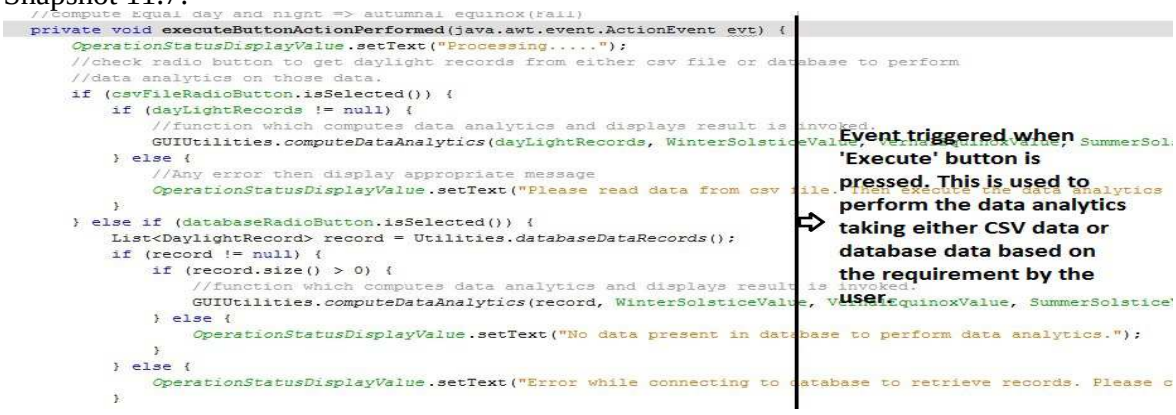
```
//This event is triggered when create button pressed.This is to create an daylight record into database.
private void CreateRecordButtonActionPerformed(java.awt.event.ActionEvent evt) {
    OperationStatusDisplayValue.setText("Processing.....");
    //Funtion which accepts the input from user , validates it and then inserts data is performed.
    GUIUtilities.createRecordJFrame(this, OperationStatusDisplayValue);
}

//This event is triggered when Retrieve all button is pressed.This is to retrieve all the
//daylight record present in the database and displays on JTable.
private void RetrieveAllRecordButtonActionPerformed(java.awt.event.ActionEvent evt) {
    OperationStatusDisplayValue.setText("Processing.....");
    //function which retrieves all daylight records and displays on jtable.
    GUIUtilities.retrieveAllDatabaseRecords(RecordDisplayTable, OperationStatusDisplayValue);
}

//This event is triggered when Retrieve By day button is pressed.This is to retrieve a particular
//daylight record present in the database and displays on JTable. It reads the day and month value input by the
private void RetrieveRecordByDayButtonActionPerformed(java.awt.event.ActionEvent evt) {
    OperationStatusDisplayValue.setText("Processing.....");
    GUIUtilities.retrieveRecordByDay(DayInputTextField.getText(), MonthForDayComboValue.getSelectedItem().toStri
}
```

Event triggered when 'Create' button is pressed. This is used to create a particular record in database using the input values given by the user.

Event triggered when 'Retrieve All' button is pressed. This is used to retrieve all records present in the database and the same is displayed on Table.

Event triggered when 'Retrieve by day' button is pressed. This is used to retrieve particular record(s) from database based on the requirement by the user.

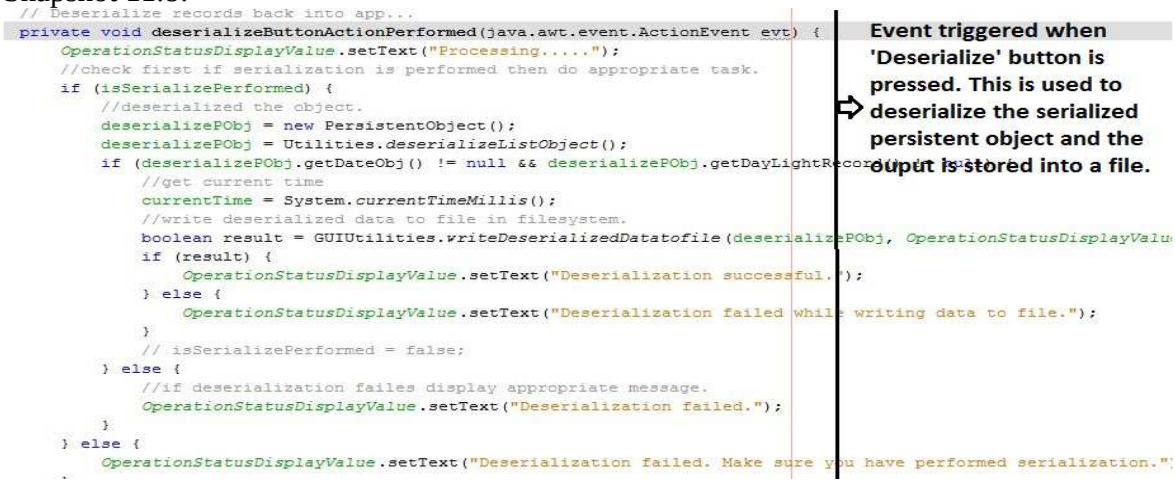
Snapshot 11.7:



```
//Compute equal day and night => autumnal equinox(fall)
private void executeButtonActionPerformed(java.awt.event.ActionEvent evt) {
    OperationStatusDisplayValue.setText("Processing.....");
    //check radio button to get daylight records from either csv file or database to perform
    //data analytics on those data.
    if (csvFileRadioButton.isSelected()) {
        if (dayLightRecords != null) {
            //function which computes data analytics and displays result is invoked.
            GUIUtilities.computeDataAnalytics(dayLightRecords, WinterSolsticeValue, VernalEquinoxValue, SummerSol
        } else {
            //Any error then display appropriate message
            OperationStatusDisplayValue.setText("Please read data from csv file. Then execute the data analytics
        }
    } else if (databaseRadioButton.isSelected()) {
        List<DaylightRecord> record = Utilities.databaseDataRecords();
        if (record != null) {
            if (record.size() > 0) {
                //function which computes data analytics and displays result is invoked.
                GUIUtilities.computeDataAnalytics(record, WinterSolsticeValue, VernalEquinoxValue, SummerSolstice
            } else {
                OperationStatusDisplayValue.setText("No data present in database to perform data analytics.");
            }
        } else {
            OperationStatusDisplayValue.setText("Error while connecting to database to retrieve records. Please c
        }
```

Event triggered when 'Execute' button is pressed. This is used to perform the data analytics taking either CSV data or database data based on the requirement by the user.

Snapshot 11.8:



Snapshot 11.9:

Snapshot 11.10:

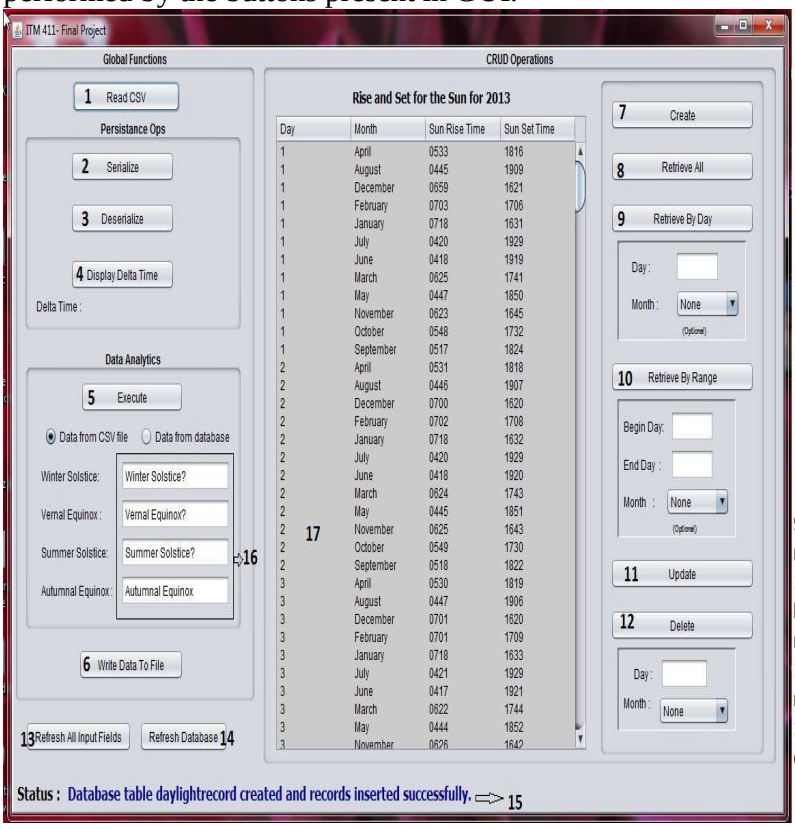
Snapshot 11.11:

```
//Event invoked when write to file button is pressed.
//This is to write all displayed data to a text file called daylight-records.txt.
private void WriteToFileButtonActionPerformed(java.awt.event.ActionEvent evt) {
    OperationStatusDisplayValue.setText("Processing.....");
    if (dayLightRecords != null) {
        //write all daylightRecords to text file daylight-Record.txt
        Utilities.writeListtoFile(dayLightRecords);
    } else {
        OperationStatusDisplayValue.setText("Please read CSV file then try to write daylightrecords to a file.");
    }
}
```

Event triggered when 'Write to file' button is pressed. This function is used to write all daylight records into file in filesystem.

12. Output:

Snapshot 12.1:
When the application is launched the GUI appears with message successfully inserting table and values into it. The below snapshot represents the GUI and explains the task performed by the buttons present in GUI.

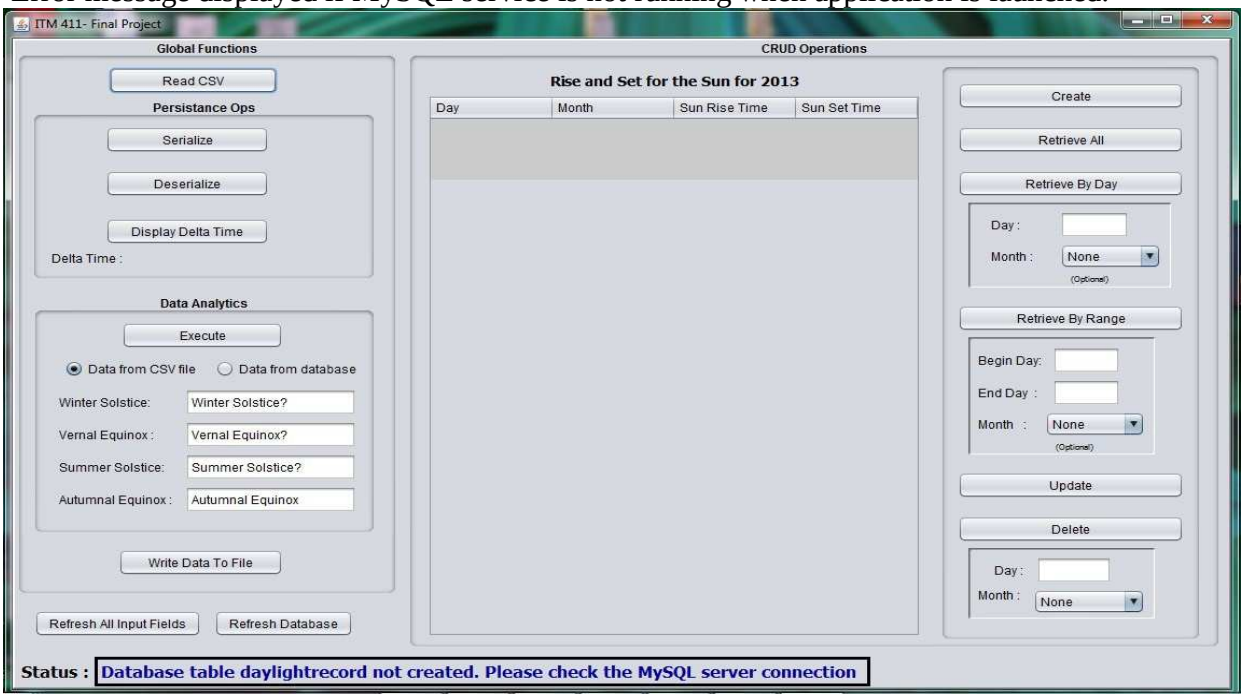


1. To read csv file data
2. Serialize persistentobject pObj
3.DeSerialize persistentobject pObj
4. Difference between serialize and deserialize
5. To compute data analytics.
6.write all read daylight records into a file.
7.Insert record into database.
8. Retrieve all records present in the database.
9. Retrieve particular record from database.
10.Retrieve records between two day ranges from database.
11. update particular record in database.
12.delete record from database.

13.Reset all input textfields.
14.Reload database values.
15. whenever a operation is performed its status is displayed.
16. Results of data analytics are displayed.
17. databse related results are displayed.

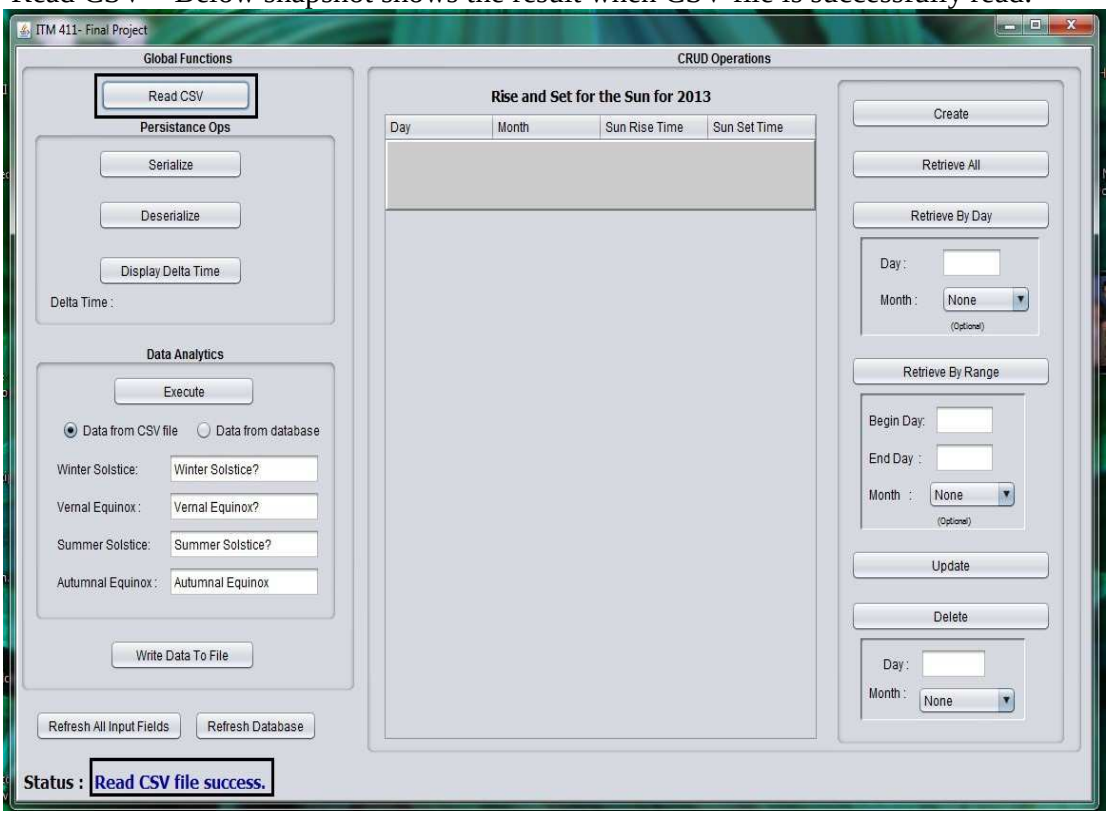
Snapshot 12.2:

Error message displayed if MySQL service is not running when application is launched.

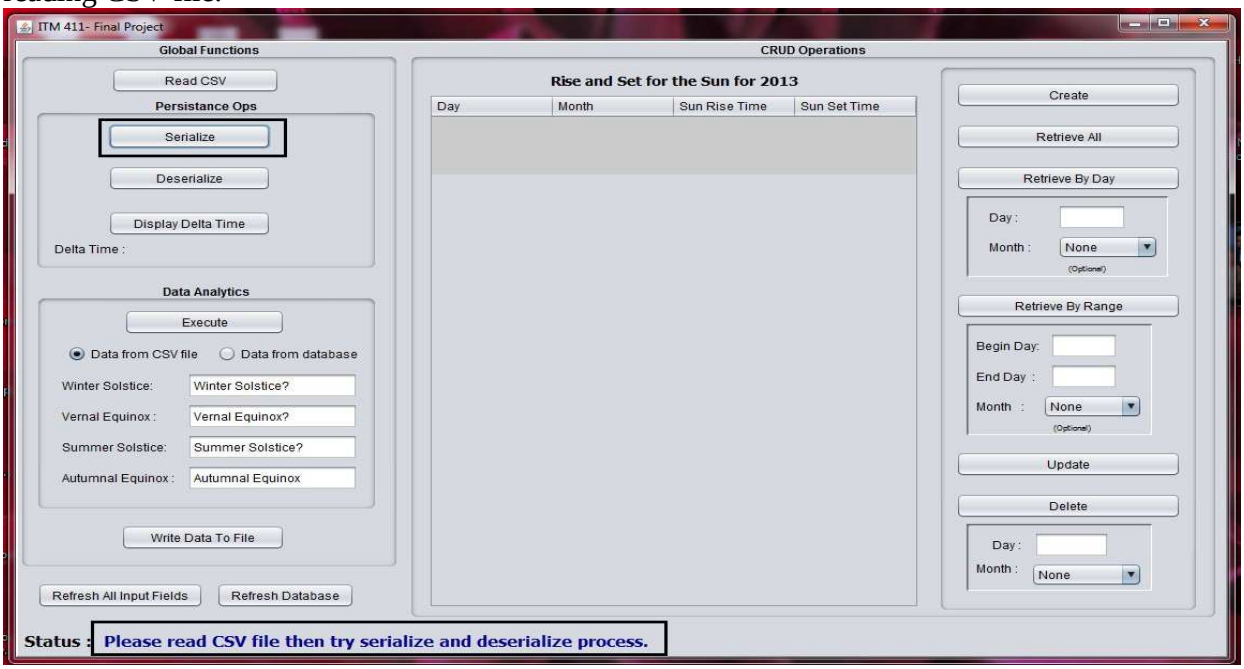


Snapshot 12.3:

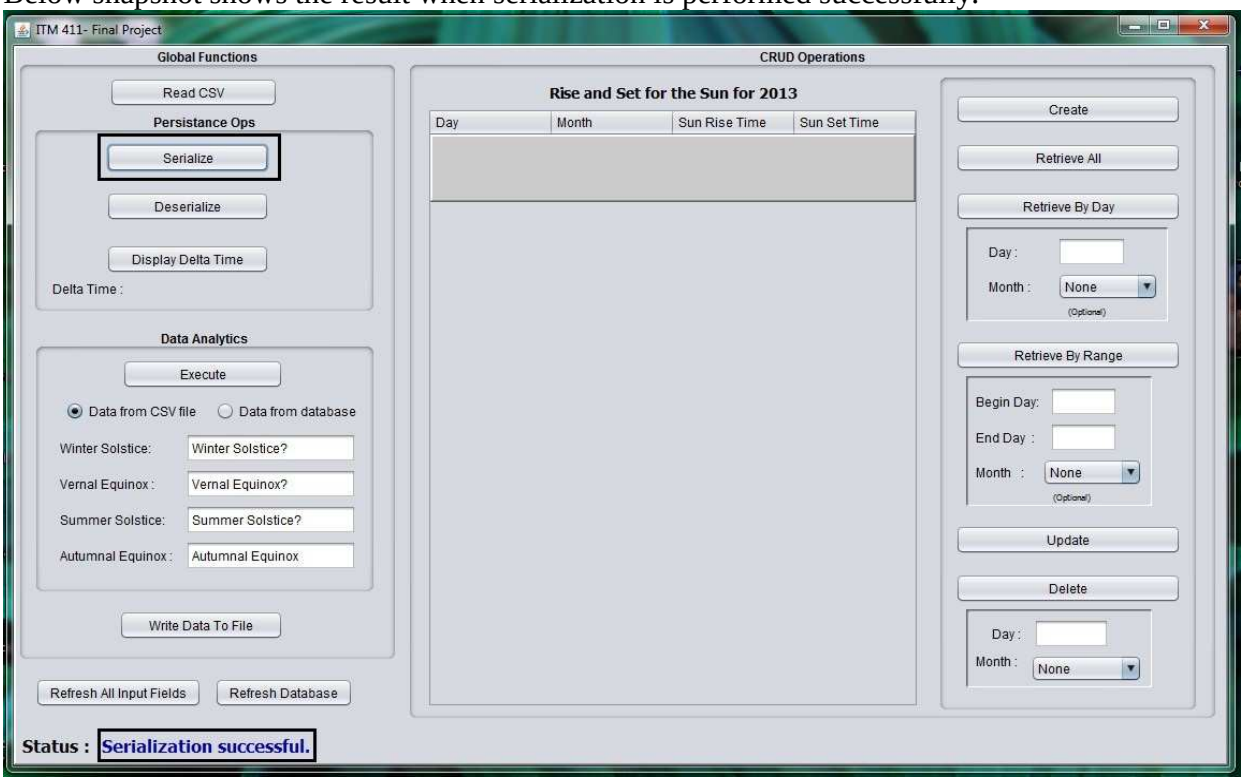
Read CSV – Below snapshot shows the result when CSV file is successfully read.



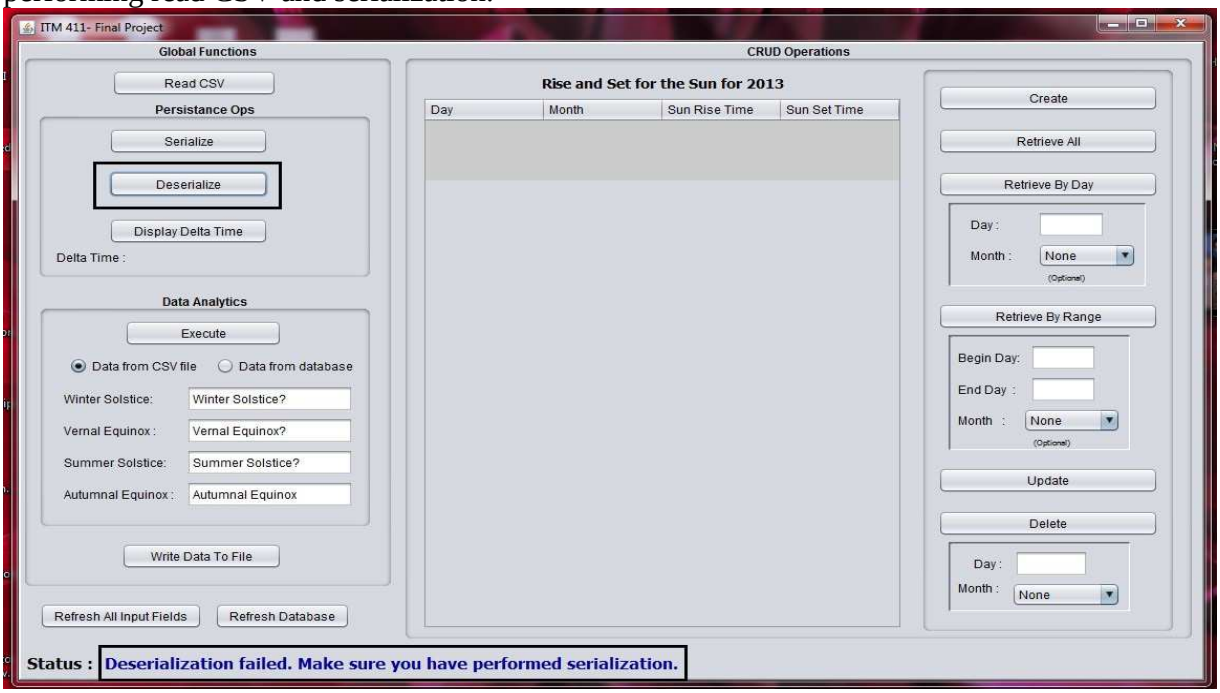
Snapshot 12.4:
Below snapshot shows the error message when user tries to perfor serialization before reading CSV file.



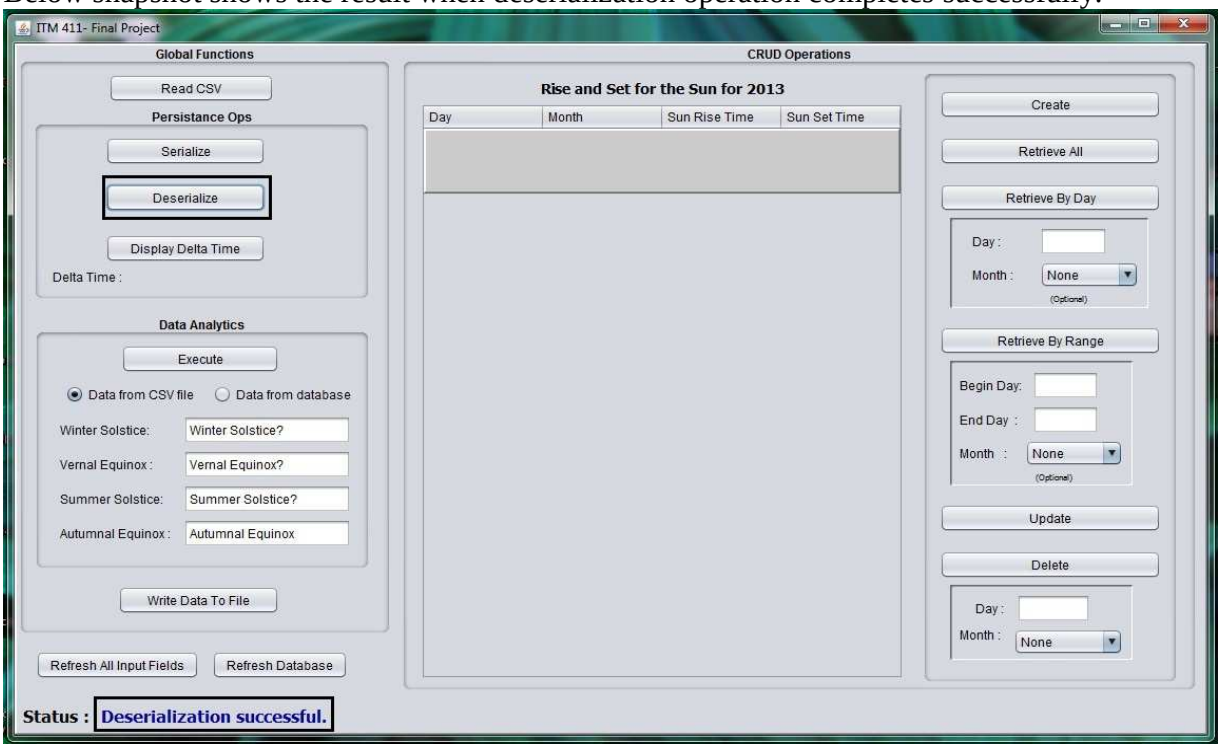
Snapshot 12.5:
Below snapshot shows the result when serialization is performed successfully.



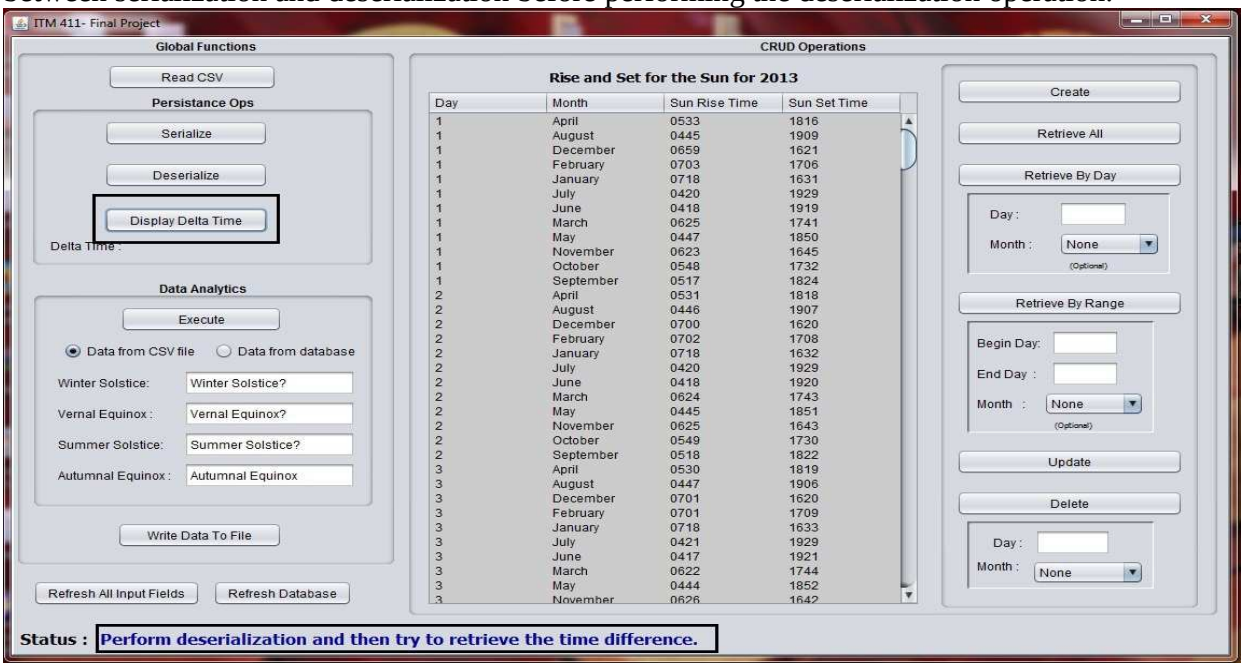
Snapshot 12.6:
Below snapshot shows the result when user tries to perform deserialization before performing read CSV and serialization.



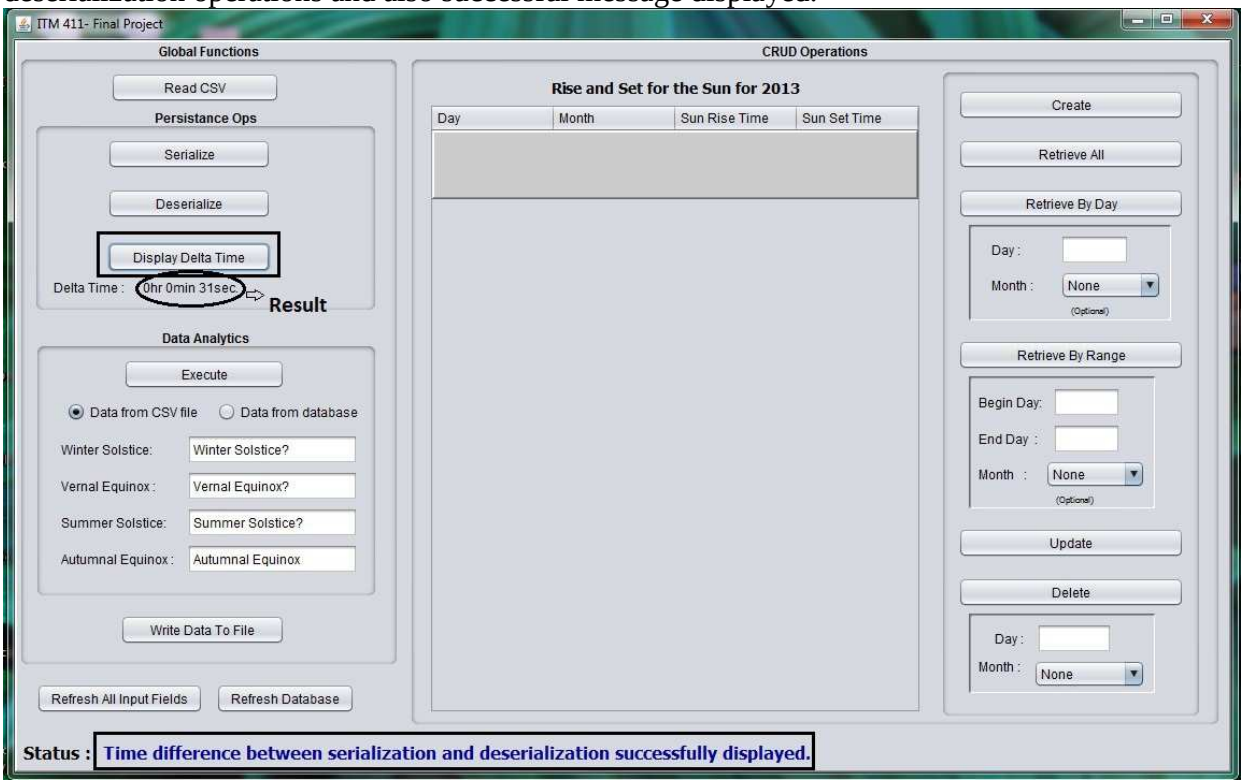
Snapshot 12.7:
Below snapshot shows the result when deserialization operation completes successfully.



Snapshot 12.8:
Below snapshot shows the error result displayed when user tries to retrieve difference between serialization and deserialization before performing the deserialization operation.

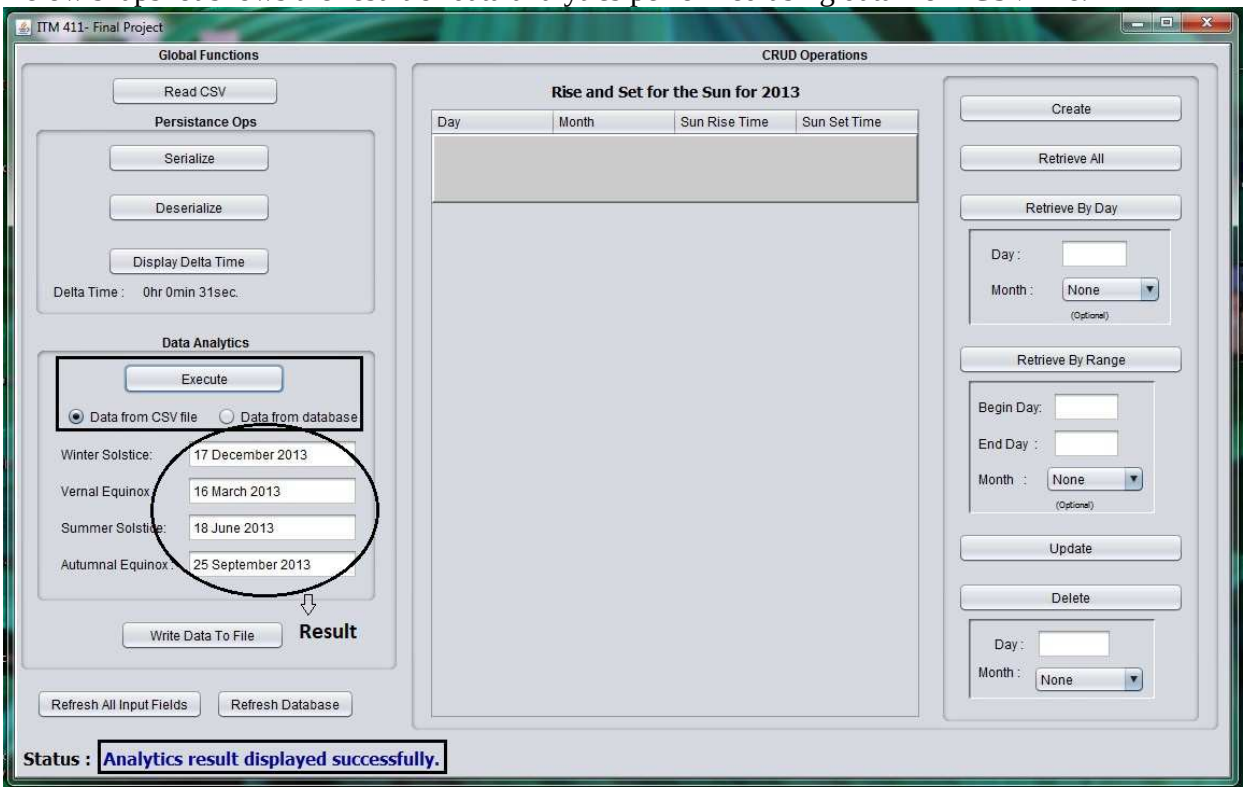


Snapshot 12.9:
Below snapshot shows the result of time retrieved between serialization and deserialization operations and also successful message displayed.

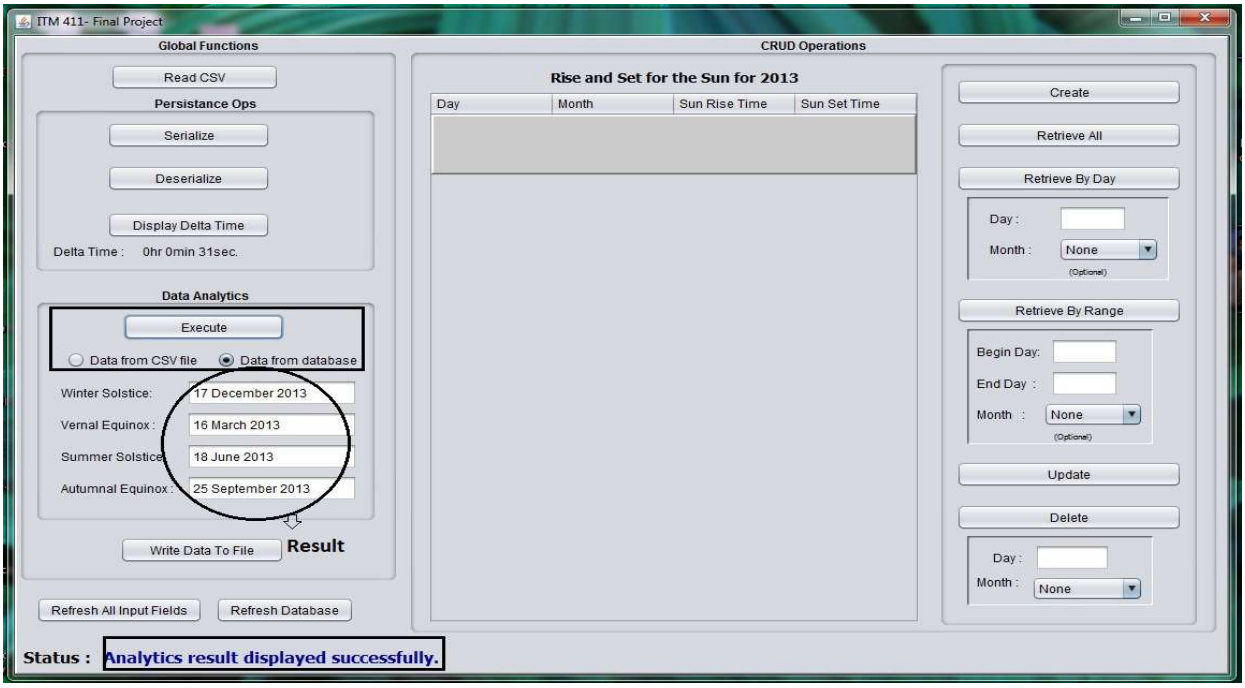


Snapshot 12.10:
Below snapshot shows the error result displayed when user tries to perform data analytics operation before reading the CSV file.

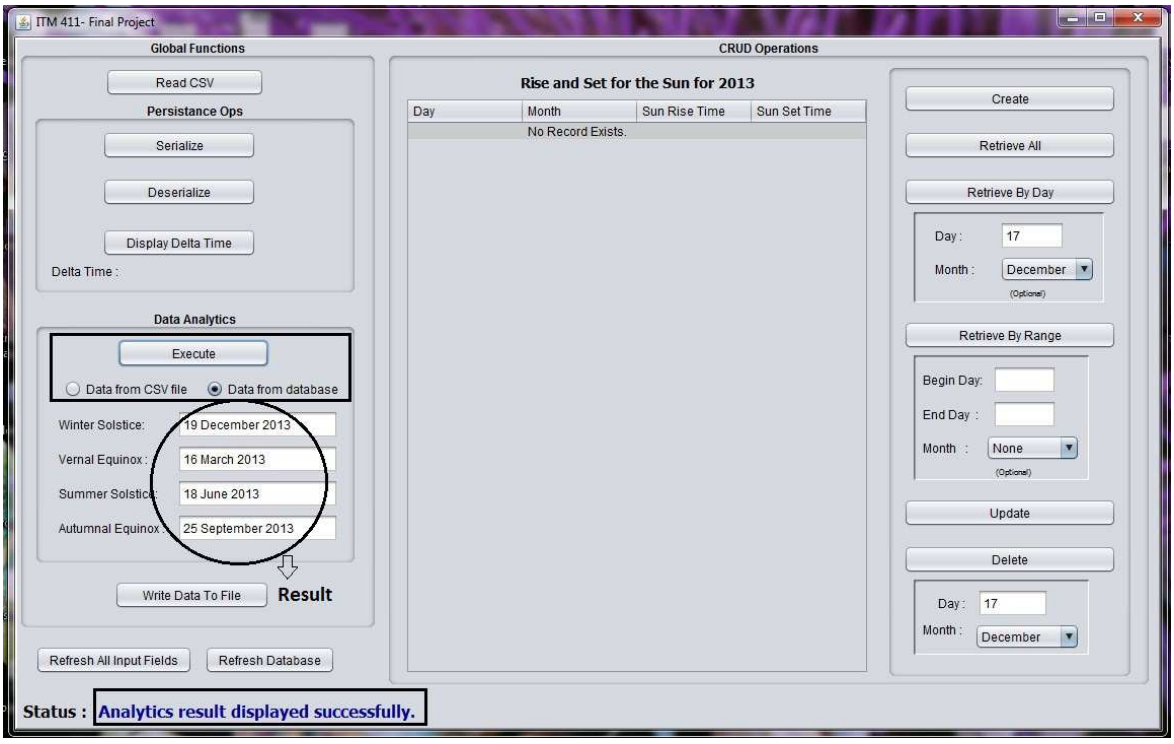
Snapshot 12.11:
Below snapshot shows the result of data analytics performed using data from CSV file.



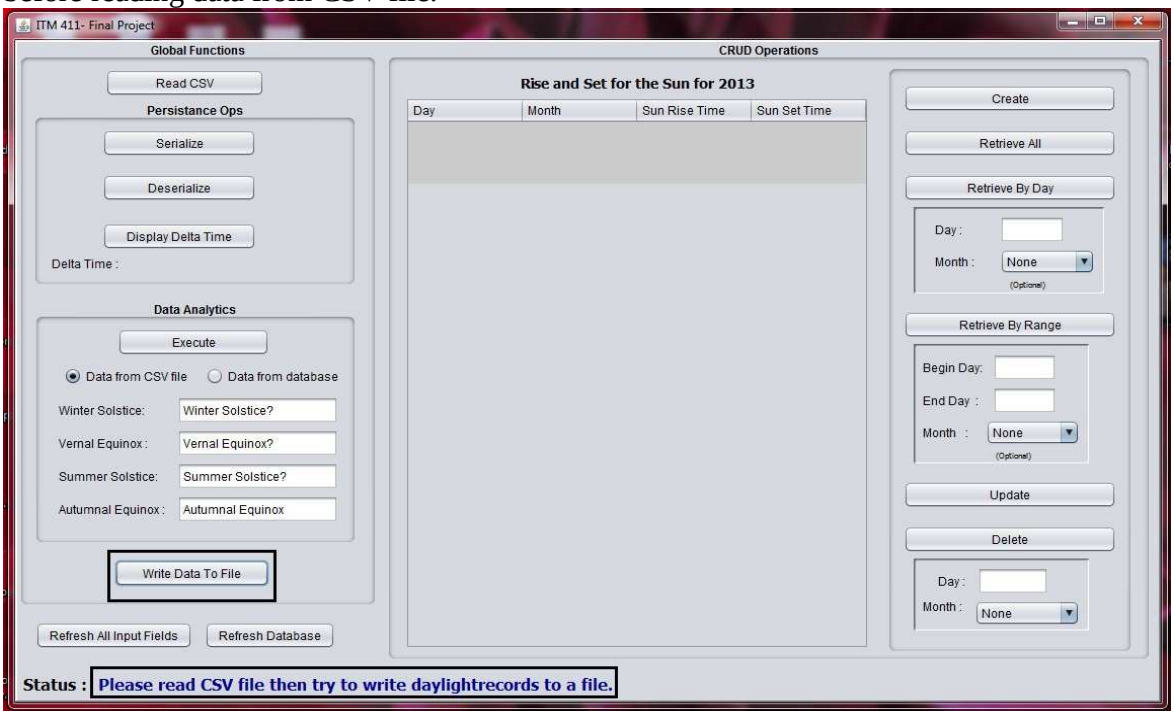
Snapshot 12.12:
Below snapshot shows the results displayed when data analytics is performed using database data.



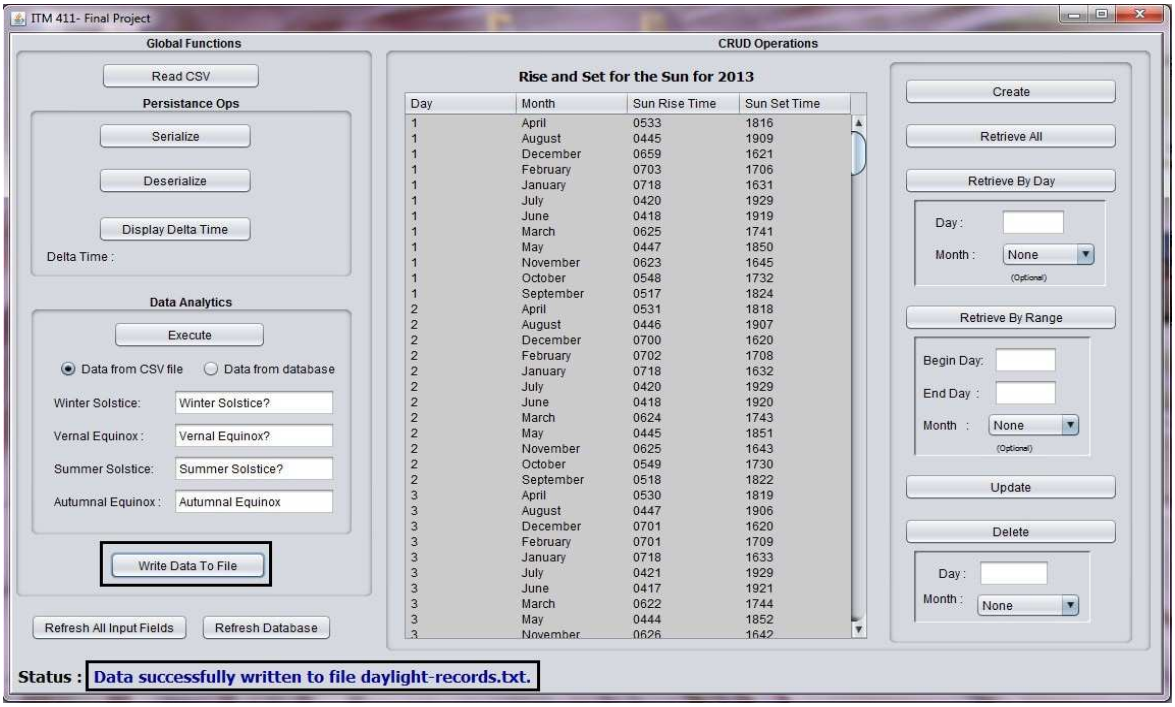
Snapshot 12.13:
Below snapshot shows the results displayed when data analytics is performed using database data.



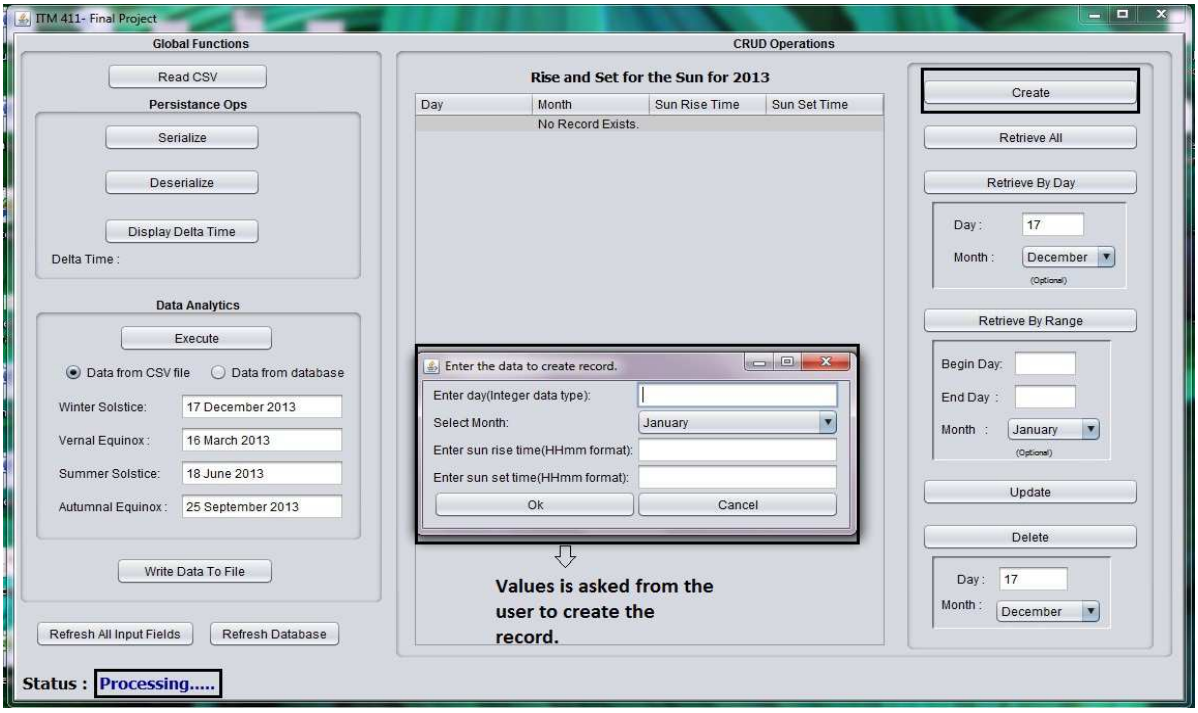
Snapshot 12.14:
Below snapshot shows the error result displayed when used tries to write data to file before reading data from CSV file.



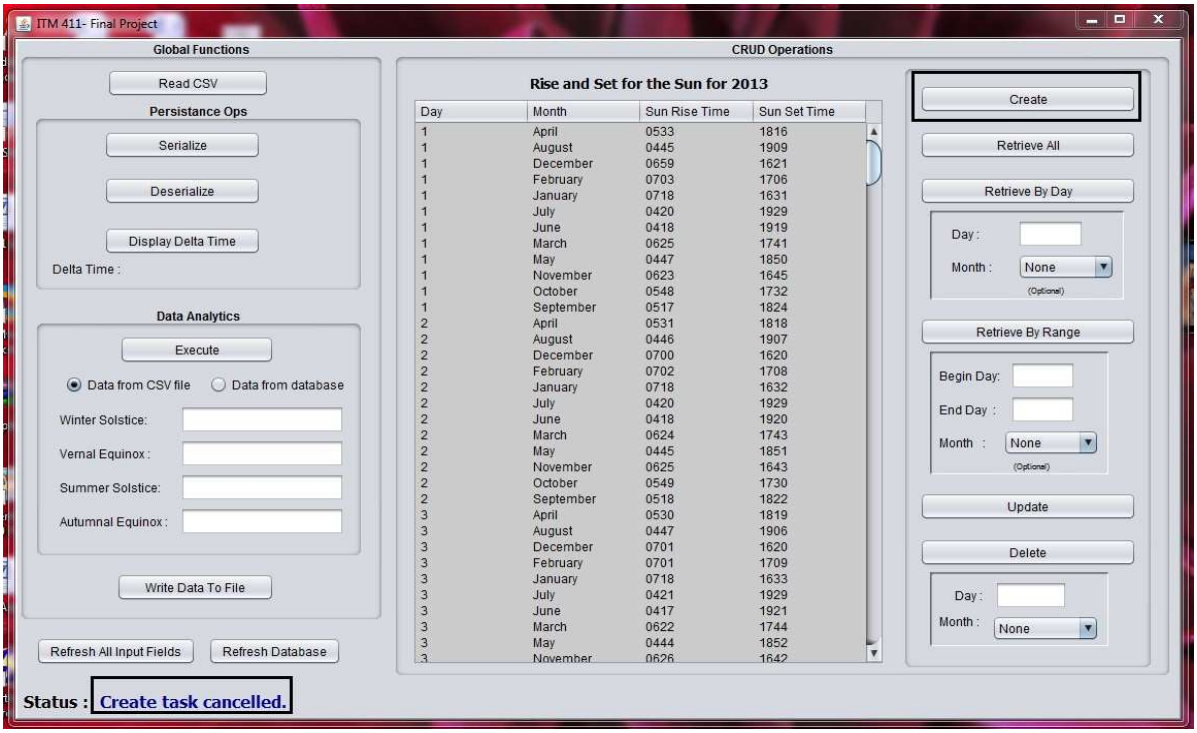
Snapshot 12.15:
Below snapshot shows the results displayed when data analytics is performed using database data.



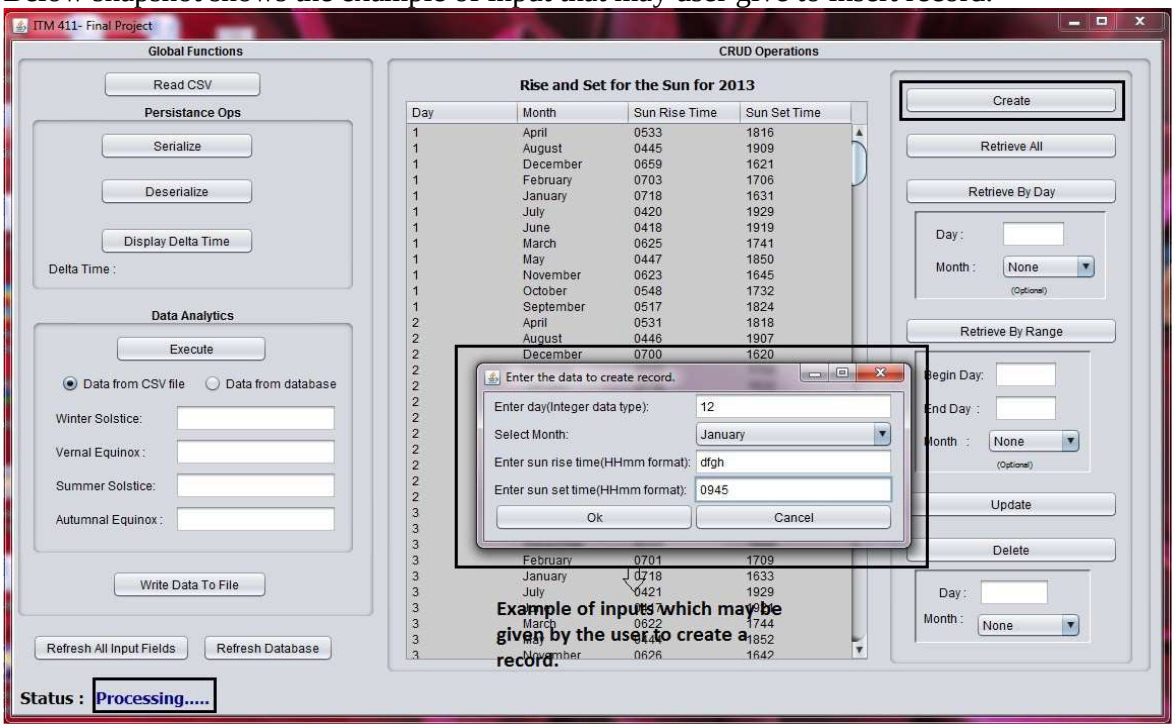
Snapshot 12.16:
Below snapshot shows the window that pops up to accept input values when user tries to create a record in the database.



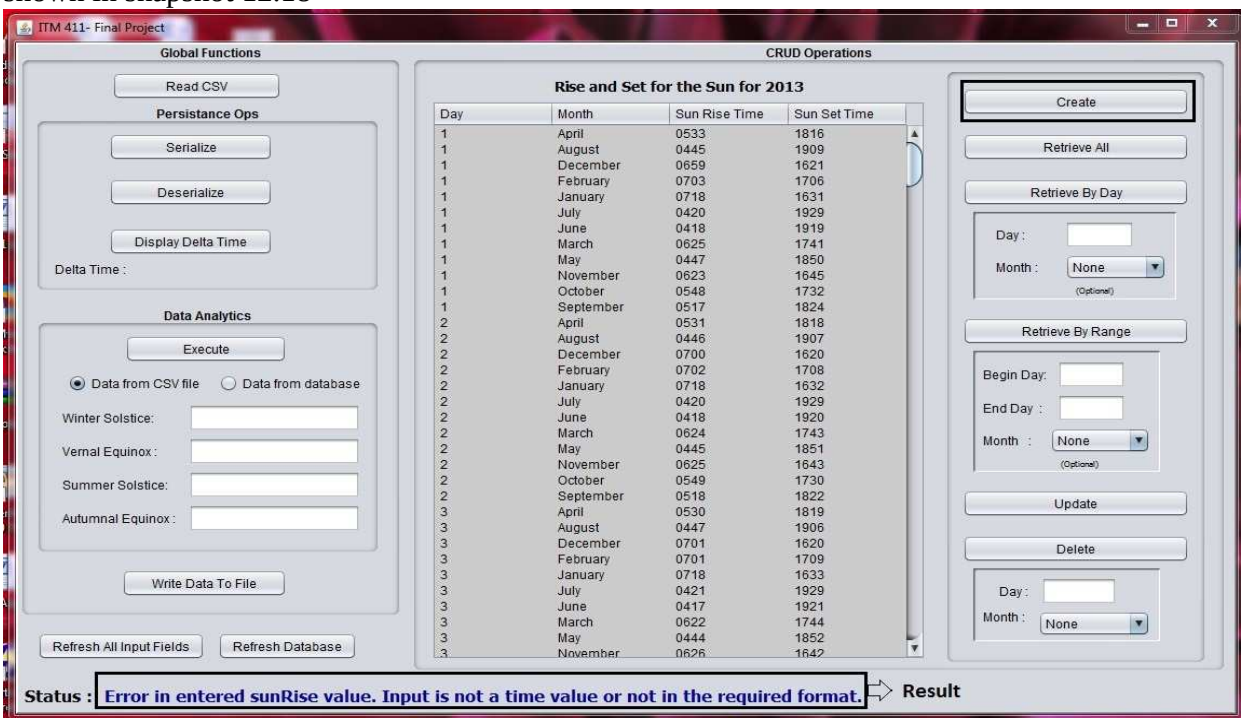
Snapshot 12.17:
Below snapshot shows the results displayed when user cancel the popped up window ie. When user cancels the idea to insert record into database.



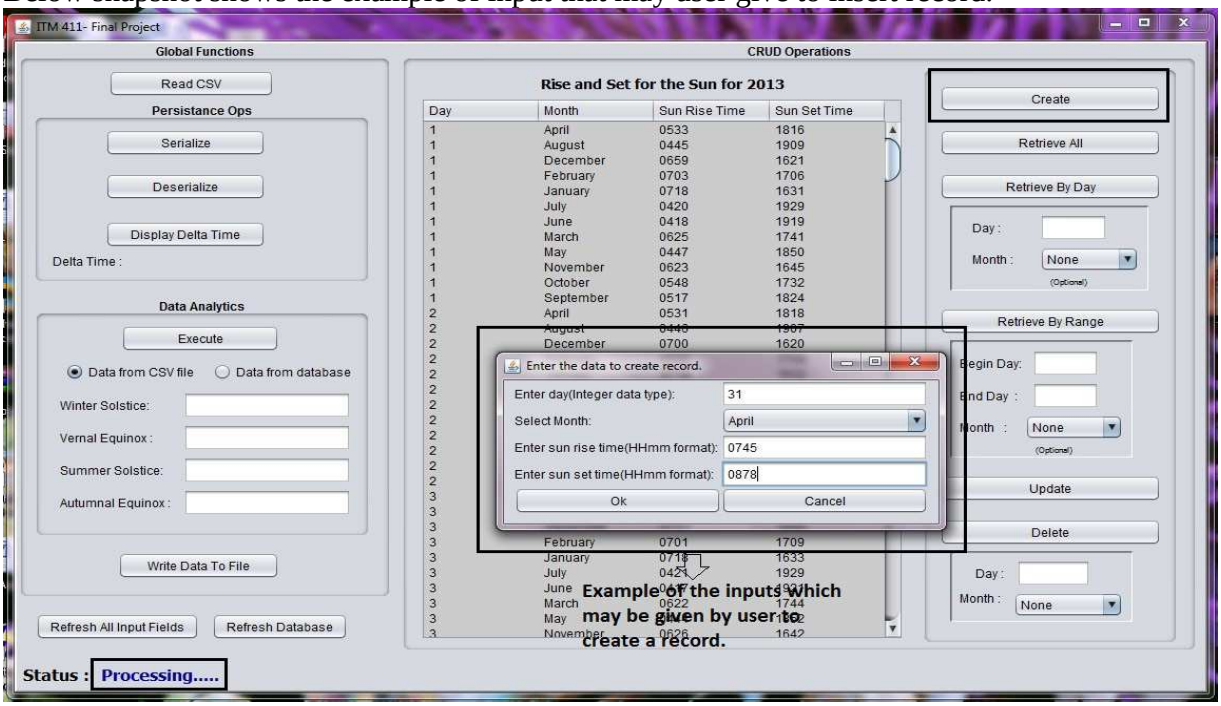
Snapshot 12.18:
Below snapshot shows the example of input that may user give to insert record.



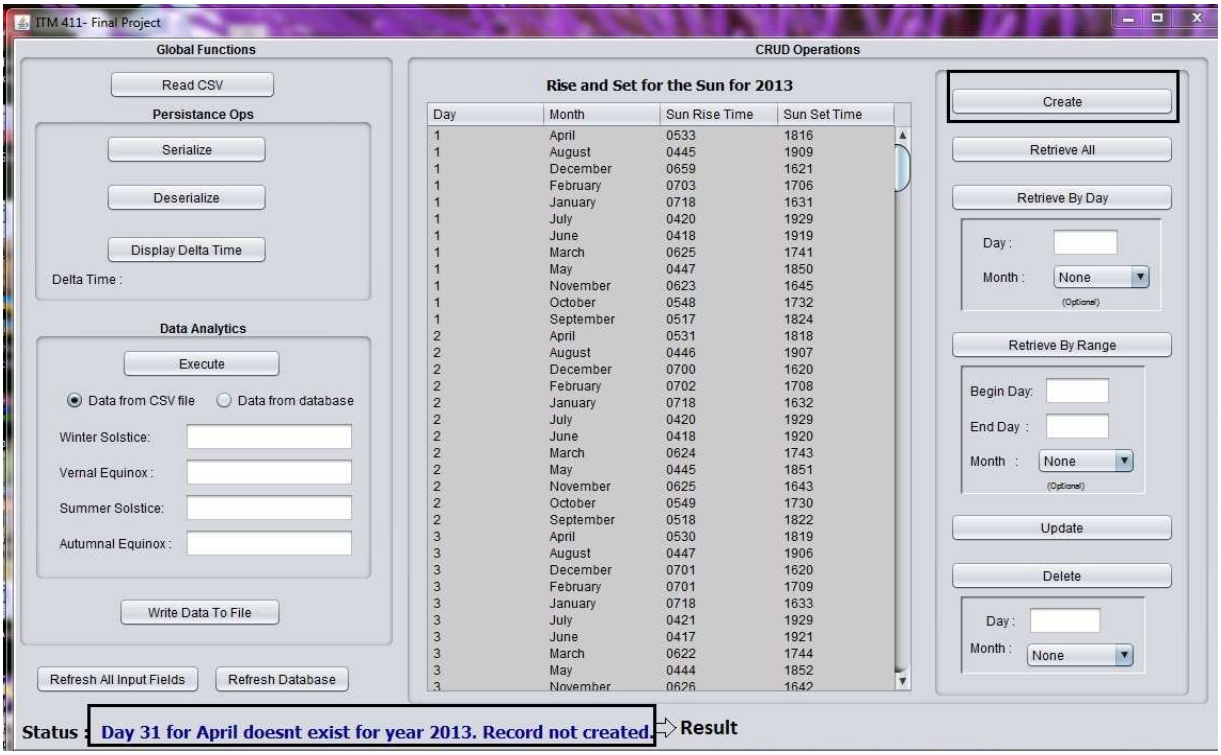
Snapshot 12.19:
Below snapshot shows the result displayed of when user tries to insert record values shown in snapshot 12.18



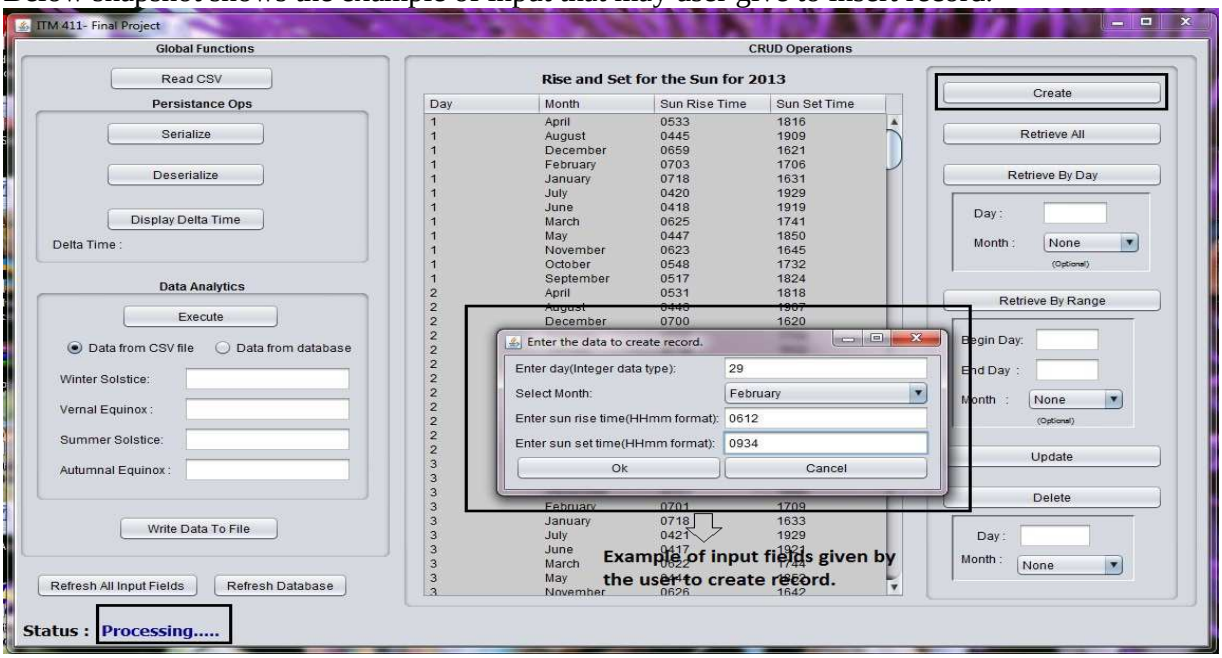
Snapshot 12.20:
Below snapshot shows the example of input that may user give to insert record.



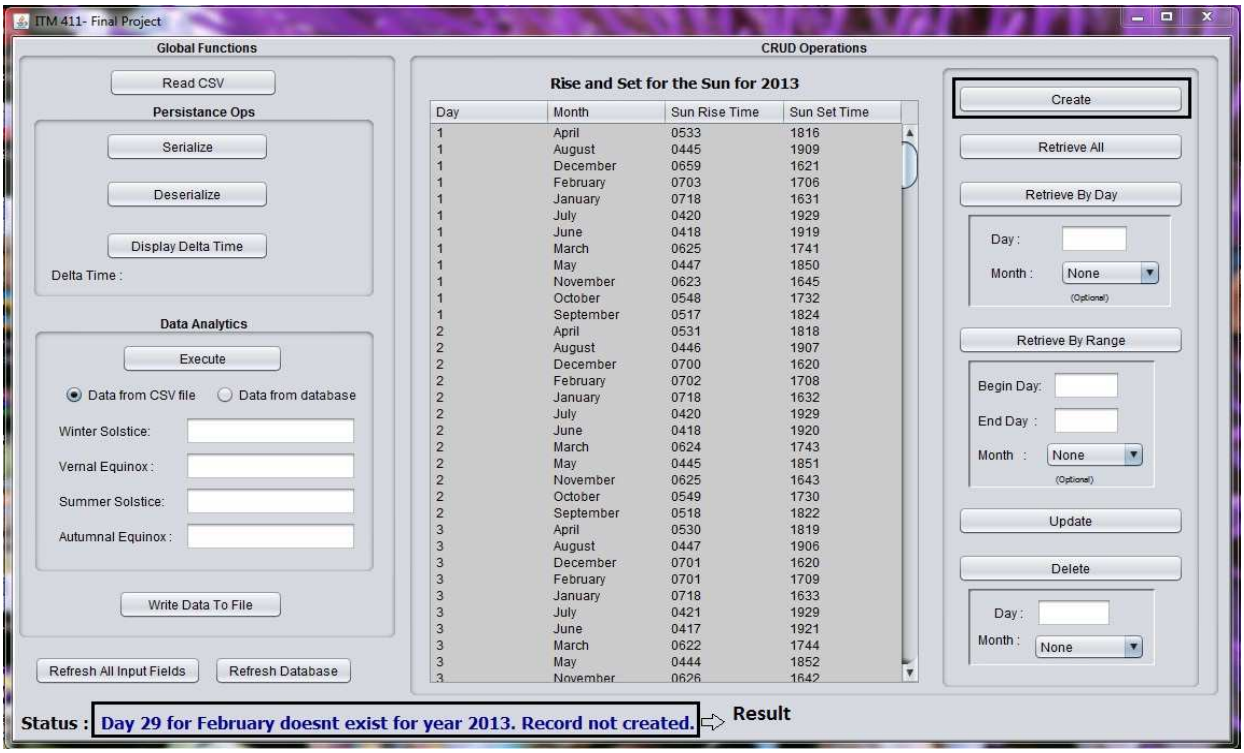
Snapshot 12.21:
Below snapshot shows the result displayed of when user tries to insert record values shown in snapshot 12.20. Since 31st day doesn't exist for April month in 2013 this error result is shown.



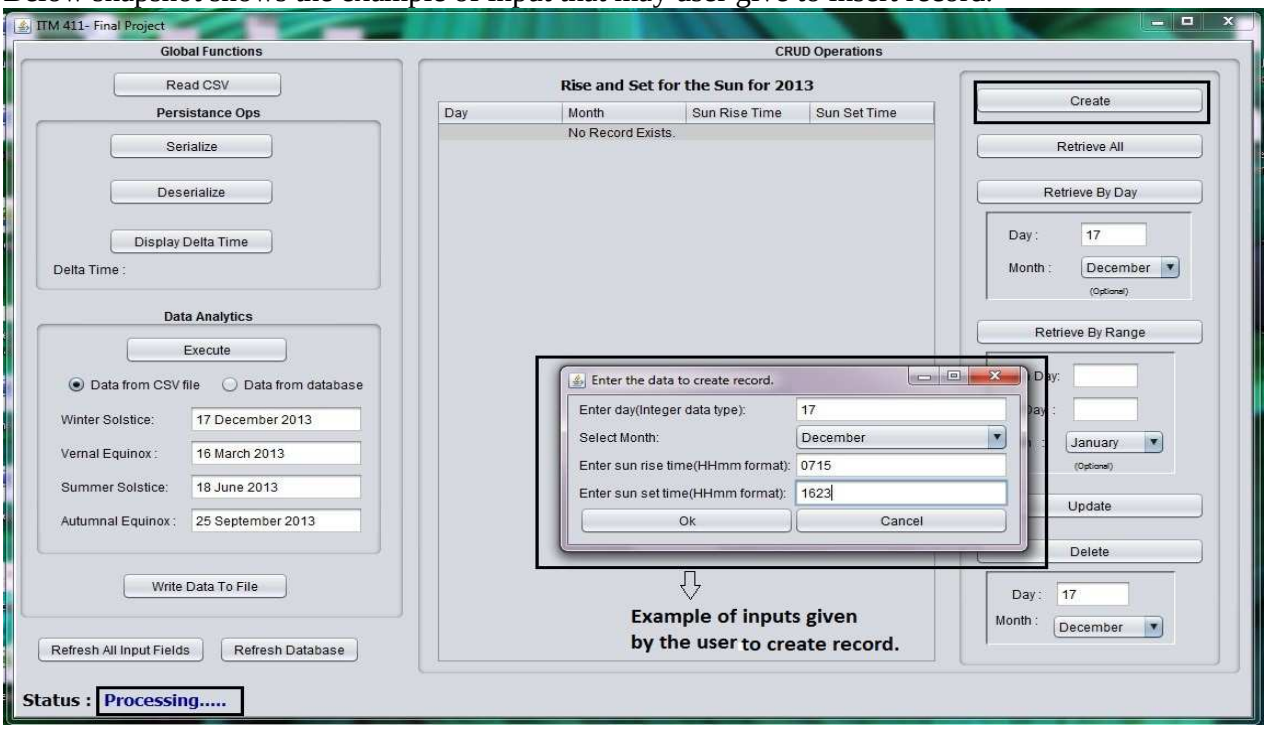
Snapshot 12.22:
Below snapshot shows the example of input that may user give to insert record.



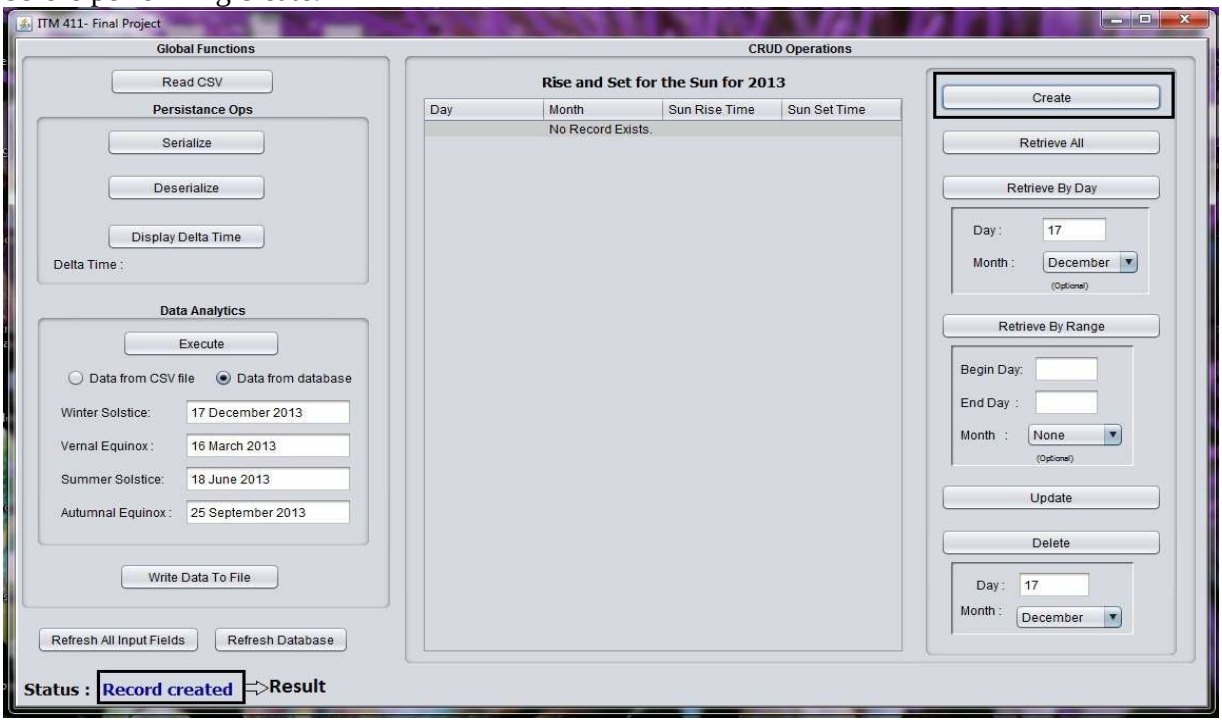
Snapshot 12.23:
Below snapshot shows the result displayed of when user tries to insert record values shown in snapshot 12.22. Since 29th day doesn't exist for February month in 2013 this error result is shown.



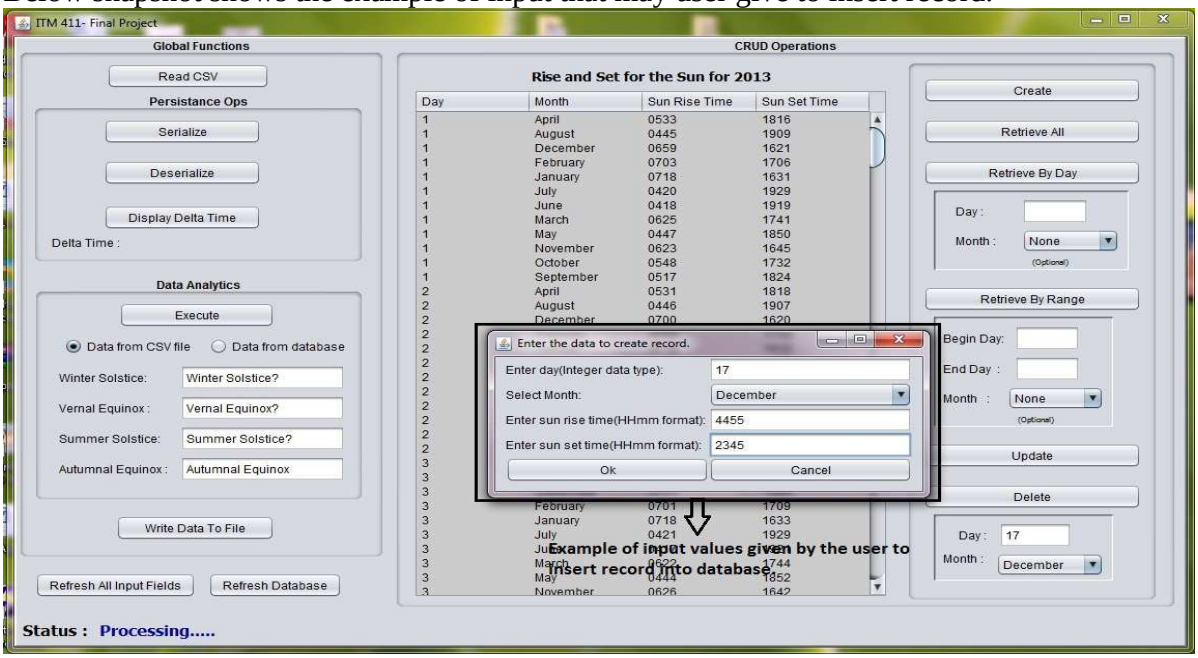
Snapshot 12.24:
Below snapshot shows the example of input that may user give to insert record.



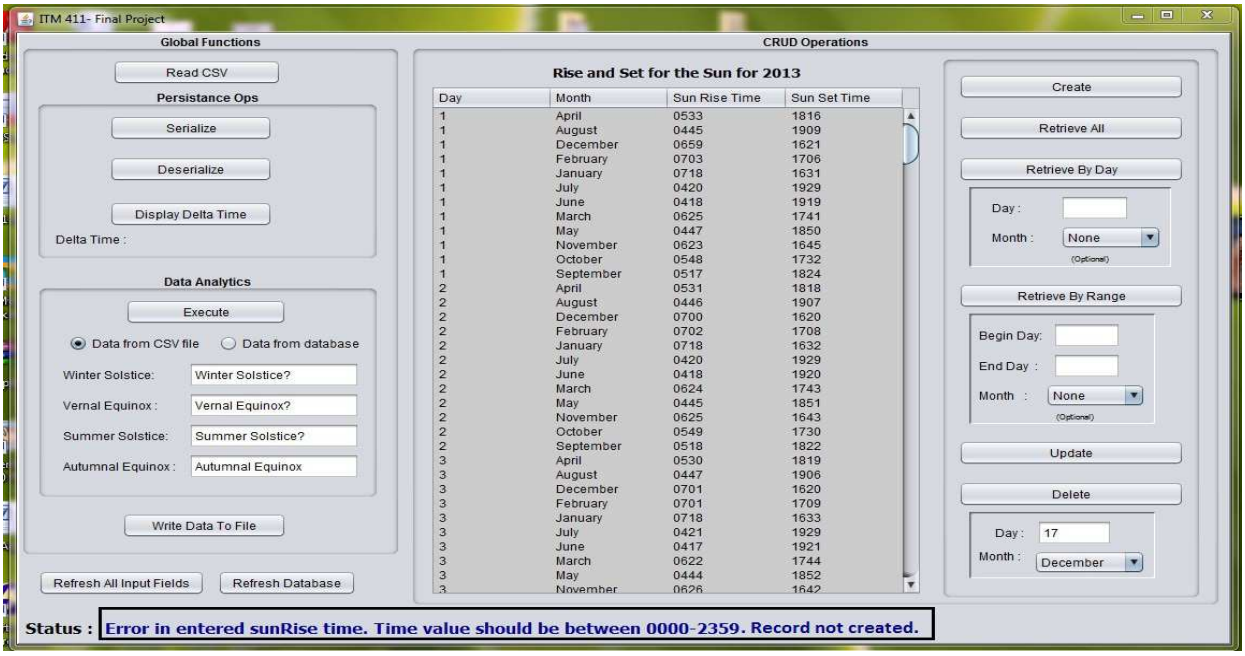
Snapshot 12.25:
Below snapshot shows the result displayed when user tries to insert record values shown in snapshot 12.24 and successfully get inserted since I had deleted 17th December record before performing create.



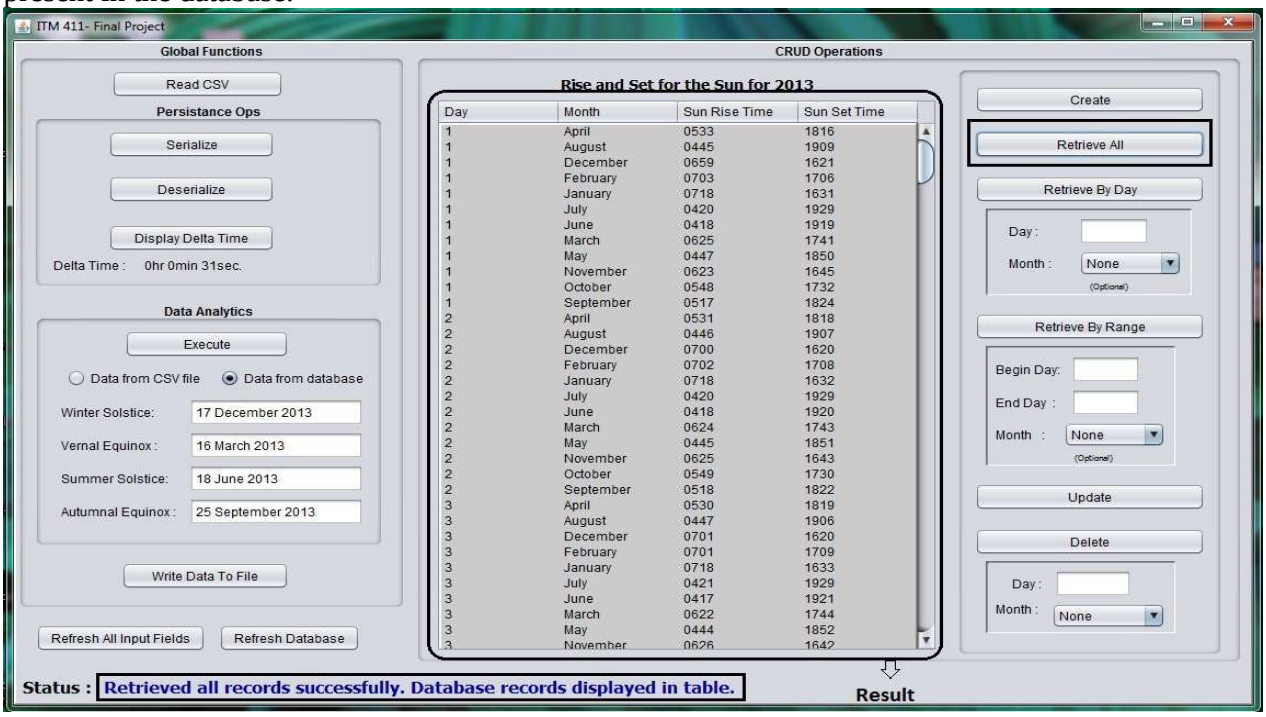
Snapshot 12.26:
Below snapshot shows the example of input that may user give to insert record.



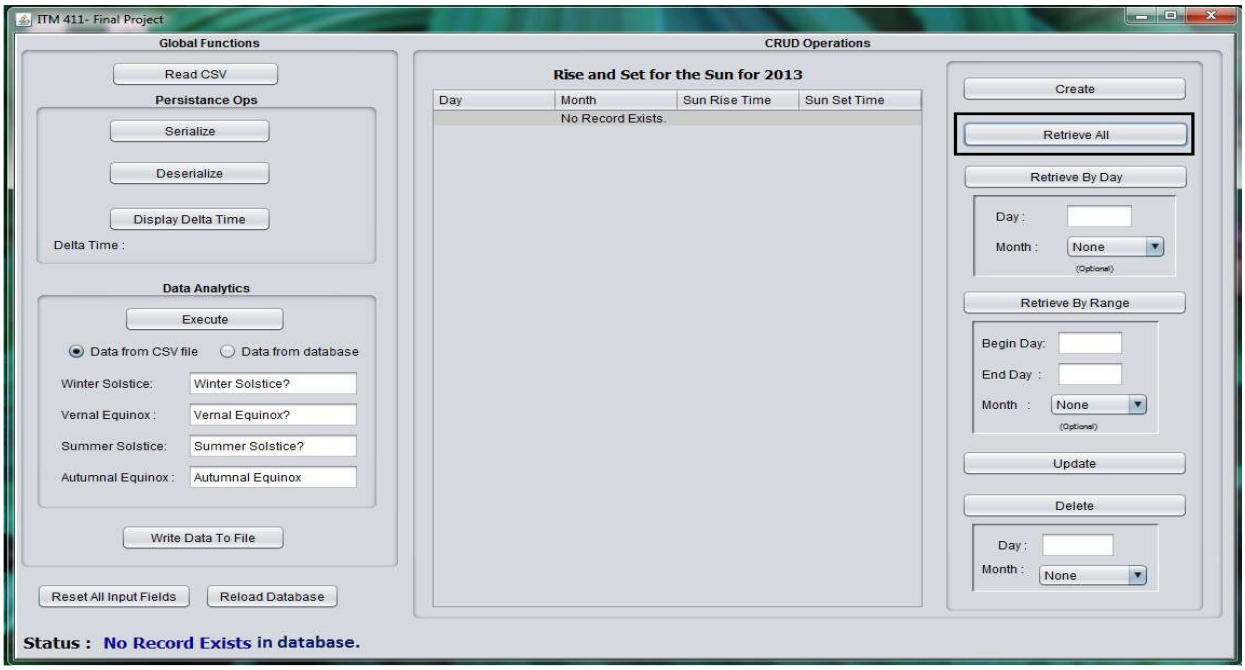
Snapshot 12.27:
Below snapshot shows the result displayed when user tries to insert the values shown in 12.26 as record into database.



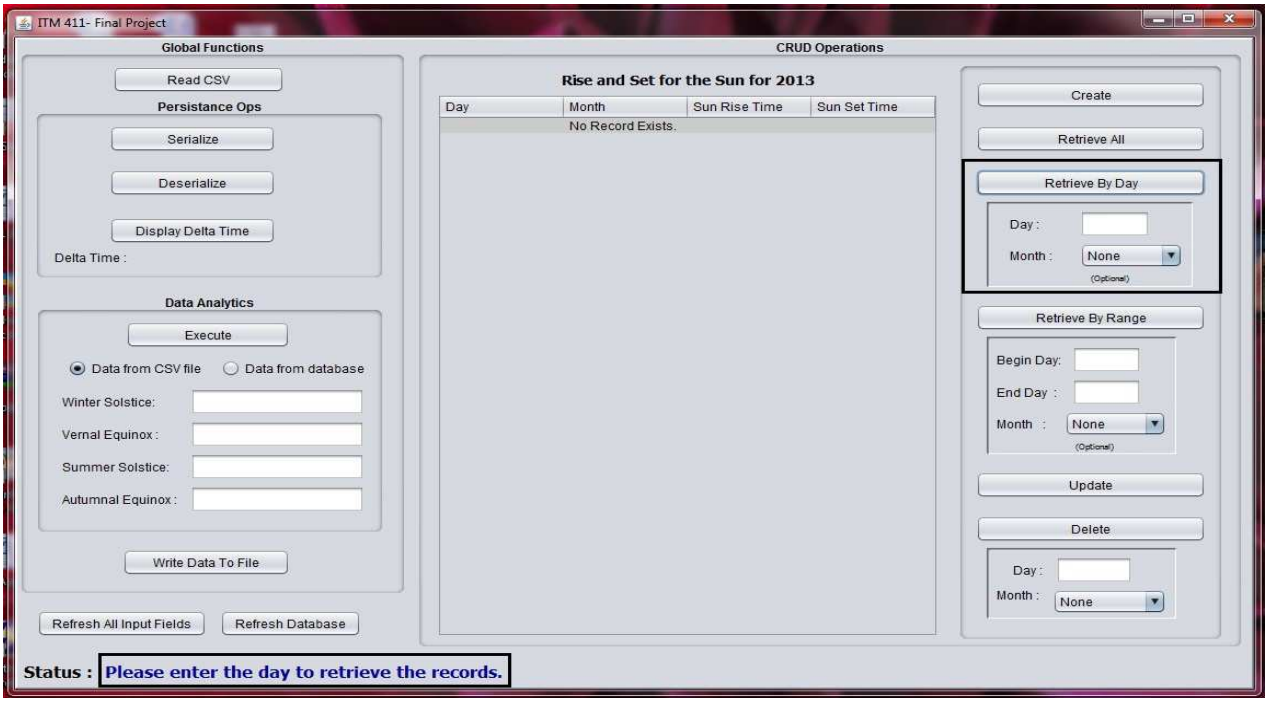
Snapshot 12.28:
Below snapshot shows the result displayed of when user tries retrieve all the records present in the database.



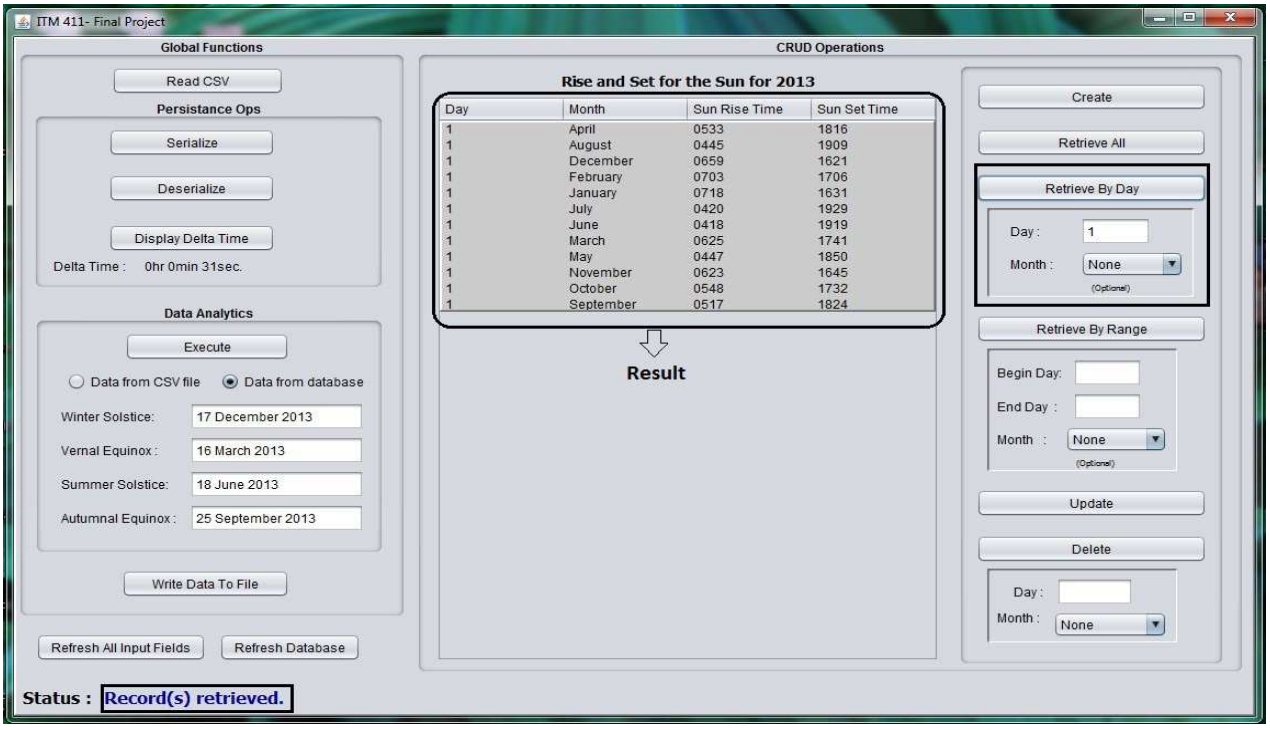
Snapshot 12.29:
Below snapshot shows the result displayed when user tries to retrieve all records from database when no records are present in it.



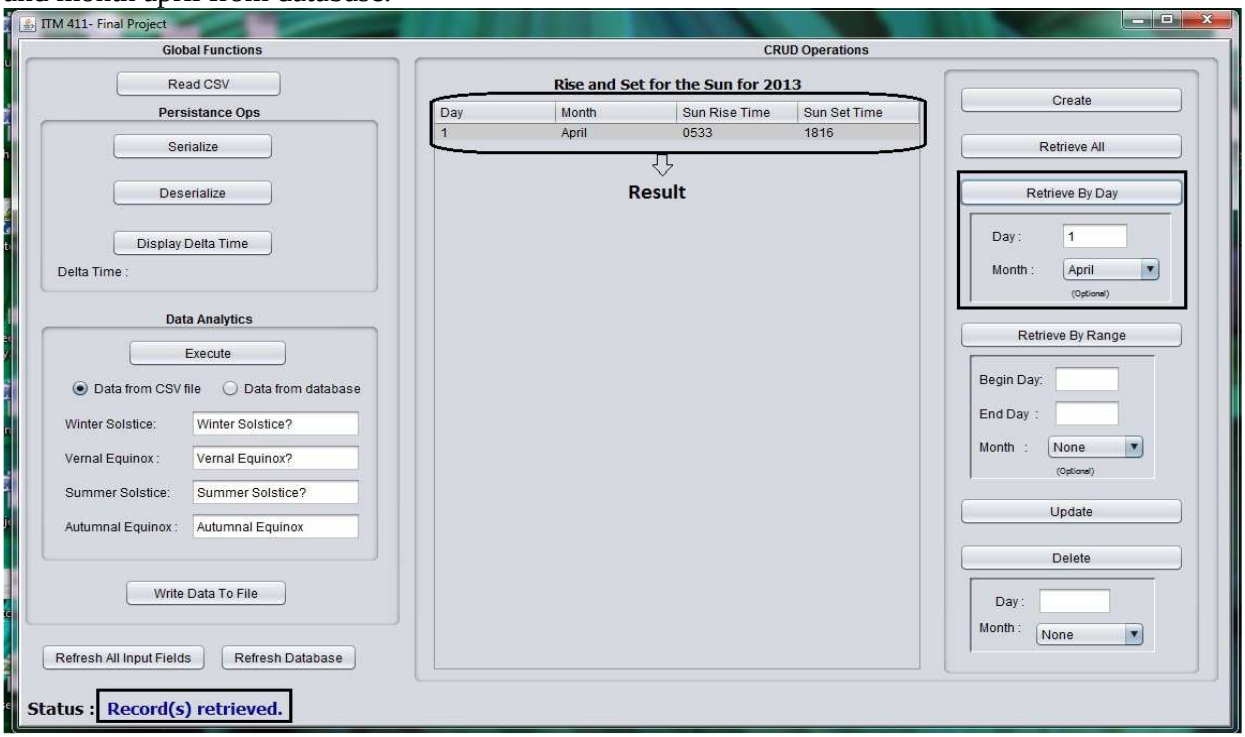
Snapshot 12.30:
Below snapshot shows the result displayed when user tries to retrieve record by day from database.



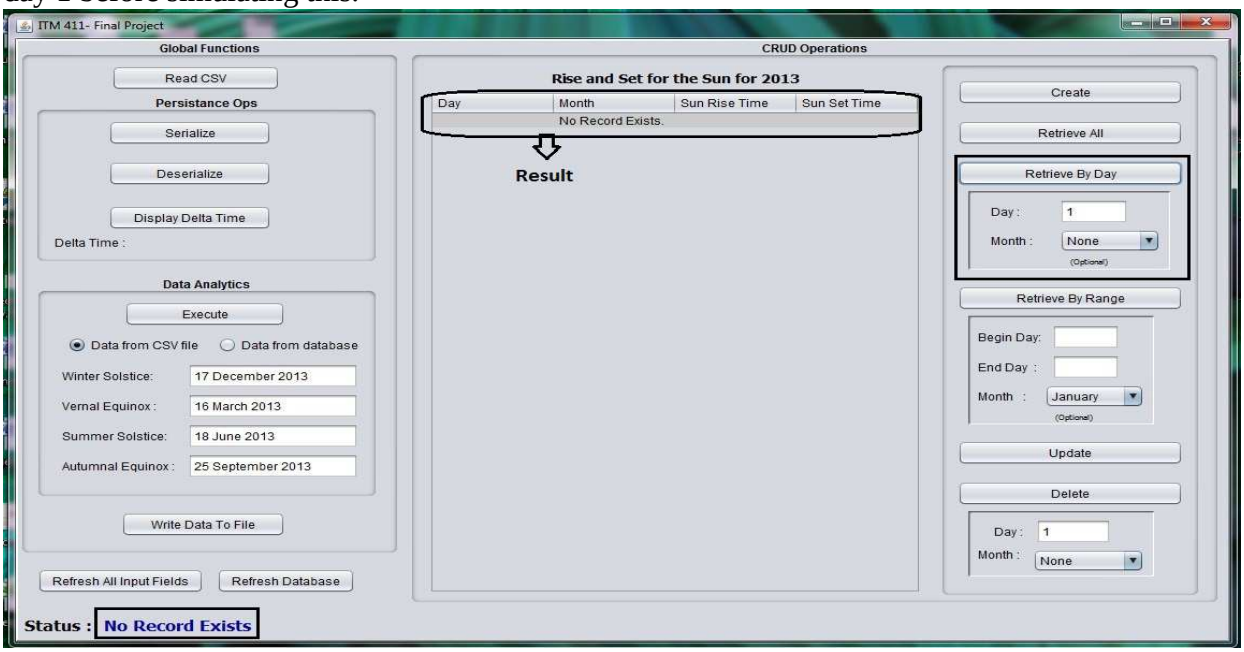
Snapshot 12.31:
Below snapshot shows the result displayed when user tries to retrieve record for day 1 from database.



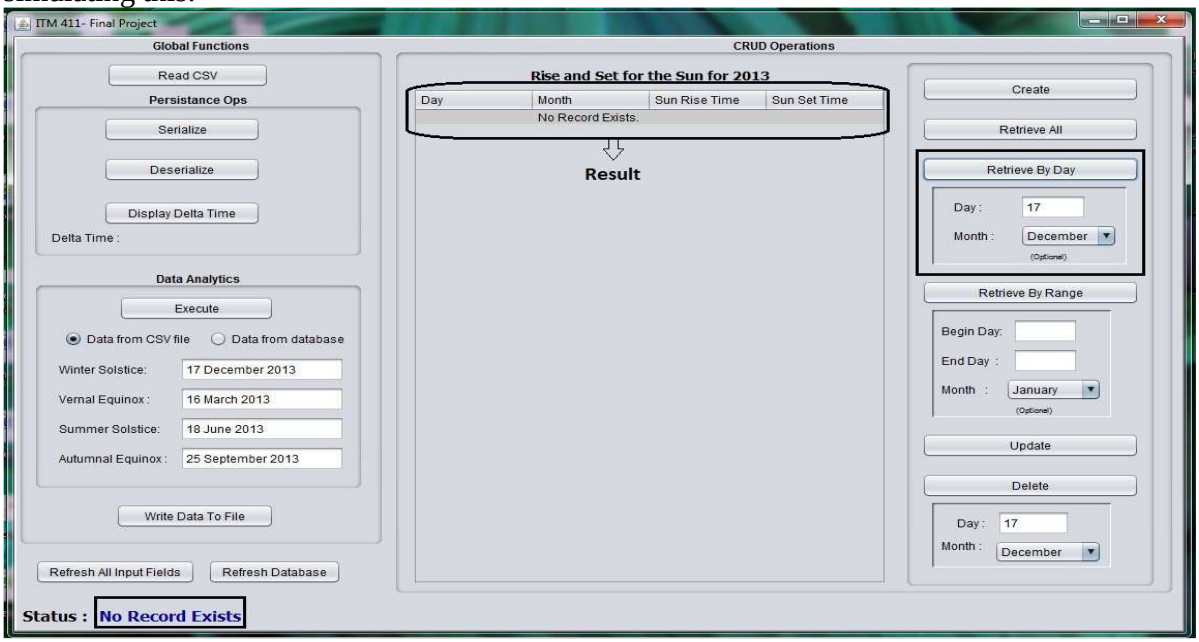
Snapshot 12.32:
Below snapshot shows the result displayed when user tries to retrieve record for day 1 and month april from database.



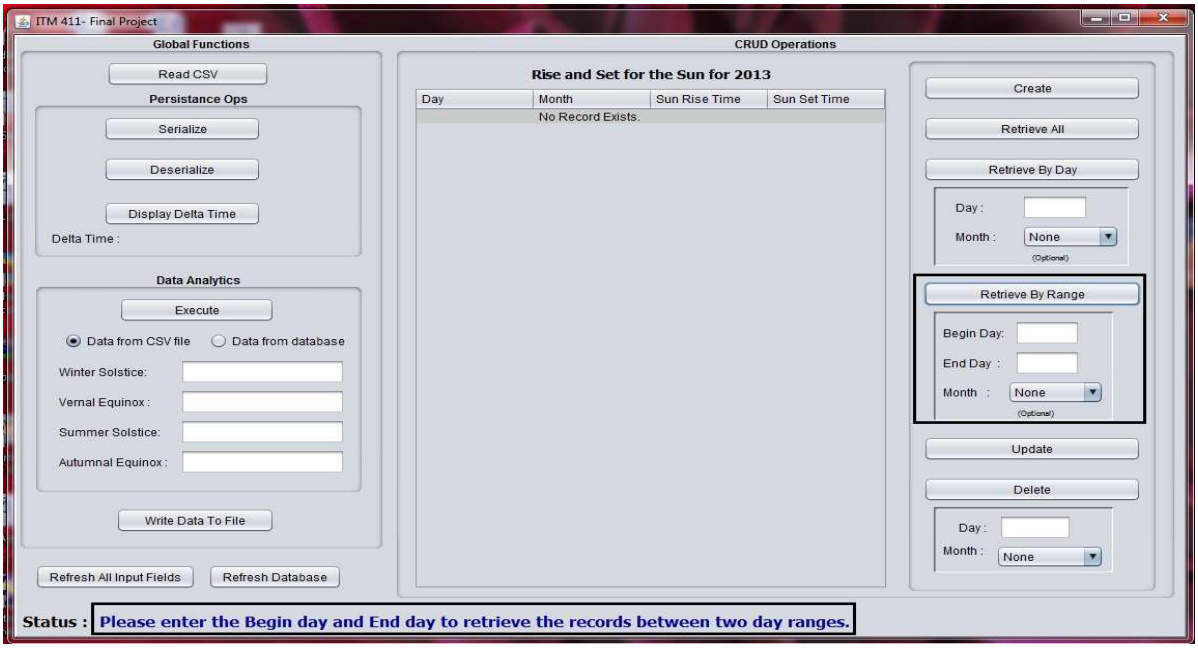
Snapshot 12.33:
Below snapshot shows the result displayed when user tries to retrieve record for day 1 from database. When no record exists for day 1 in database. I had deleted the records of day 1 before simulating this.



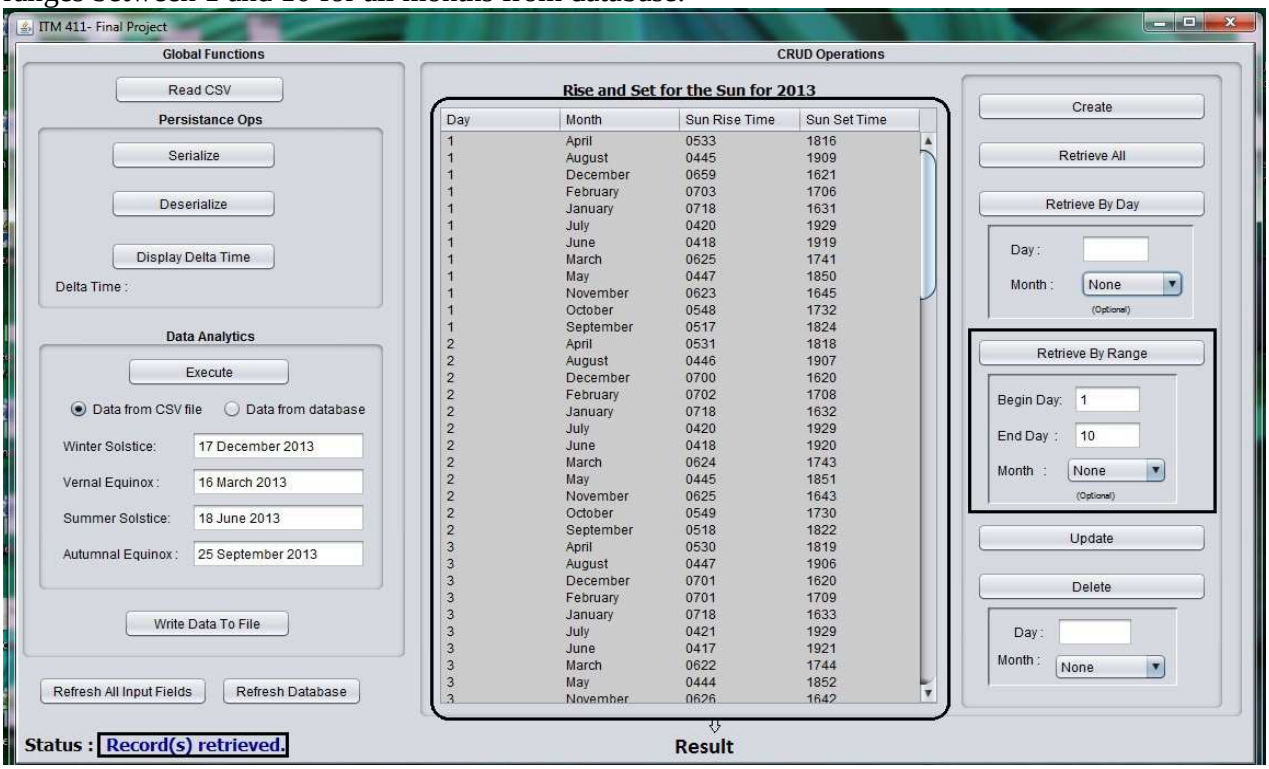
Snapshot 12.34:
Below snapshot shows the result displayed when user tries to retrieve record for day 17 and month December from database. When no record exists for day 17 for moth december in database. I had deleted the record of day 17 for December month before simulating this.



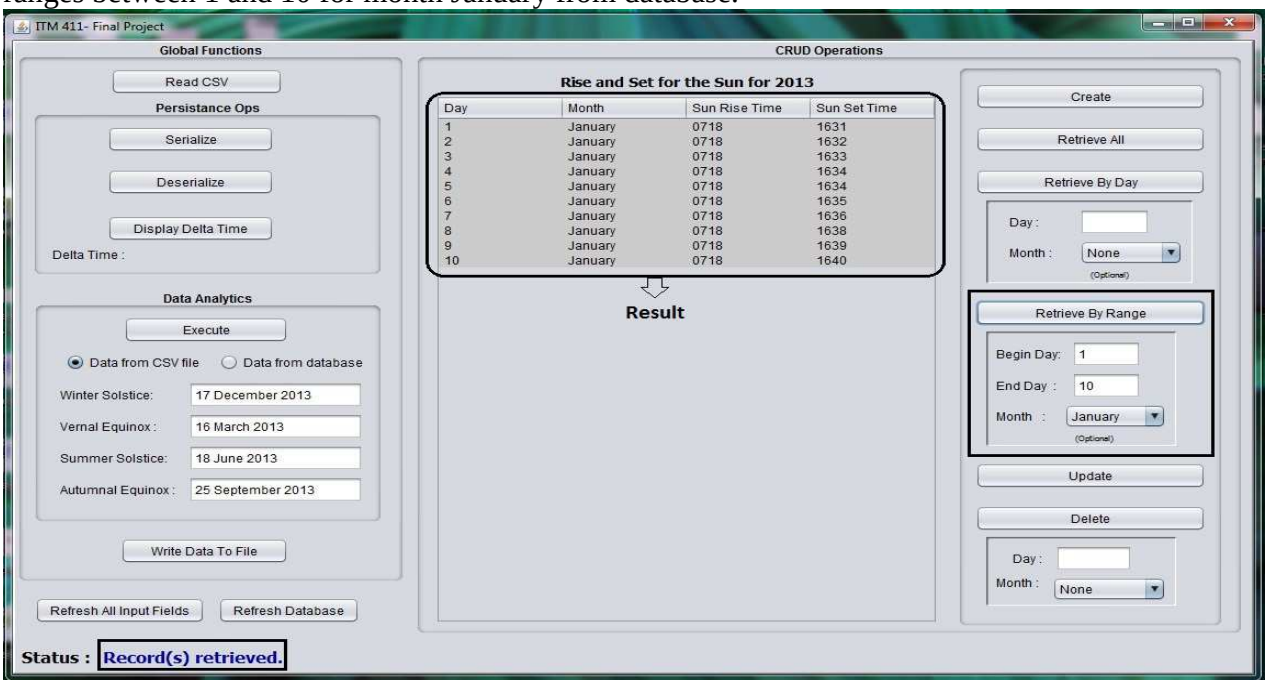
Snapshot 12.35:
Below snapshot shows the result displayed when user tries to retrieve record by range from database.



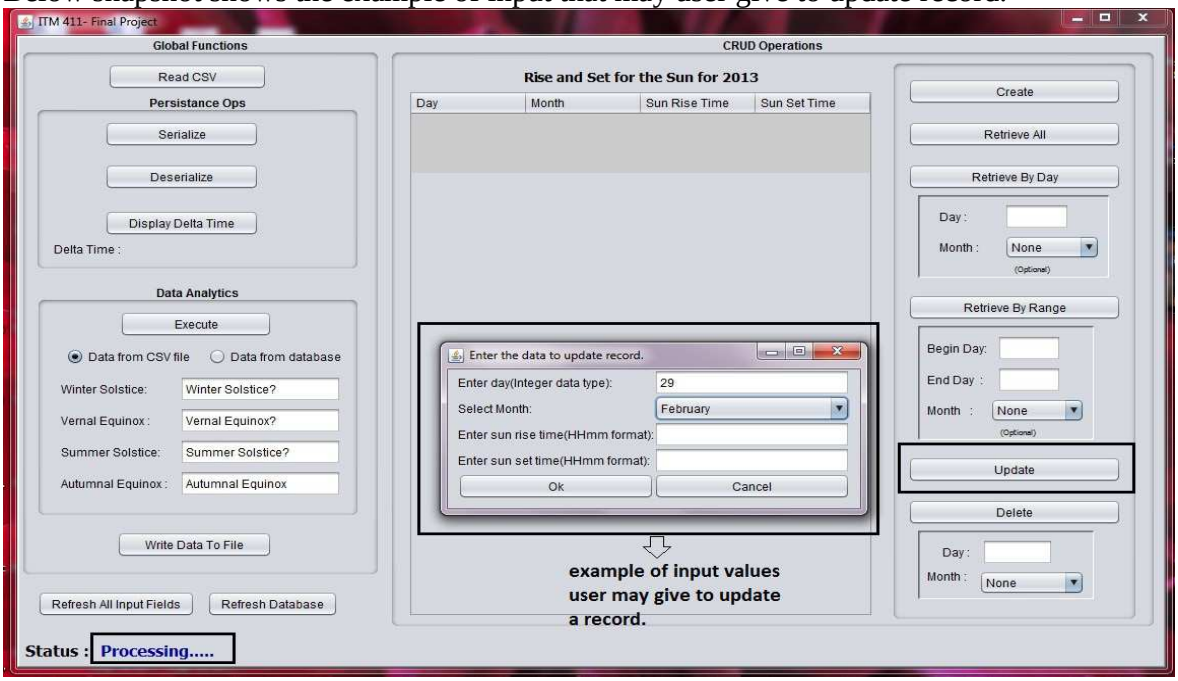
Snapshot 12.36:
Below snapshot shows the result displayed when user tries to retrieve record for day ranges between 1 and 10 for all months from database.



Snapshot 12.37:
Below snapshot shows the result displayed when user tries to retrieve record for day ranges between 1 and 10 for month January from database.

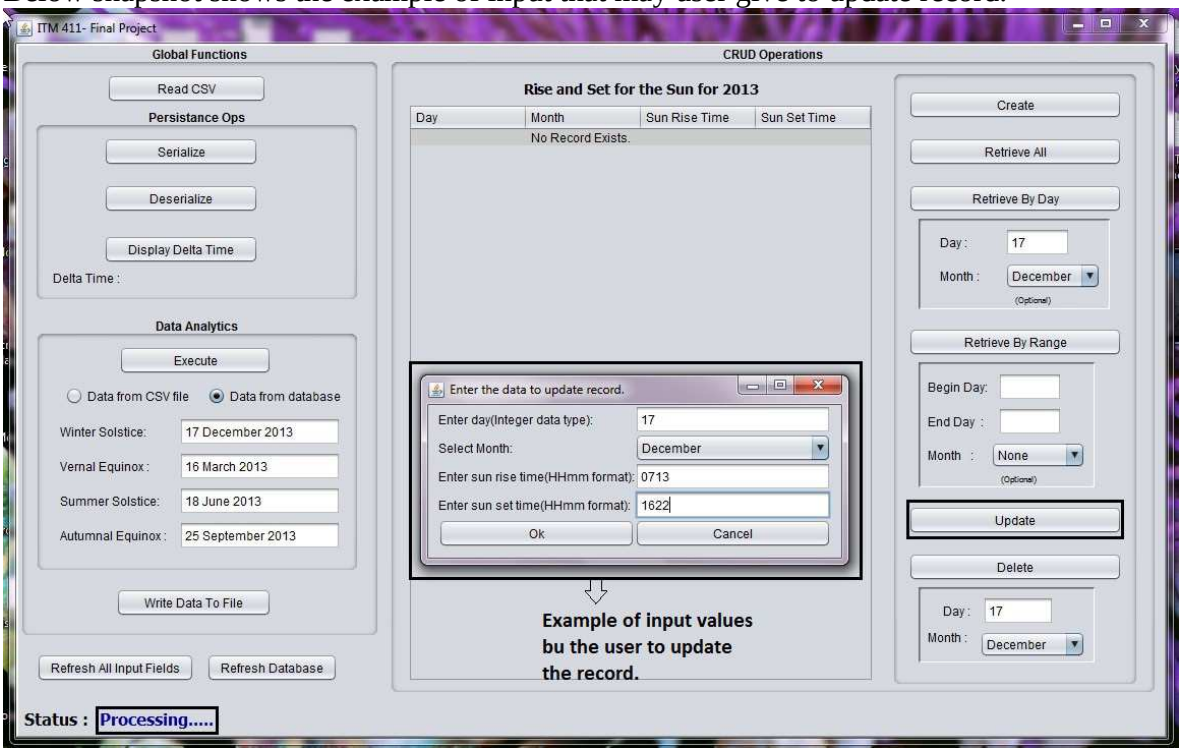


Snapshot 12.38:
Below snapshot shows the example of input that may user give to update record.

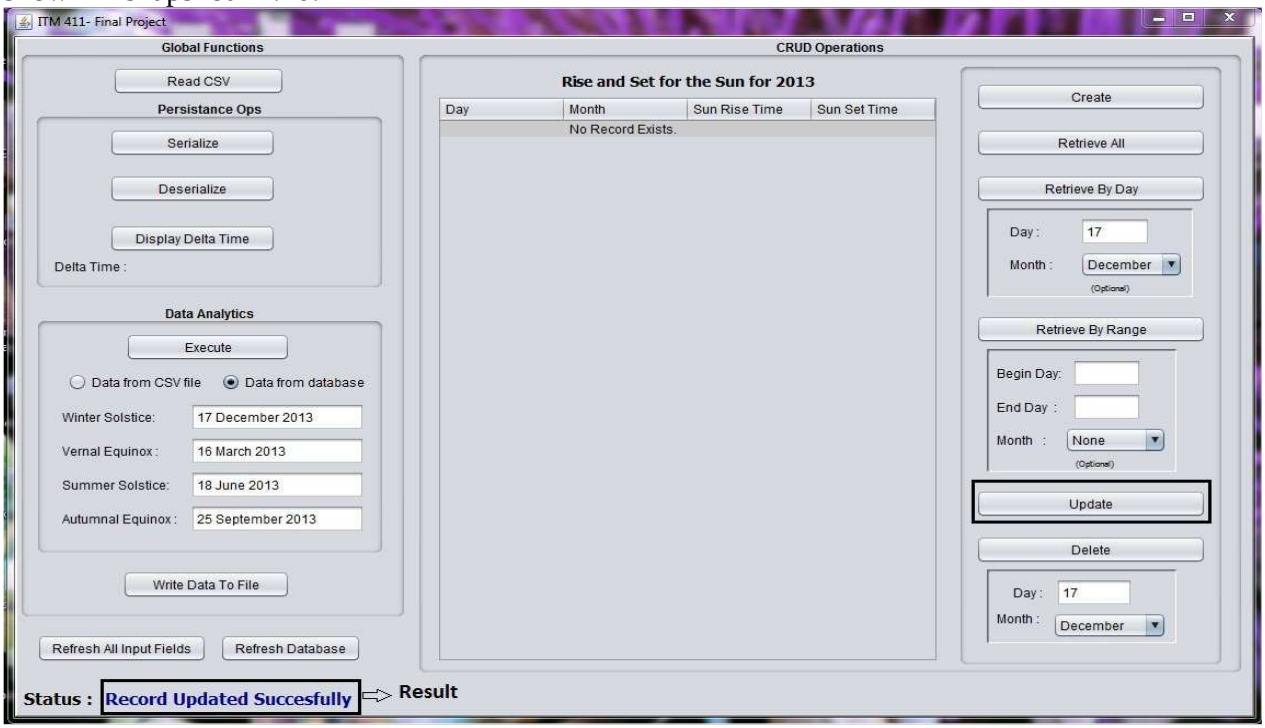


Snapshot 12.39:
Below snapshot shows the result displayed of when user tries to insert record values shown in snapshot 12.38. Since 29th day doesn't exist for February month in 2013 this error result is shown.

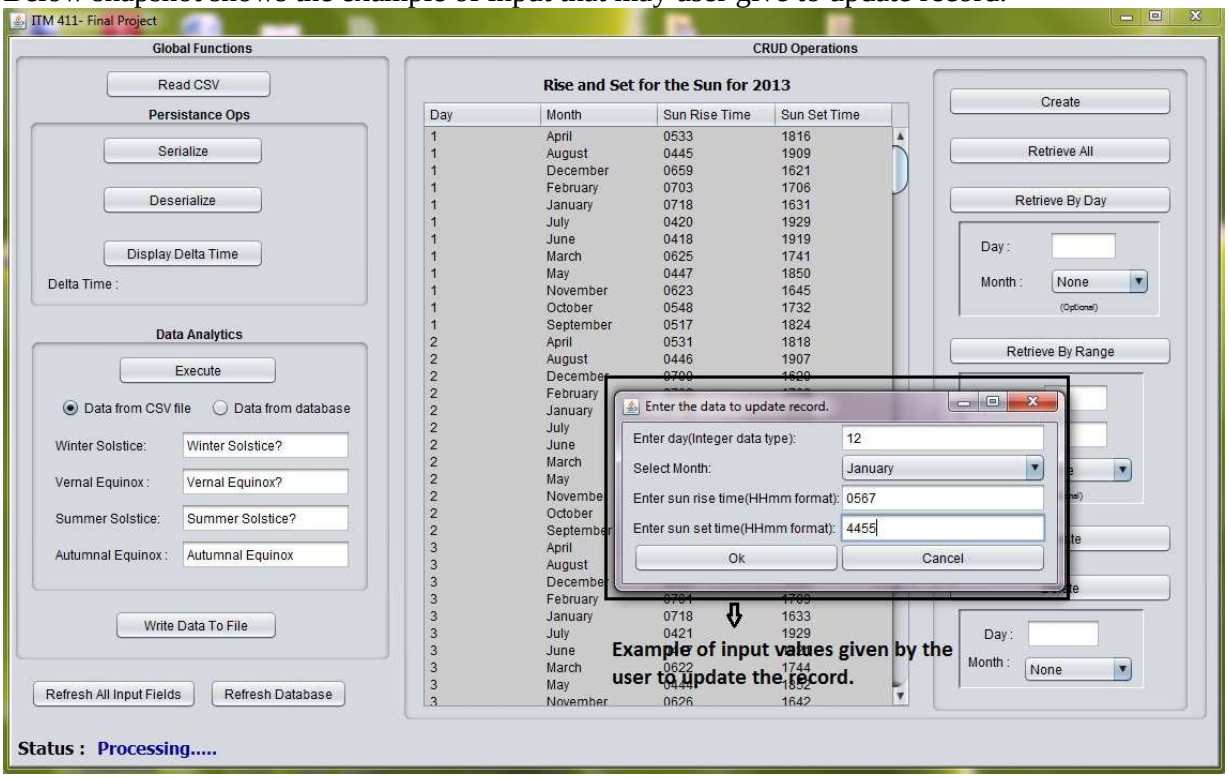
Snapshot 12.40:
Below snapshot shows the example of input that may user give to update record.



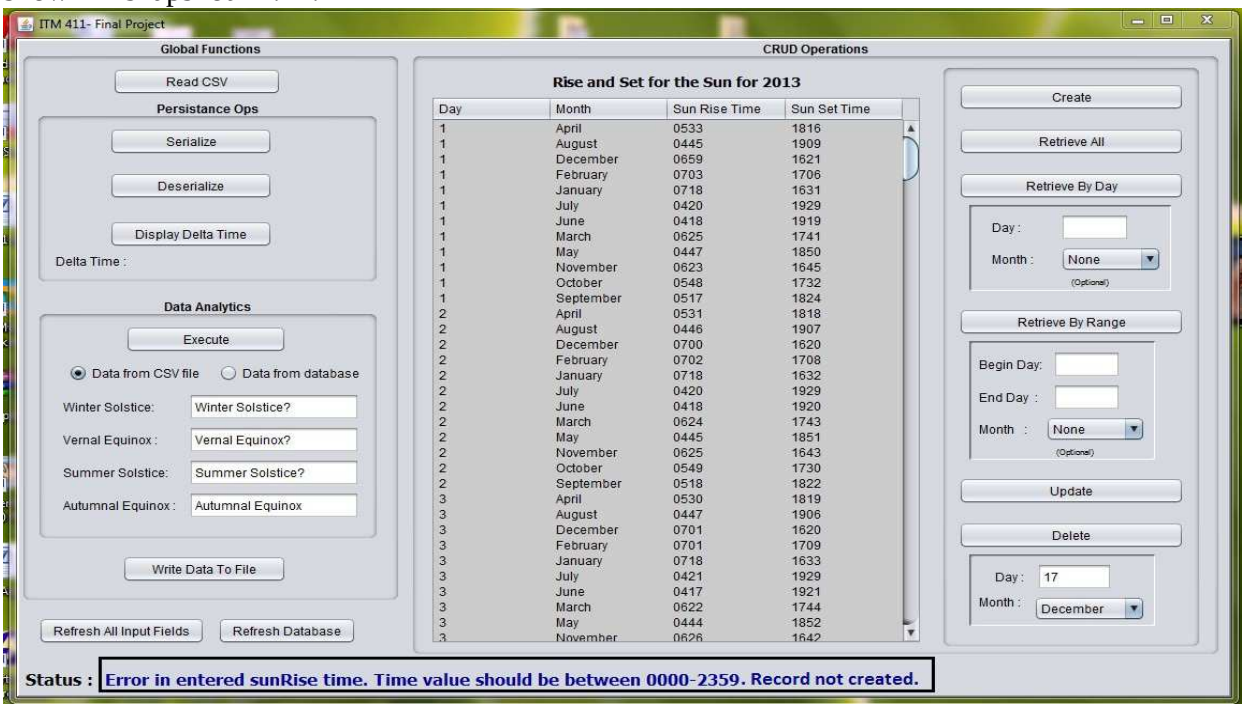
Snapshot 12.41:
Below snapshot shows the result displayed when user the record is updated with values shown in snapshot 12.40.



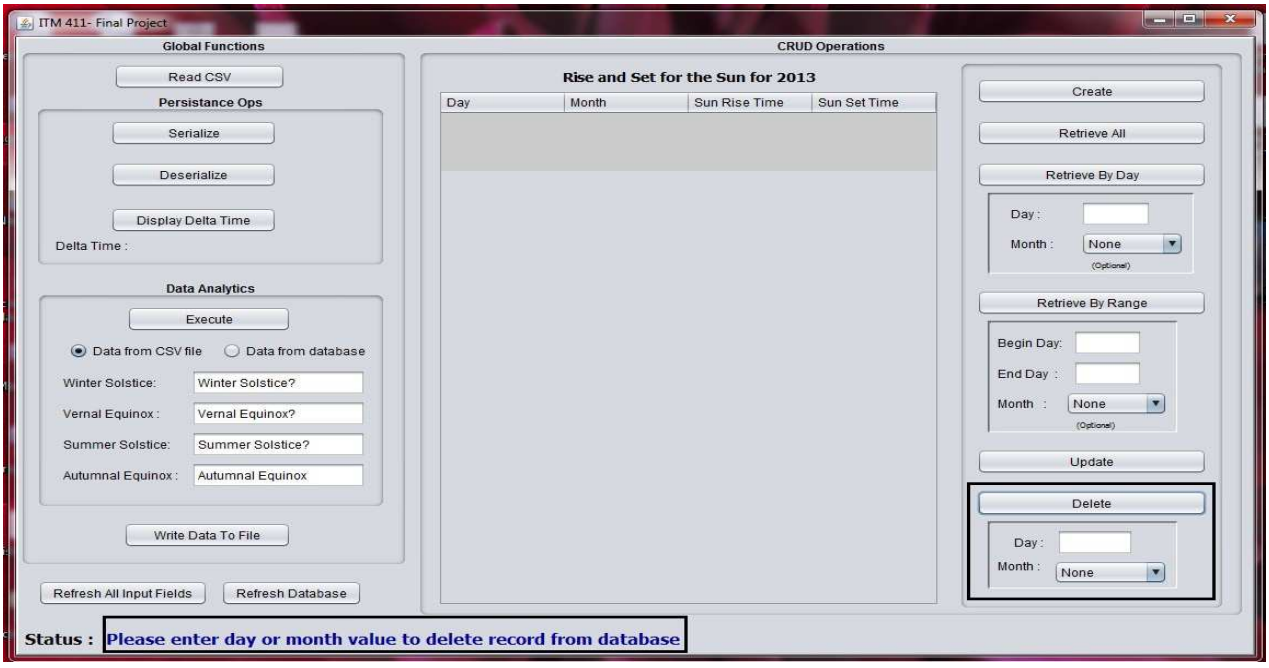
Snapshot 12.42:
Below snapshot shows the example of input that may user give to update record.



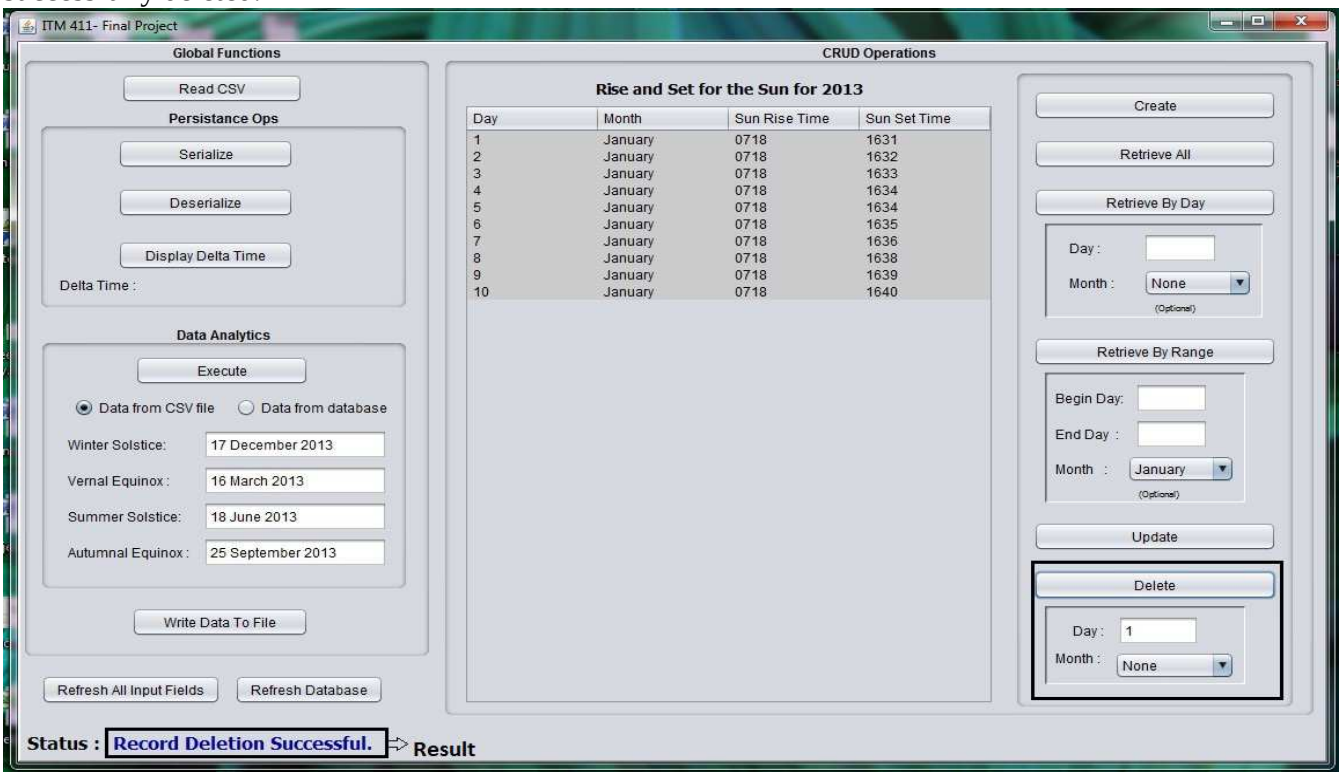
Snapshot 12.43:
Below snapshot shows the result displayed when user tries to update record with values shown in snapshot 12.42.



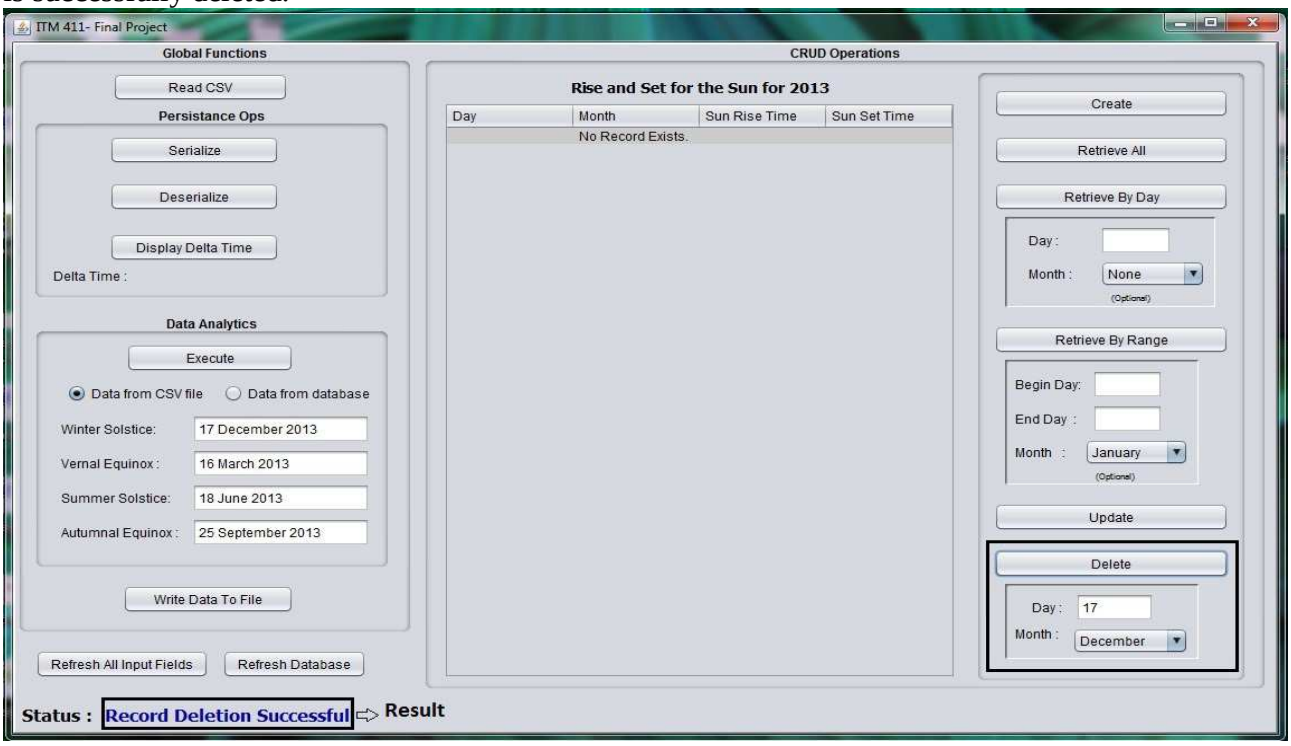
Snapshot 12.44:
Below snapshot shows the result displayed when user tries to delete record from database.



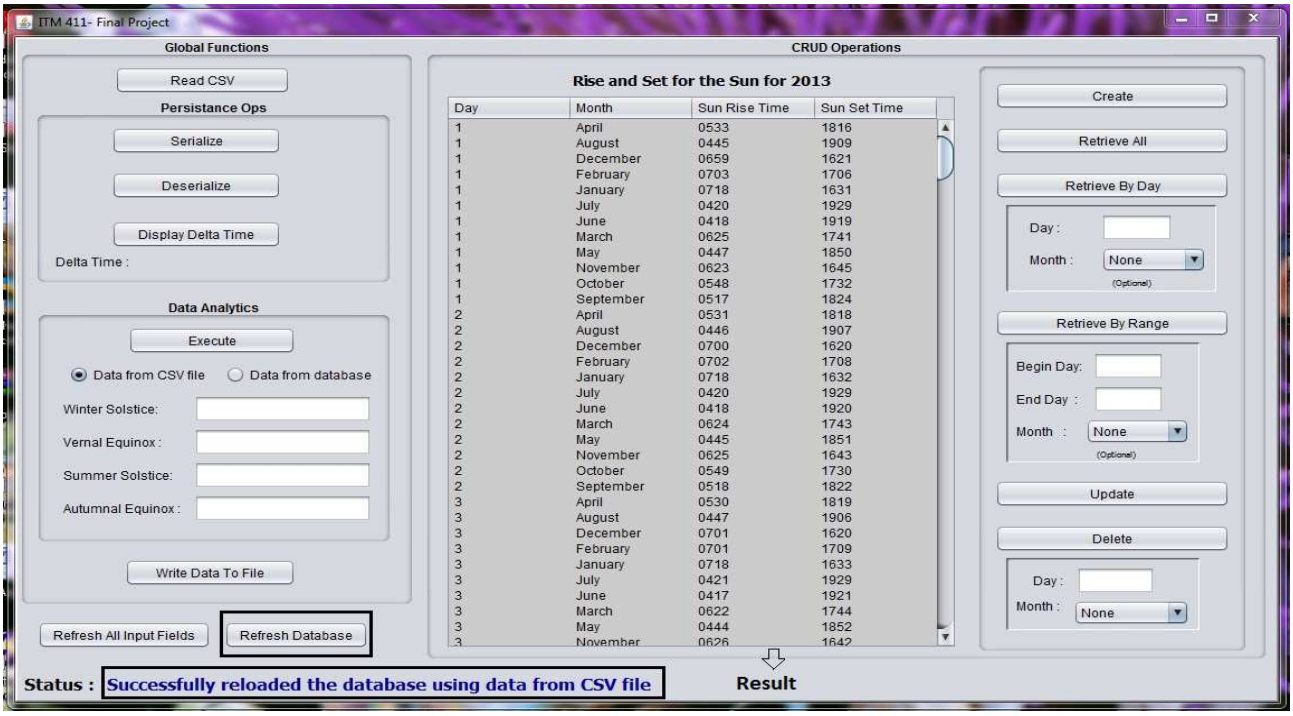
Snapshot 12.45:
Below snapshot shows the result displayed when records for day 1 from all month is successfully deleted.



Snapshot 12.46:
Below snapshot shows the result displayed when records for day 17 for month december is successfully deleted.



Snapshot 12.47:
Below snapshot shows the result displayed when user reloads the database from the data of CSV file.



Snapshot 12.48:
Below snapshot shows the result displayed when user resets all the input text fields.