

## **INFORMATION AND COMMUNICATION TECHNOLOGIES ENGINEERING**

### **GRADUATION PROJECT REPORT**

# **Thesis Title**

## **Administration and Student Affairs System**

### **Supervisors**

Prof. Sameh A.Salem  
Prof. Claudio Fornaro  
Dr. Paolo Prinetto

### **Students**

Ahmed Adel Awad.  
Hany Abd EL-Samad.  
Ahmed Hisham Saad.  
Amro Salem Hassan.  
Ramy Saad Mohareb.

Academic year 2010/11

## Administration and Student Affairs System

### Project Team:

#### 1- Amro Salem Hassan:

- Database Design and Security.
- Student Module.
- Shared in Admin Module.

#### 2- Ramy Saad Mohareb:

- Constructing ER Model Using SQL Developer Datamodeler (Oracle 2010 release).
- Professor Module.
- Shared in Admin Module.

#### 3- Hany Abd El-Samad:

- Database Creation Scripts.

#### 4- Ahmed Adel Awad:

- Control Module.

#### 5- Ahmed Hisham Saad:

- Graduate Module.

# Index

## **Part I : Modeling the Problem.**

<b>Chapter 1 : <u>Introduction:</u></b>	<b>Page</b>
1.1- Introducing the Problem.	4
1.2- Graphical Representation of the Project.	5
1.3- Deeper Look into the Database.	8
1.4- Modules and Layers of the Client Application.	10
1.5- Products Used in the Project.	16

## **Part II : Solving the Problem.**

<b>Chapter 2: <u>Amro Salem Contribution:</u></b>	
2.1- Designing the Implementation.	19
2.2- Identifying the Users of the System and Creating Corresponding Roles.	24
2.3- Some Functions of the Student Module.	26
2.4- Important Features of the Student Module Java Code.	29
2.5- Some Functions of the Admin Module.	33
<b>Chapter 3: <u>Ramy Saad Contribution:</u></b>	
3.1- Some Functions of the Professor Module.	37
3.2- Important Features of the Professor Module Java Code.	42
3.3- Important Features of the Admin Module.	49
3.4- The Java Mail API.	52
<b>Chapter 4: <u>Hany AbdEISamad Contribution:</u></b>	
4.1- Creating Tables.	54
4.2- Enable Row Movement Feature.	57
<b>Chapter 5: <u>Ahmed Awad Contribution.</u></b>	58
<b>Chapter 6: <u>Ahmed Hisham Contribution.</u></b>	83

# **Chapter 1**

## **Introduction**

### **1.1- File VS Computerized System:**

Due to the increasing number of students applying to Helwan/Uninnetuno Engineering Degree, it became almost impossible to separately manage the requests of those students. There are several solutions for this problem. One of the most competing solutions is implementing a three-tier system (Data server - Application server - Clients). This solution requires a lot of resources: at least three up-to-date server machines, at least two dedicated data servers, and a cooling system. Since those requirements were difficult to allocate, and since there were a lack in resources, another solution was considered.

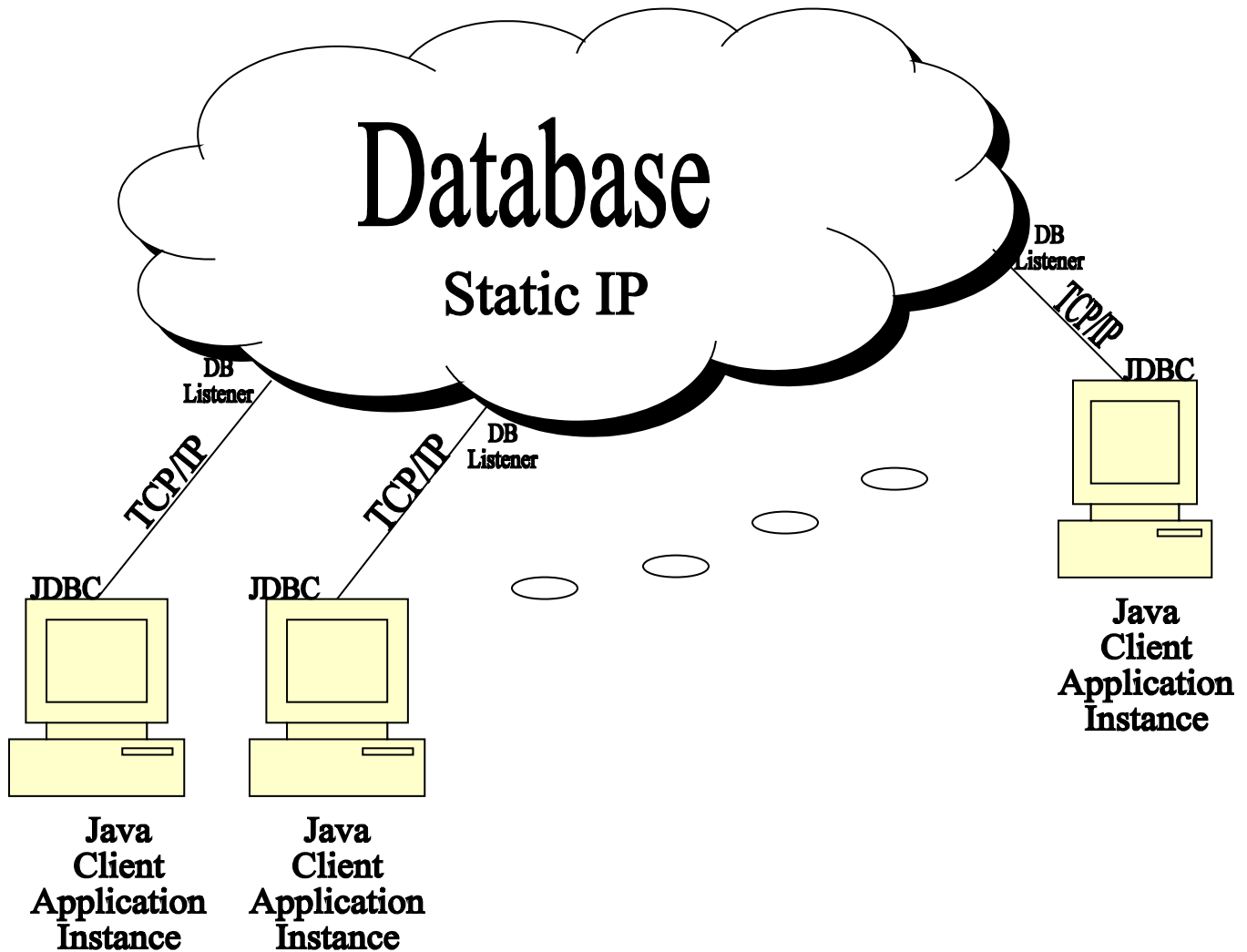
The solution we chose to implement is a two-tier system which consists of a dedicated database, and a specially constructed Java Client Application. The upside of this solution is that the processing is no more centralized. On the contrary, the client application consumes the resources of the user running it locally. The only thing that is centralized is the database. The trade off was harder programming to manage parallel execution problems, and resources (processor and memory) consumption.

We managed to manipulate the database through Java Client Application by using JDBC API offered by Oracle for connecting and retrieving data from data sources.

The resource consumption was minimized by utilizing parallel processing through the Threading API of the Java platform. Also the event driven nature of the desktop application made minimizing the resource consumption easier, since only the process that the user initiates will be using the resources.

The choice of Java Platform (Java programming language and APIs) was made based on supporting a solution for the problem of diversity of platforms. Since the two-tier architecture requires installing the client application on the users PCs (which may have different platforms), the portability of the system has been a major issue. The Java Platform solves this problem by providing different JVMs ( Java Virtual Machines ) for different architectures. So the Java bytecode resulting from compiling the Java source codes could run on any platform regardless how it is implemented. So as a conclusion, the Java Platform was chosen to overcome the interoperability issue resulting from the different platforms that would be running the client application.

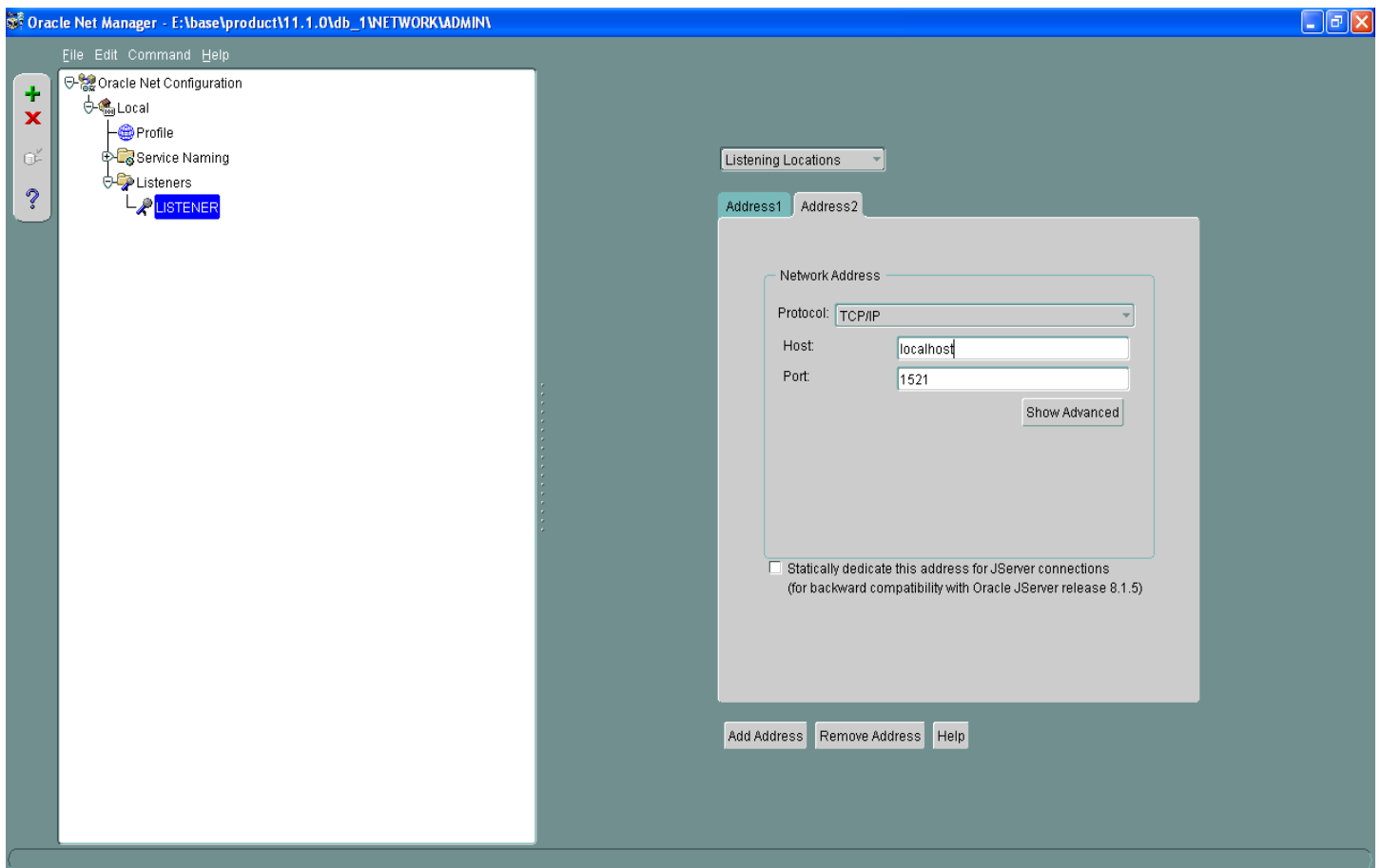
1.2- A Graphical Representation of the Project:



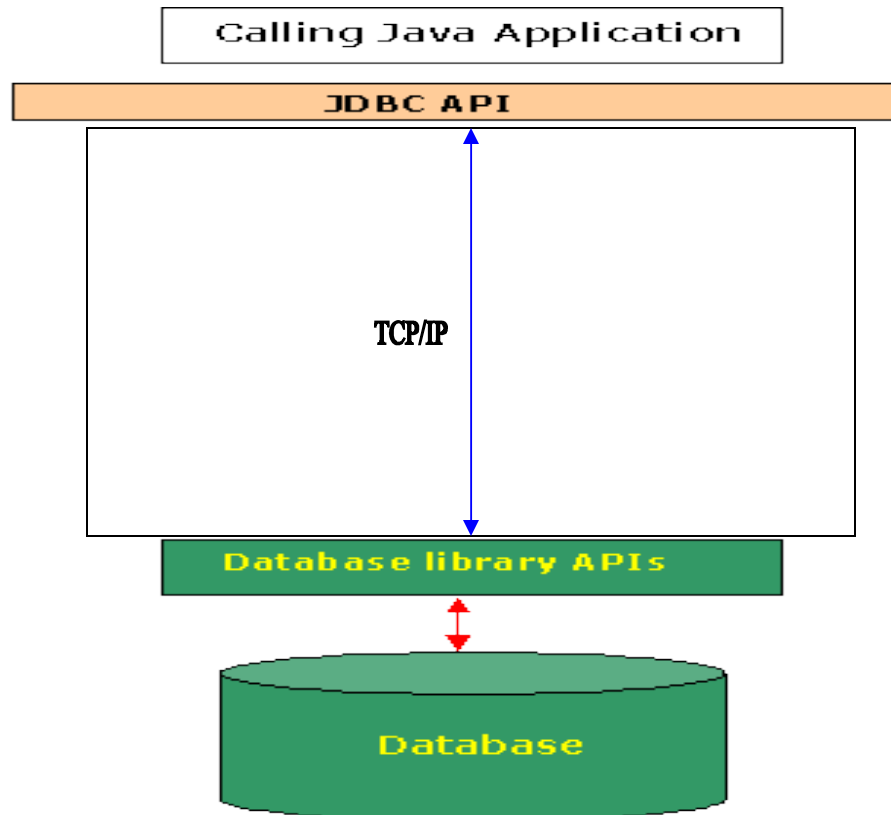
The static IP is a fixed IP address given to a cloud terminal. If the static IP was not used in the system it would have been impossible for the client application to recognize the server over the cloud. Any terminal accessing the cloud is normally assigned a dynamic IP that changes periodically or whenever the connection is terminated. With this situation the possible solutions are either to construct a virtual private network (VPN) between the clients and the database, or to assign a fixed IP to the database so that whenever a client is calling the database, this IP address could be used to establish the connection. The problem with the VPN was that whenever a new client is introduced to the system, the

network administrator must add this client to the VPN manually using the MAC address. So it is obvious that the dynamicity of the system would have been unaccomplished if this solution was implemented. On the other hand, the solution of the static IP put no overhead for establishing a fixed closed network. Besides the static IP solution is by far more compatible with configuring the database listener. The disadvantage of the static IP was the vulnerability for unauthorized access. Using the Oracle database helped us overcoming this disadvantage on the server side, since the Oracle database utilizes multiple security features. Using the Java Platform helped us overcoming this problem on the client side, since the Java Platforms implements the Sandbox Security Model, which requires authentication whenever a connection is requested from a cloud.

Database Listener is an application offered by Oracle and bundled with their database server. The listener continuously checks for connections coming over a TCP/IP port. Oracle's Net Manager is used to configure the listener.



There were multiple protocols that could be used to achieve successful communication over the cloud: UDP, TCP, and OracleNet. The TCP was chosen specifically because by far the TCP is the most compatible communication protocol used in computer networks. Additionally, the TCP is perfectly compatible with both the database listener and the JDBC API. Dividing the data to be communicated into small segments called packets, the TCP creates messages to be communicated over the cloud.



JDBC is an API provided by Oracle to achieve the communication link between client applications and the database. The JDBC provides an interface between the client application and one end of the TCP channels, so basically translates Java strings into SQL queries that could be understood by the data-server and translates back the result returned by the database into ResultSet objects that could be understood by JRE.

### 1.3- Deeper Look into the Database:

A large organization such as a university contains a large number of entities. Each entity is correlated to the other entities by a relationship. Relationships between entities defined the way each entity will affect and be affected by the other entities. When constructing the database the designer is supposed to figure out a mapping between the entities existing in the environment to be modeled and the tables of the database modeling this environment. Such mapping should represent each one of those entities into a table, the columns of this table carries the attributes of the entity. As an example when modeling the employees of an organization a table would be created with its columns carrying the attributes that describe each of the employees. So the table would look like:

Column Name	Data Type	Nullable	Data Default	COLUMN ID	Primary Key	COMMENTS
EMPLOYEE_ID	VARCHAR2(25 BYTE)	No	(null)	21	1 (null)	
FIRST_NAME	VARCHAR2(27 BYTE)	Yes	(null)	1	(null) (null)	
MIDDLE_NAME	VARCHAR2(27 BYTE)	Yes	(null)	2	(null) (null)	
LAST_NAME	VARCHAR2(27 BYTE)	No	(null)	3	(null) (null)	
MANAGER_ID	VARCHAR2(25 BYTE)	Yes	(null)	4	(null) (null)	
NATIONALITY	VARCHAR2(20 BYTE)	Yes	(null)	5	(null) (null)	
EMAIL	VARCHAR2(35 BYTE)	No	(null)	6	(null) (null)	
DATE_OF_BIRTH	DATE	Yes	(null)	7	(null) (null)	
PLACE_OF_BIRTH	VARCHAR2(27 BYTE)	Yes	(null)	8	(null) (null)	
STREET	VARCHAR2(200 BY...	Yes	(null)	9	(null) (null)	
CITY	VARCHAR2(20 BYTE)	Yes	(null)	10	(null) (null)	
STATE	VARCHAR2(20 BYTE)	Yes	(null)	11	(null) (null)	
COUNTRY	VARCHAR2(20 BYTE)	Yes	(null)	12	(null) (null)	
HIRE_DATE	DATE	No	(null)	13	(null) (null)	
JOB_TITLE	VARCHAR2(20 BYTE)	No	(null)	14	(null) (null)	
COURSE_EXPERTISE1	VARCHAR2(25 BYTE)	Yes	(null)	15	(null) (null)	
COURSE_EXPERTISE2	VARCHAR2(25 BYTE)	Yes	(null)	16	(null) (null)	
COURSE_EXPERTISE3	VARCHAR2(25 BYTE)	Yes	(null)	17	(null) (null)	
COURSE_EXPERTISE4	VARCHAR2(25 BYTE)	Yes	(null)	18	(null) (null)	
COURSE_EXPERTISE5	VARCHAR2(25 BYTE)	Yes	(null)	19	(null) (null)	
BACKGROUND	VARCHAR2(100 BY...	Yes	(null)	20	(null) (null)	
DEPARTMENT_ID	VARCHAR2(25 BYTE)	Yes	(null)	22	(null) (null)	
USERNAME	VARCHAR2(25 BYTE)	Yes	(null)	23	(null) (null)	
TELEPHONE_NO	VARCHAR2(11 BYTE)	Yes	(null)	24	(null) (null)	
MOBILE_NO	VARCHAR2(11 BYTE)	Yes	(null)	25	(null) (null)	
EMPLOYEE_TYPE	VARCHAR2(25 BYTE)	Yes	(null)	26	(null) (null)	



Each one of these attributes is bounded by natural conditions, for example a hundred years old employee cannot be hired. These natural boundaries are modeled in the database through constraints. Constraints are also used to enforce the business process rules. For example in the case of a university a student cannot enroll in a course which he hasn't passed its prerequisites. The types of constraints available in Oracle database are:

*1- Primary Key constraints:* The primary key is used to identify any row in the system. Primary key enforces the no two rows can have the same value for a column also rows cannot have null values in these column.

*2- Unique constraints:* Unique constraints are the same as primary key constraints with the exception that they accept null values.

*3- Not Null constraints:* Not Null constraints enforce the rows to have values in the column to which they are applied.

*4- Check constraints:* Check constraints demands that the values inserted should adhere to a given condition specified in the creation of the constraint.

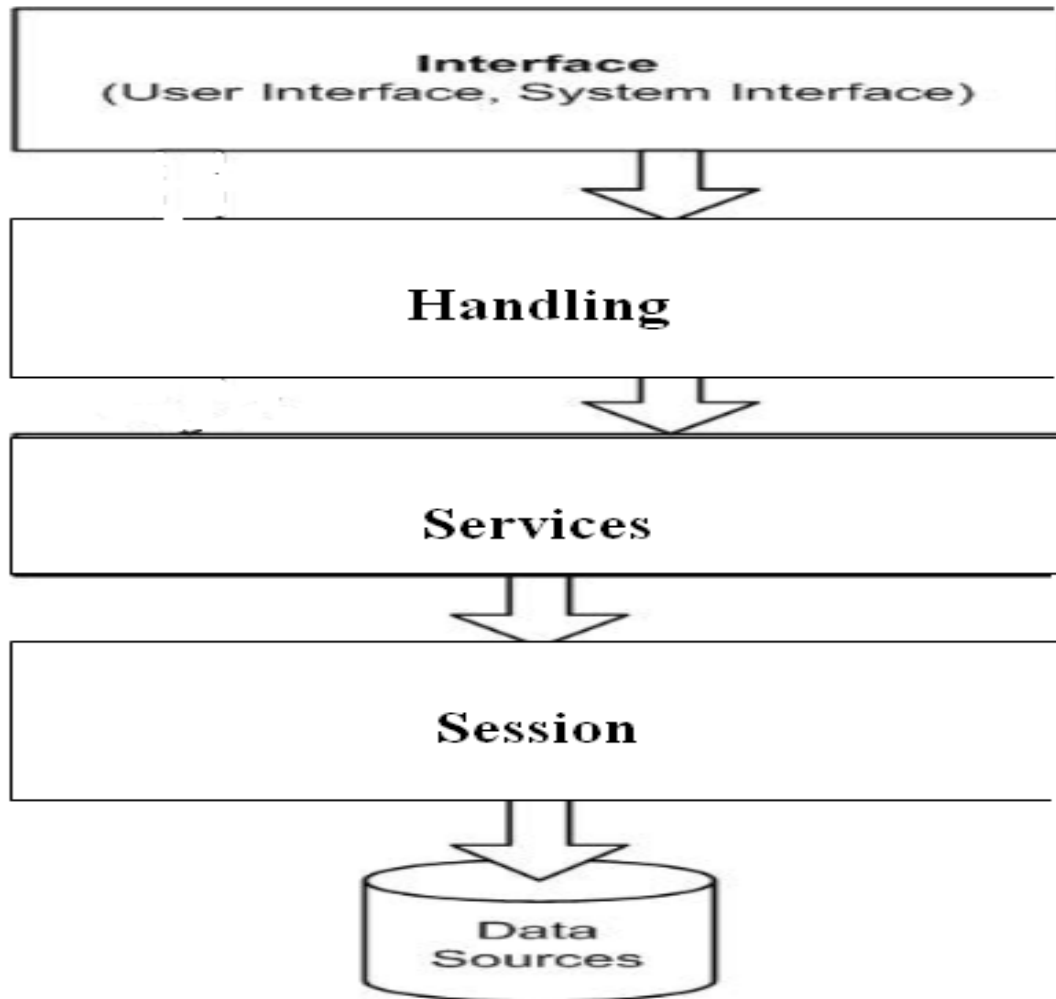
*5- Foreign Key:* This kind of constraints is used to construct relationships. As foreign keys requires that rows inserted in a child table cannot have values in the column that refer to the parent table that does not already exists in the parent table.

Some of the constraints included in our database design and resulted from the modeling done to the business process of the university are:

Table Name	Constraint Name	Constraint Type	Search Condition
EMPLOYEES	SYS_C009932	C	"LAST_NAME" IS NOT NULL
EMPLOYEES	SYS_C009933	C	"EMAIL" IS NOT NULL
EMPLOYEES	SYS_C009934	C	"HIRE_DATE" IS NOT NULL
EMPLOYEES	SYS_C009935	C	"JOB_TITLE" IS NOT NULL
EMPLOYEES	EM_EML_CK	C	regexp_like(email, '^([[:graph:]]+)([[:alnum:]]+)(.com org net it edu)\$')
EMPLOYEES	JT_CK	C	job_title in ('Employee', 'Professor')
EMPLOYEES	SYS_C0010273	C	employee_type in ('Professor', 'Employee')
EXAM_ENROLLMENT_REQUESTS	FK_EXAM_ID_EXAM_ENR_REQ	R	(null)
EXAM_ENROLLMENT_REQUESTS	FK_STUDENT_ID_EXAM_ENR_REQ	R	(null)
EXAM_ENROLLMENT_REQUESTS	SYS_C009928	C	"EXAM_ID" IS NOT NULL
EXAM_ENROLLMENT_REQUESTS	SYS_C009929	C	"STUDENT_ID" IS NOT NULL
EXAM_ENROLLMENT_REQUESTS	SYS_C009930	C	"REQUEST_DATE" IS NOT NULL
STUDENTS	SYS_C009915	C	"LAST_NAME" IS NOT NULL
STUDENTS	SYS_C009916	C	"EMAIL" IS NOT NULL
STUDENTS	SYS_C009917	C	"ENROLLMENT_DATE" IS NOT NULL
STUDENTS	ST_EM_CK	C	regexp_like(email, '^([[:graph:]]+)([[:alnum:]]+)(.com org net it edu)\$')
STUDENT_COURSES	SYS_C009995	R	(null)
STUDENT_COURSES	FK_FINAL_ID	R	(null)
STUDENT_COURSES	FK_MIDTERM_ID	R	(null)
STUDENT_COURSES	SYS_C009996	R	(null)

#### 1.4- Layers and Modules of the Client Application:

The system was divided into five layers each of these layers was implemented separately. Each layer can communicate only either with the directly successor layer or with the directly predecessor layer.



The Data-Layer contains database which consists of the physical data files and the logical representation of those files in the form of tables, views, constraints, and sequence. The database also contains logical entities that maintain the security and access control to the data, for example users, and roles. The database also carries metadata represented in the Data Dictionary in the form of views which contains information about all the objects, users, roles ... etc in the database. Data Dictionary can only be manipulated by the database administrator.

The Session-Layer wraps up the operations that could be done against the database and offers to the upper layers an interface or API that can be used to manipulate the data regardless where the data is placed. Some of the operations offered by the session-layer are execute select and execute DML:

## Method Summary

int	<a href="#">executeDML</a> (java.lang.String sql)
java.sql.ResultSet	<a href="#">executeSelect</a> (java.lang.String sql)
void	<a href="#">fireUpdate</a> ()
java.sql.Connection	<a href="#">getConnection</a> ()

The executeDML() process is used by the services-layer whenever a DML command is to be executed. The method takes a string argument representing the SQL DML statement and returns the number of rows affected by this statement. The executeSelect() method is used by the services layer whenever a SQL select statement is to be executed. This method takes a string argument and returns a ResultSet object that represents the result returned by the database.

As a sum up the session layer wraps up the database manipulation and hides this manipulation from the upper layers. The main advantages provided by this layer are:

- 1- Offline retrieval of data in case of connection loss.
- 2- Minimize the time of manipulating the database over the cloud by keeping track of the query results.
- 3- Enforces the software concept of data hiding and abstraction.

The services layer constructs the SQL queries and translates the operations initiated by the user through the GUI into SQL code. Whenever the handling layer is making a query against the database to retrieve results from the database to be represented as GUI forms, actually the services layer is manipulated. In other words, the services layer acts as an intermediate between the handling layer and the data retrieval code. Classes of this layer includes:

Class Summary	
<a href="#">CheckRightPassword</a>	
<a href="#">ConfirmationEmail</a>	
<a href="#">CourseConfirmation</a>	
<a href="#">CourseServices</a>	
<a href="#">ExamConfirmation</a>	
<a href="#">GeneralServices</a>	
<a href="#">InitializeProfInfo</a>	
<a href="#">InitializeViewLectures</a>	
<a href="#">NewAdmin</a>	
<a href="#">NewEmployee</a>	
<a href="#">NewStudent</a>	
<a href="#">ReadingFileFromDB</a>	
<a href="#">RequestRejection</a>	
<a href="#">RequestsInsertion</a>	
<a href="#">RequestsRetrievalServices</a>	
<a href="#">ResultSetTableModel</a>	
<a href="#">SendingEmail</a>	
<a href="#">UpdateProfessorProfilePanel</a>	
<a href="#">UploadFileToDB</a>	

The most important feature of this layer is that it provides a portable API for sending emails. This feature built based on the Java mail API, which is the open source extension provided by Oracle. The Java mail API is used to send emails using external SMTP servers like Gmail, Hotmail, and Yahoo.

The Handling-layer is used to manage the GUI forms by hiding the services initiated by buttons and whatsoever from the final GUI representation of these services. In other words the handling layer maps every button, form, GUI table, and list to a service provided to the system by the services layer. For example pressing search in the search courses panel of the GUI layer:

File View Personal Account Tools Help

Main Search Courses Search Courses

Select Courses By

No of Credits:

No. of Prerequisites:

Prerequisites:

Course Status:

Course ID:

Course Name:

Professors:

Help

This process is mapped into the search courses process in the services layer by the handling layer as follow:

```

public void actionPerformed( ActionEvent evt)
{
    Services.StudentSearchCourses search;
    if( evt.getActionCommand().equalsIgnoreCase("search"))
        search = new
Services.StudentSearchCourses( manipulate,
                                parent.getCourseID().equalsIgnoreCase("") ? null : parent.getCourseID(),
                                parent.getCourseName().equalsIgnoreCase("") ? null : parent.getCourseName(),

```

```

        null,
        parent.getCredits(),
parent.getNumOfPrequisites(),
        parent.getStatus1().equalsIg
noreCase("any") ? null : parent.getStatus1(),
        parent.getStatus2().equalsIg
noreCase("any") ? null : parent.getStatus2());

        else
            search = new
Services.StudentSearchCourses( manipulate, null,
null, null, -1, -1, null, null);

            ViewCurrentCourses view = new
ViewCurrentCourses( parent.getOurParent());
            view.setTable( search.getResult() );
            parent.getTabbedPane().setComponentA
t( parent.getTabbedPane().getSelectedIndex() ,
view );
        }

```

Each of the GUI layer and the handling layer is divided into five separate modules:

1- *The Student module*: Is made to be used by the students of the university using the system. It offers multiple functions that the student may be using to update, view, enroll ... etc in a course or exam.

2- *The Professor module*: Is made to be used by professors independently from the administrator. As an example he can schedule a midterm, quiz or final exam to be seen by the whole users of the system after the scheduling request being confirmed by the system administrator. He can also view his students' attendance or even other students on the system, but he cannot change the attendance sheet values as it is an administrator authority.

3- *The Control module*: Is made to be used by the professors or employees that have the authority of submitting the exam scores in the database. This module also enables the control users to edit, delete or view all the results for the students, considering that the system has the highest authority administrator that should confirm the requested changes in the results and place it on the database.

4- *The Graduate module*: Is made to be used by the students being graduated from the university. Those users can use the system in

order to grab some useful important information to them. This grabbed information for example can be viewing or printing student transcript, viewing accumulated GPA, and viewing currently stored courses on the database in case of that graduate needs to make post-graduate studies.

5- *The Admin module*: It is the user that has the highest privilege on the system. The admin user is the user that takes and confirms important decisions, such as confirming registration requests made by the users registering the system... etc.

## 1.5- Products Used in the Project:

### 1.5.1 Oracle data server:

Oracle database is an object-relational database management system (ORDBMS) produced and marketed by Oracle Corporation. Oracle database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key for solving the problems of information management. In general, a server reliably manages a large amount of data in a multi-user environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance. A database server also prevents unauthorized access and provides efficient solutions for failure recovery. Oracle Database is the first database designed for enterprise grid computing, the most flexible and cost effective way to manage information and applications. Enterprise grid computing creates large pools of industry-standard, modular storage and servers. With this architecture, each new system can be rapidly provisioned from the pool of components. There is no need for peak workloads, because capacity can be easily added or reallocated from the resource pools as needed. The database has logical structures and physical structures. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures.

### 1.5.2 Java Database Connectivity:

Commonly referred to as JDBC, is an application programming interfaces language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases. A JDBC-to-ODBC bridge enables connections to any ODBC-accessible data source in the JVM host. Environment. JDBC helps you to write java applications that manage these three programming activities:



- Connect to a data source, like a database.
- Send queries and update statements to the database.
- Retrieve and process the results received from the database in answer to your query.

1. *History and implementation:* Sun Microsystems released JDBC as part of JDK 1.1 on February 19, 1997. It has since formed part of the Java Standard Edition. The JDBC classes are contained in the Java package `java.sql` and `javax.sql` starting with version 3.0, JDBC has been developed under the Java Community Process.

2. *Functionality:* JDBC allows multiple implementations to exist and be used by the same application. The API provides a mechanism for dynamically loading the correct Java packages and registering them with the JDBC Driver Manager. The Driver Manager is used as a connection factory for creating JDBC connections. JDBC connections support creating and executing statements. These may be update statements such as SQL's CREATE, INSERT, UPDATE and DELETE, or they may be query statements such as SELECT. Additionally, stored procedures may be invoked through a JDBC connection. JDBC represents statements using one of the following classes:

- [Statement](#) – the statement is sent to the database server each and every time.
- [PreparedStatement](#) – the statement is cached and then the execution path is pre-determined on the database server allowing it to be executed multiple times in an efficient manner.
- [CallableStatement](#) – used for executing stored procedures on the database. Update statements such as INSERT, UPDATE and DELETE return an update count that indicates how many rows were affected in the database. These statements do not return any other information. Query statements return a JDBC row result set. The row result set is used to walk over the result set. Individual columns in a row are retrieved either by name or by column number. There may be any number of rows in the result set. The row result set has metadata that describes the names of the columns and their types. There is an extension to the basic JDBC API in the `javax.sql`. JDBC connections are often managed via a connection pool rather than obtained directly from the driver.

### 1.5.3 NetBeans:

Refers to both a platform framework for Java desktop applications, and an integrated development environment (IDE) for developing with Java, JavaScript, PHP, Python, Ruby, Groovy, C, C++, Scala, Clojure, and others. The NetBeans IDE is written in Java and can

run anywhere a JVM is installed, including Windows, Mac OS, Linux, and Solaris. A JDK is required for Java development functionality, but is not required for development in other programming languages. The NetBeans platform allows applications to be developed from a set of modular software components called *modules*. Applications based on the NetBeans platform (including the NetBeans IDE) can be extended by third party developers. The NetBeans Platform is a reusable framework for simplifying the development of Java Swing desktop applications. The NetBeans IDE bundle for Java SE contains what is needed to start developing. NetBeans plug-in and NetBeans Platform based applications; no additional SDK is required. Applications can install modules dynamically. Any application can include the Update Center module to allow users of the application to download digitally-signed upgrades and new features directly into the running application. Reinstalling an upgrade or a new release does not force users to download the entire application again.

## **Chapter 2**

### **Amro Salem Contribution**

#### **2.1- Designing Implementation:**

The implementation was designed to make use of the features and the technologies of the available products. Beside using the features and the technologies of the available products, common programming techniques was utilized to encapsulate the data layer and make use of the upper layers in such a way that those layers is not directly manipulating the database. This was done by utilizing the session coordinate in the system. The session package containing the system offers a lot of features that would make the system more usable. Usability of the system was the main factor considered when designing the data manipulation scheme. The session package contains two classes. Those classes are DBManAPI and Session. The Session class carries the HashMap that holds each SQL query as a key and the result of this SQL query as a ResultSet value. Also it offers basic manipulation services such as retrieving data and updating data in the HashMap. In other words the way data is retrieved is hidden from the upper layers using the session class, so whenever a query is made, the user do not know whether it is retrieved directly from the database or from the history contained in the HashMap.:

```
package session;

import com.sun.rowset.CachedRowSetImpl;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.HashMap;
import java.util.Map.Entry;
import javax.sql.RowSet;
import javax.swing.JOptionPane;

public class Session implements Runnable
{

    public HashMap< String, RowSet > hash;
    Connection connect;

    public Session( Connection connect)
    {
        this.connect = connect;
        hash = new HashMap<String, RowSet>();
    }
}
```

```

    }

    public RowSet addEntry( String sql, RowSet
result)
    {
        return hash.put( sql, result );
    }

    public RowSet getOperation( String element )
    {
        return hash.get( element );
    }

    public void updateOccured()
    {
        try
        {
            for( Entry< String, RowSet > ent :
hash.entrySet() )
            {
                Statement statement =
connect.createStatement( ResultSet.TYPE_SCROLL_SENSIT
IVE, ResultSet.CONCUR_READ_ONLY );
                CachedRowSetImpl result = new
CachedRowSetImpl();
                result.populate(statement.execute
eQuery( ent.getKey() ));
                ent.setValue( result );
                hash.put(ent.getKey(),
ent.getValue());
            }
        }

        catch( SQLException ex )
        {
            JOptionPane.showMessageDialog( null
, "Error Ocurrred While Updating Session Data!", "Data
Base Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    public boolean keyExists( String sql )
    {
        return hash.containsKey( sql );
    }

    public void run()

```

```

        {
            updateOccured();
        }
    }

```

This class offers a user multiple functions, the most important function is:

1- The user can access the data retrieved before, during the session even if the connection is terminated (offline).

2- It decreases the time required to fetch data from the Oracle database server, because no every time data is required it is fetched from database. In other words, if this data is being fetched for the first time it will be brought from the database then placed in HashMap then viewed to the user. On the other hand, if this data was fetched before, so the result of the query is fetched from the HashMap saving the time of querying the database over the cloud.

The DBManAPI class offers common manipulation tasks built on the session class. Those operations are executing a select statement against the database and executing a DML/DDI statement against the database. :

```

package session;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import javax.swing.JOptionPane;
import com.sun.rowset.CachedRowSetImpl;

public class DBManAPI
{
    Connection connect;
    Session session;

    public DBManAPI( Connection con)
    {
        session = new Session(con);
        connect= con;
    }

    public ResultSet executeSelect( String sql )

```

```

        {
            try
            {
                if( session.keyExists(sql))
                {
                    if( session.getOperation( sql )
!= null )
                    {
                        return session.getOperation(
sql );
                    }
                    else
                    {
                        Statement statement =
connect.createStatement( ResultSet.TYPE_SCROLL_SENSIT
IVE, ResultSet.CONCUR_READ_ONLY );
                        CachedRowSetImpl res = new
CachedRowSetImpl();
                        res.populate( statement.exec
uteQuery( sql ) );
                        statement.close();
                        System.gc();
                        return
session.addEntry( sql, res );
                    }
                }
                else
                {
                    Statement statement =
connect.createStatement( ResultSet.TYPE_SCROLL_SENSIT
IVE, ResultSet.CONCUR_READ_ONLY );
                    CachedRowSetImpl res = new
CachedRowSetImpl();
                    res.populate( statement.executeQ
uery( sql ) );
                    statement.close();
                    System.gc();
                    session.addEntry(sql, res);
                    return session.addEntry(sql,
res);
                }
            }
            catch(SQLException ex)
            {
                JOptionPane.showMessageDialog( null,
ex, "ERROR", JOptionPane.ERROR_MESSAGE);
            }
        }
    }
}

```

```

        ex.printStackTrace();
        return null;
    }

    }

    public int executedDML( String sql ) throws
SQLException
    {
        Statement statement = statement =
connect.createStatement( ResultSet.TYPE_SCROLL_SENSIT
IVE, ResultSet.CONCUR_READ_ONLY );
        int result =
statement.executeUpdate(sql);
        statement.close();
        System.gc();
        return result;
    }

    public void fireUpdate()
    {
        Thread th = new Thread(session);
        th.start();
    }
    public Connection getConnection(){
        return connect;
    }
}
}

```

One important feature of this class is that it makes use of the threading API of the Java to update the result contained in the HashMap in the background without pausing the execution of the program. This could be noticed in fireUpdate() method which creates a new instance of Thread class to achieve virtually parallel processing.

## 2.2- Identifying the Users of the System and Creating the Corresponding Roles:

A system of a university normally has more than one type of users. When the target users of the system were determined a separate kind of logical category of privileges was created. Normally in Oracle database this logical category of privileges is grouped into schema object called 'Role'. The roles that were created for the database system are Student, Professor, Administrator, Control and Graduate. Those roles were mapped also in the GUI and the Handling layer of the system. As an example of creating a role is:

```
CREATE ROLE student;
GRANT CREATE SESSION TO student;
GRANT SELECT ON courses TO student;
GRANT SELECT ON exams TO student;
GRANT UPDATE student_courses( status ) TO
student;
.
.
.
.
```

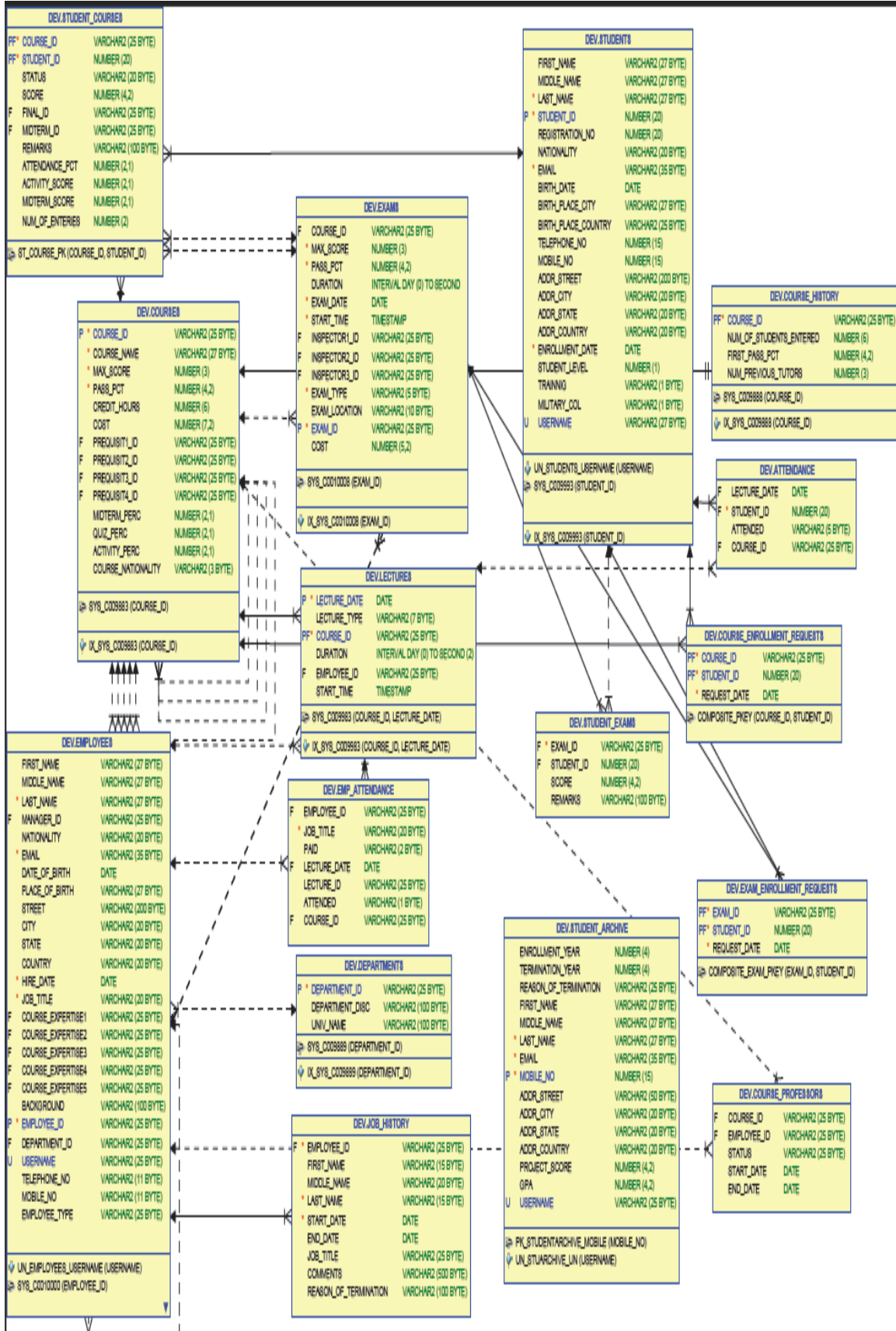
The previous code is a snippet of the code of creating a role. Each user that is granted access to the system is assigned a new account with a unique username and password. Whenever a new account is created Oracle database assigns a new schema, this schema can hold any kind of objects, whether a tables, views, sequence, or index.

The tables composing the system are all contained in one schema this schema is called developer. The developer schema has its own username and password which is 'DEV' and 'DEV' respectively. Access to this schema is planned to be granted only to the database administrator of the system.

After logically grouping the data that will be saved in the database, the ER model was created. The ER model represents:

- 1- How data would be stored.
- 2- What data would be stored.
- 3- What are the logical relationships between the tables.
- 4- What views would be created on these tables.
- 5- What users will be accessing which information.
- 6- Which columns will be the primary keys of the tables.

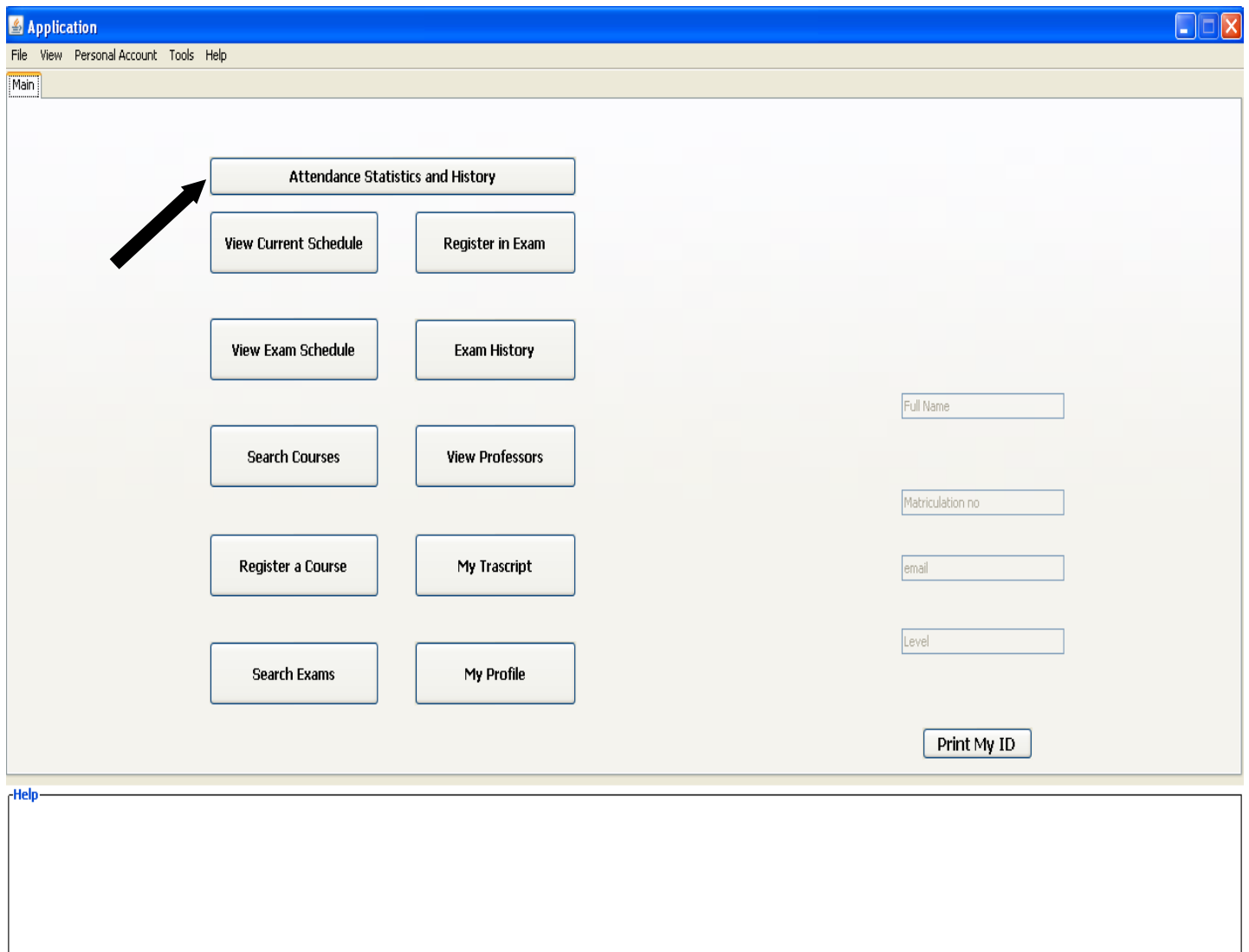


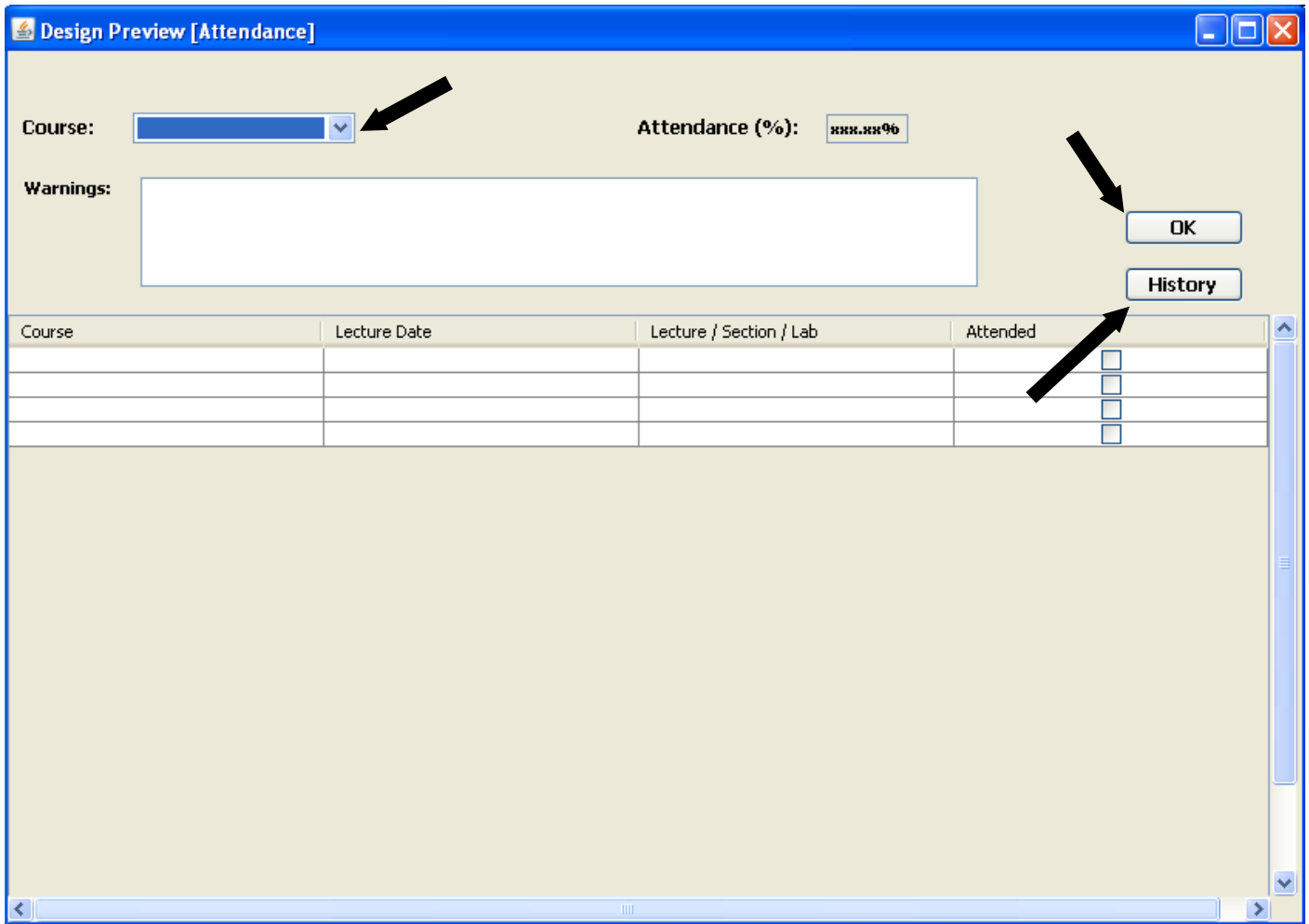


## 2.3- Some Functions of the Student Module:

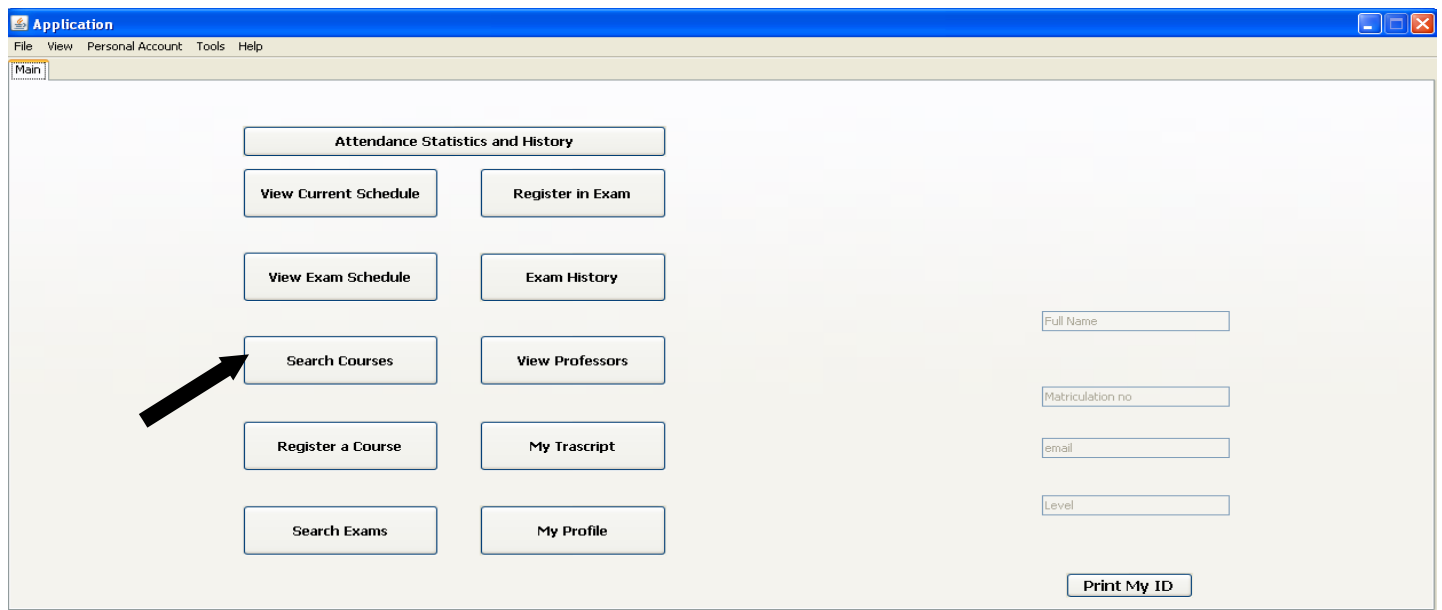
The Student module contains the GUI and the Handling code that models the typical operations that would be initiated by a student using the system. These operations reflect the privileges granted for the student and hide the operations that are not allowed for a student. For example the user that logged in as a student cannot remove another student from the database. Some of those functions are mentioned in the following sub-sections.

### 2.3.1- Reviewing the attendance in a given course:





2.3.2- Keeping track of the courses of the user student:



**Design Preview [SearchCourses]**

Select Courses By

No of Credits:

Course ID:

No. of Prerequisites:

Course Name:

Prerequisites:

Professors:

Course Status:

**Design Preview [ViewCurrentCourses]**

Course ID	Course Name	Full Mark	Pass (%)	Credit Hours	Cost	Prerequisites	Final (%)	Midterm (%)
1234567890	12345678901234...	123	1,234	1234	1,234			

## 2.4- Important Features of Student Module:

The features of the student module appear in how services reflect the operations that would be requested by a typical student. The coordinates of these features are:

1- SQL code that could be translated by the database.

2- Java code that offers a mapping between the Java variables and the values of the columns. For example, the student needs to search the courses of his department according to a certain criteria, in order to enroll in one of these courses. These criteria can be searching the courses by number of credit hours, course name, a specific prerequisite course, a specific number of prerequisites to that course or searching by the status of the course ('Currently Enrolled', 'Passed', 'Never Enrolled', 'Tried Before'). :

```
package Services;

import java.sql.ResultSet;
import session.DBManAPI;

public class StudentSearchCourses
{
    DBManAPI manipulate;
    String courseID;
    String courseName;
    String profID;
    int numOfCredits;
    int numOfPre;
    String status1;
    String status2;
    String query;

    public StudentSearchCourses( DBManAPI
manipulate )
    {
        this.manipulate = manipulate;
    }

    public StudentSearchCourses( DBManAPI
manipulate, String courseID, String courseName, String
profID, int numOfCredits, int numOfPre, String
status1, String status2)
    {
        this.manipulate = manipulate;
        this.courseID = courseID;
        this.courseName = courseName;
```

```

        this.profID = profID;
        this.numOfCredits = numOfCredits;
        this.numOfPre = numOfPre;
        this.status1 = status1;
        this.status2 = status2;
        query = String.format( "select
c.course_id \"Course ID\", c.course_name \"Course
Name\", "
                                +
"c.max_score \"Full Mark\", to_char( c.pass_pct ) || '
%%' \"Pass Percentage\", "
                                +
"c.credit_hours \"Number of Hours\",
nvl2( c.prequisit1_id, p1.course_name, '') \"Prequisit
1\", "
                                +
"nvl2( c.prequisit2_id, p2.course_name,
'' ) \"Prequisit 2\", "
                                +
"nvl2( c.prequisit3_id, p3.course_name,
'' ) \"Prequisit 3\", "
                                +
"nvl2( c.prequisit4_id, p4.course_name,
'' ) \"Prequisit 4\", "
                                + "sc.status \"Your
Status\", sc.score \"Your Score\", "
                                +
"sc.num_of_enteries \"Times You Entered The Course\" "
                                + "from
dev.courses c left outer join courses p1 "
                                + "on
( c.PREQUISIT1_ID = p1.COURSE_ID) "
                                + "left outer join
courses p2 "
                                + "on
( c.PREQUISIT2_ID = p2.COURSE_ID) "
                                + "left outer join
courses p3 "
                                + "on
( c.PREQUISIT3_ID = p3.COURSE_ID) "
                                + "left outer join
courses p4 "
                                + "on
( c.PREQUISIT4_ID = p4.COURSE_ID) "
                                + "left outer join
(select course_id, "

```

```

num_of_enteries "
student_courses "
upper( student_id )= '%S' ) sc "
= sc.COURSE_ID ) "
GeneralServices.getCurrentStudentID( manipulate );
setCourseID();
setCourseName();
setCredits();
setNumPre();
setStatus();
}

public final void setCourseID()
{
    if( courseID == null)
        return;
    else
        query += String.format(" OR
upper( c.course_id ) like '%%S%%'", courseID);
}

public final void setCourseName()
{
    if( courseName == null )
        return;
    else
        query += String.format(" OR
upper( c.course_name ) like '%%S%%'", courseName);
}

public final void setCredits()
{
    if( numOfCredits == -1)
        return;
    else
        query += String.format(" OR
c.credit_hours = %d", numOfCredits);
}

public final void setNumPre()
{
    if( numOfPre == -1)

```

```

        return;
    else
        query += String.format(" OR (0 + "
                                + "nvl2( c.prequisit1_id, 1,
0 )"
                                + " + nvl2( c.prequisit2_id,
1, 0 )"
                                + " + nvl2( c.prequisit3_id,
1, 0 )"
                                + " + nvl2( c.prequisit4_id,
1, 0 ) ) = %d", numOfPre);
    }

    public final void setStatus()
    {
        if( status1 == null && status2 == null)
            return;
        if( status1 == null && status2 != null)
            query += String.format( "or
upper( sc.status ) = '%S'", status2);
        if( status2 == null && status1 != null)
            query += String.format( "or
upper( sc.status ) = '%S'", status1);
        if( status2 == null && status1 != null )
            query += String.format( "or
upper( sc.status ) in ( '%S', '%S' )", status1,
status2);
    }

    public ResultSet getResult()
    {
        return manipulate.executeSelect(query);
    }
}

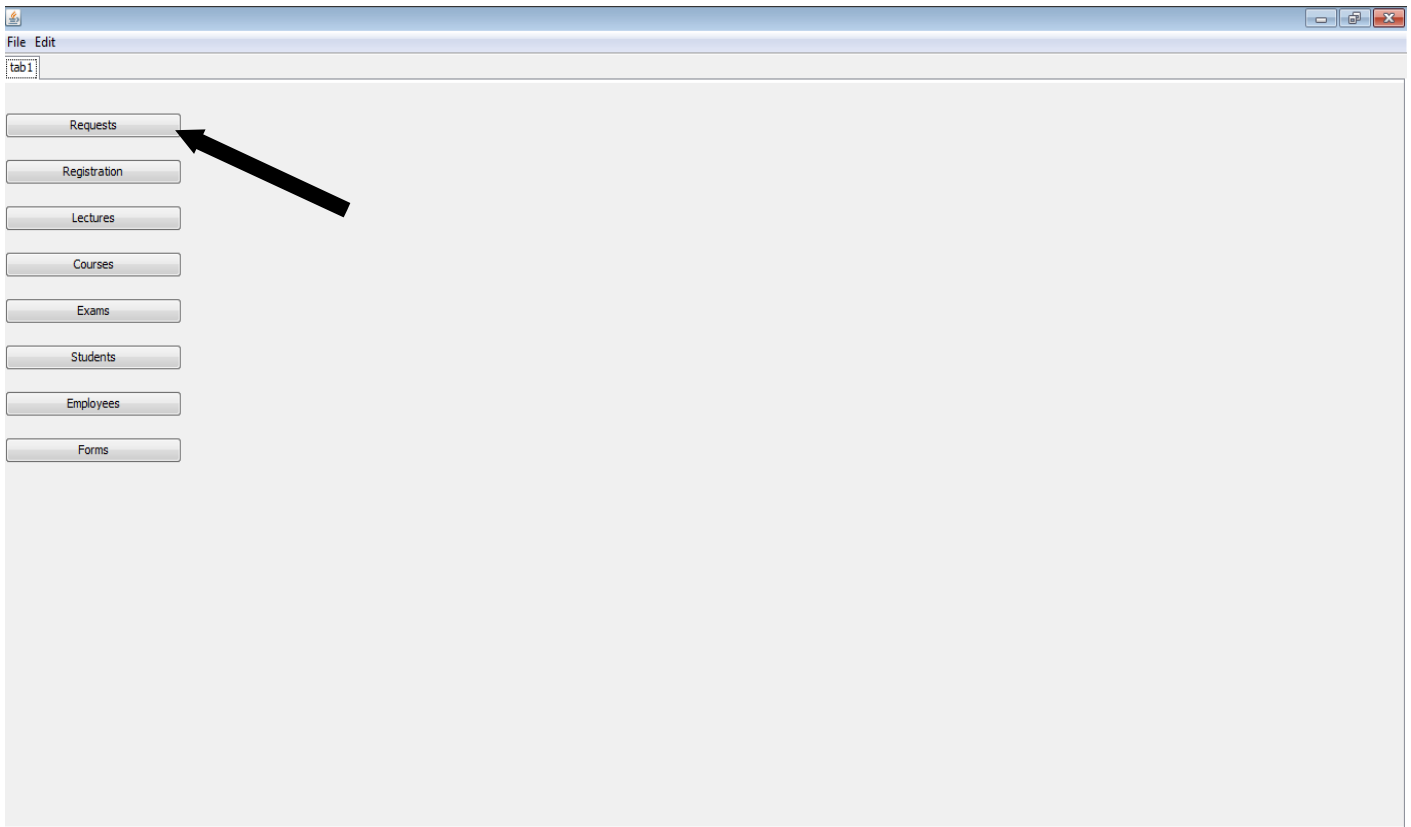
```

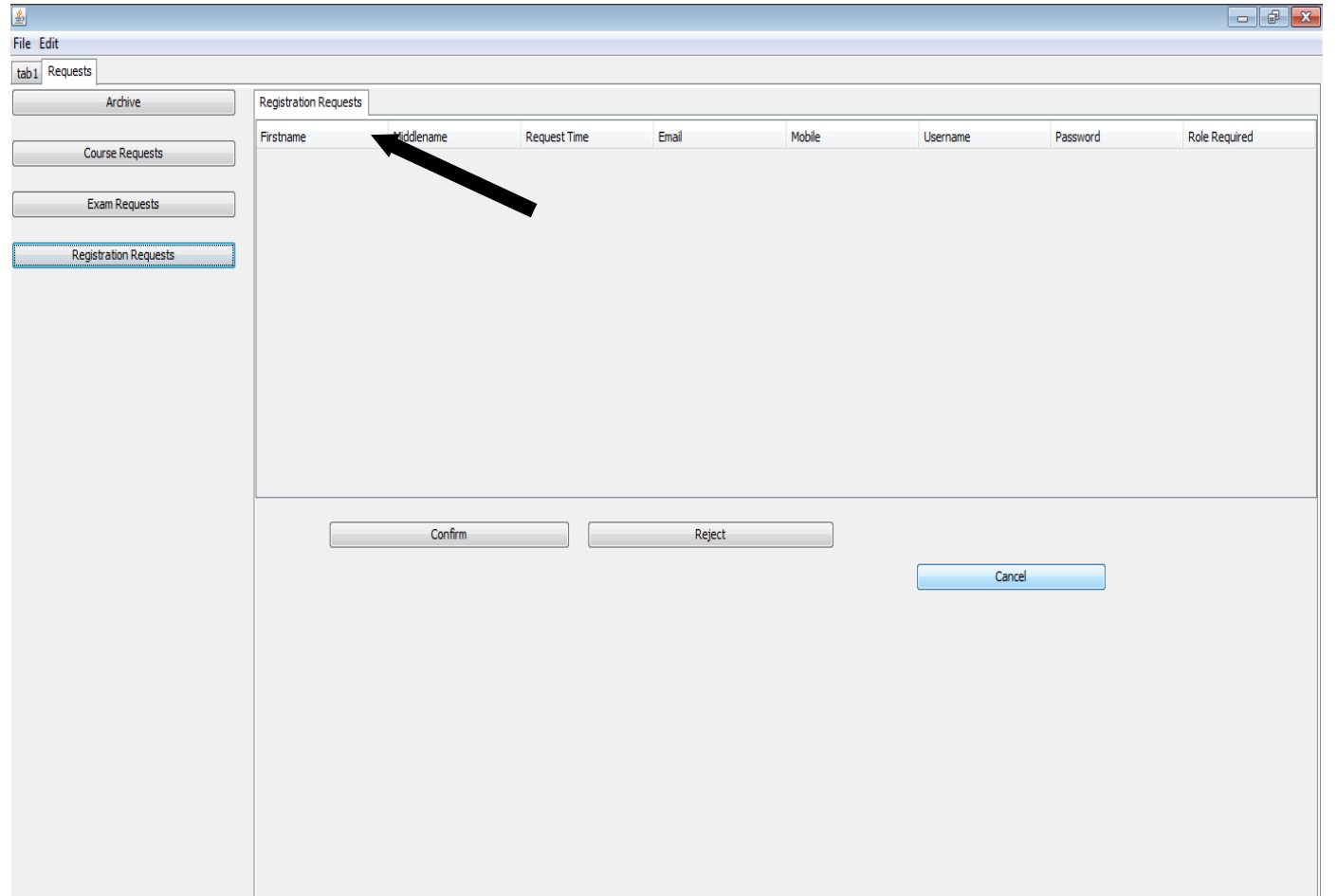


## 2.5- Some Functions of the Admin Module:

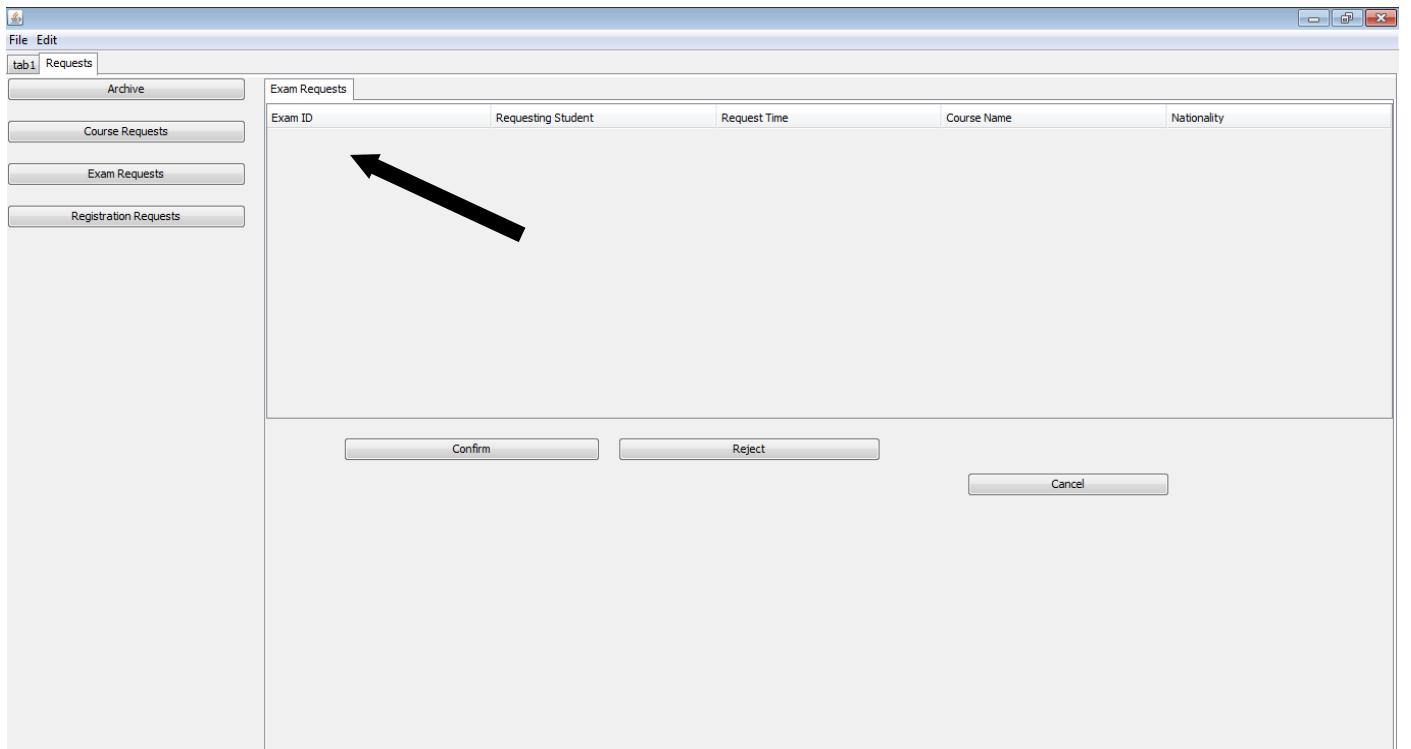
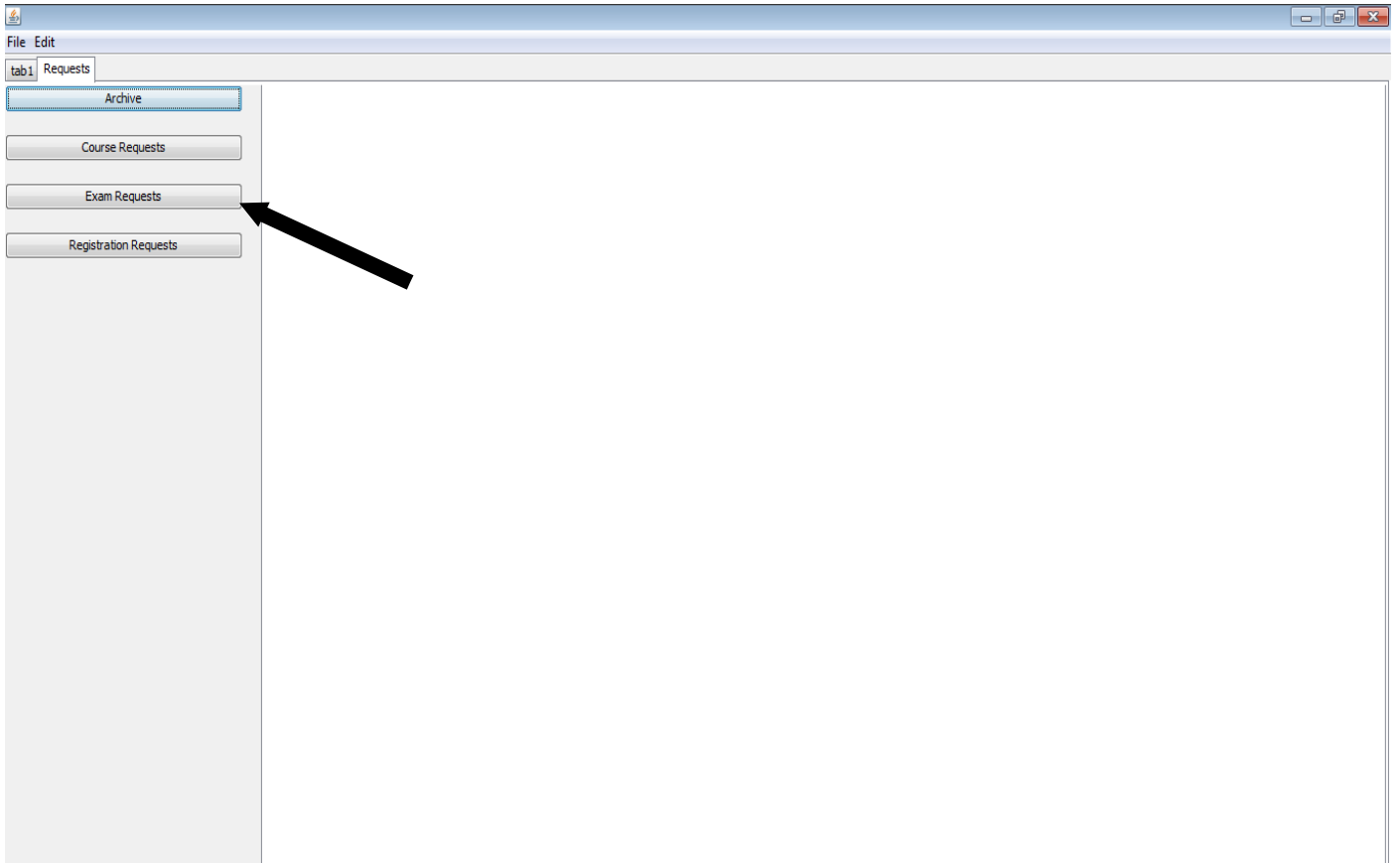
The admin module contains the GUI and handling code that models the typical operations that would be initiated by an admin using the system. These operations reflect the privileges granted for the admin. For example the user that logged in as an admin can remove a user from the database. Some of those functions are mentioned in the following sub-sections.

### 2.5.1- Confirming Registration Requests:

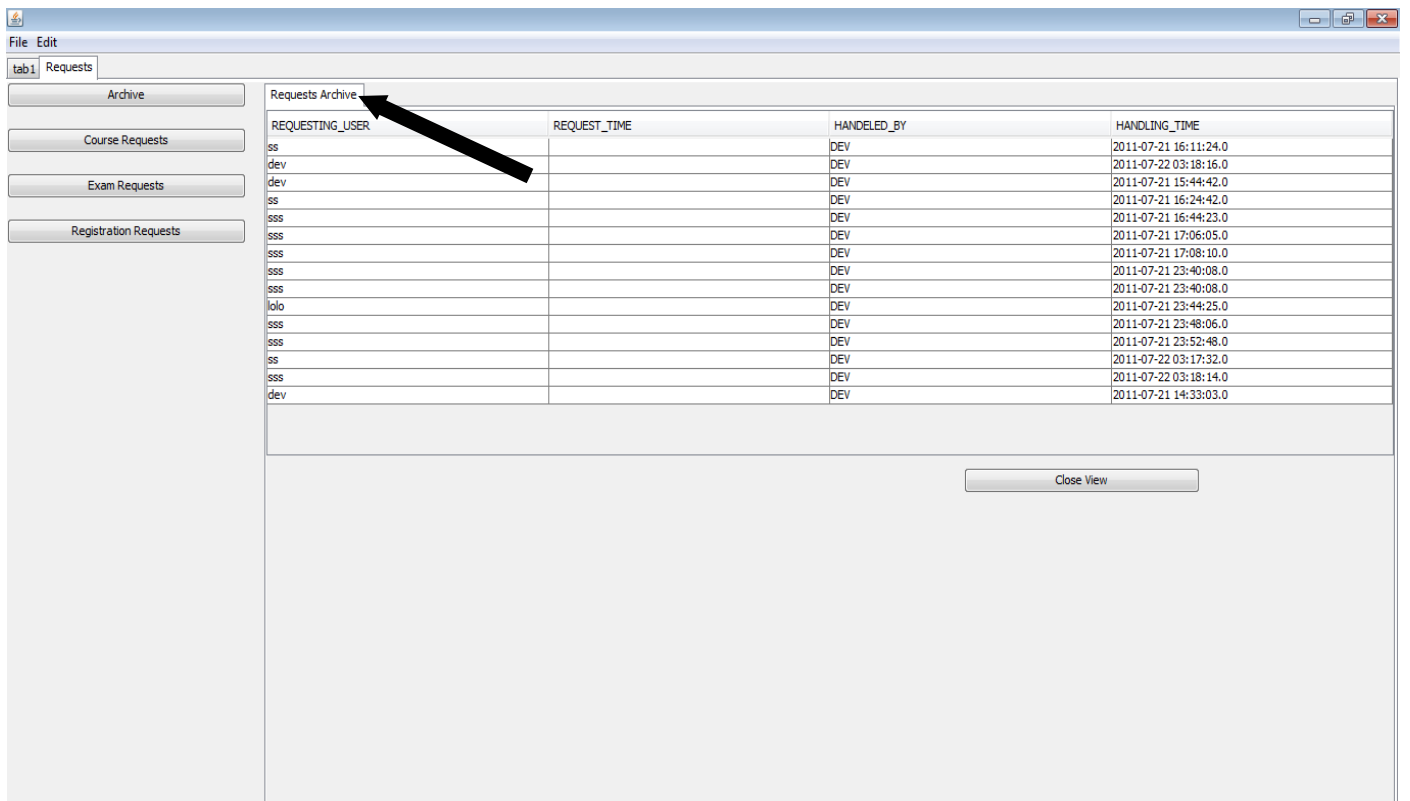
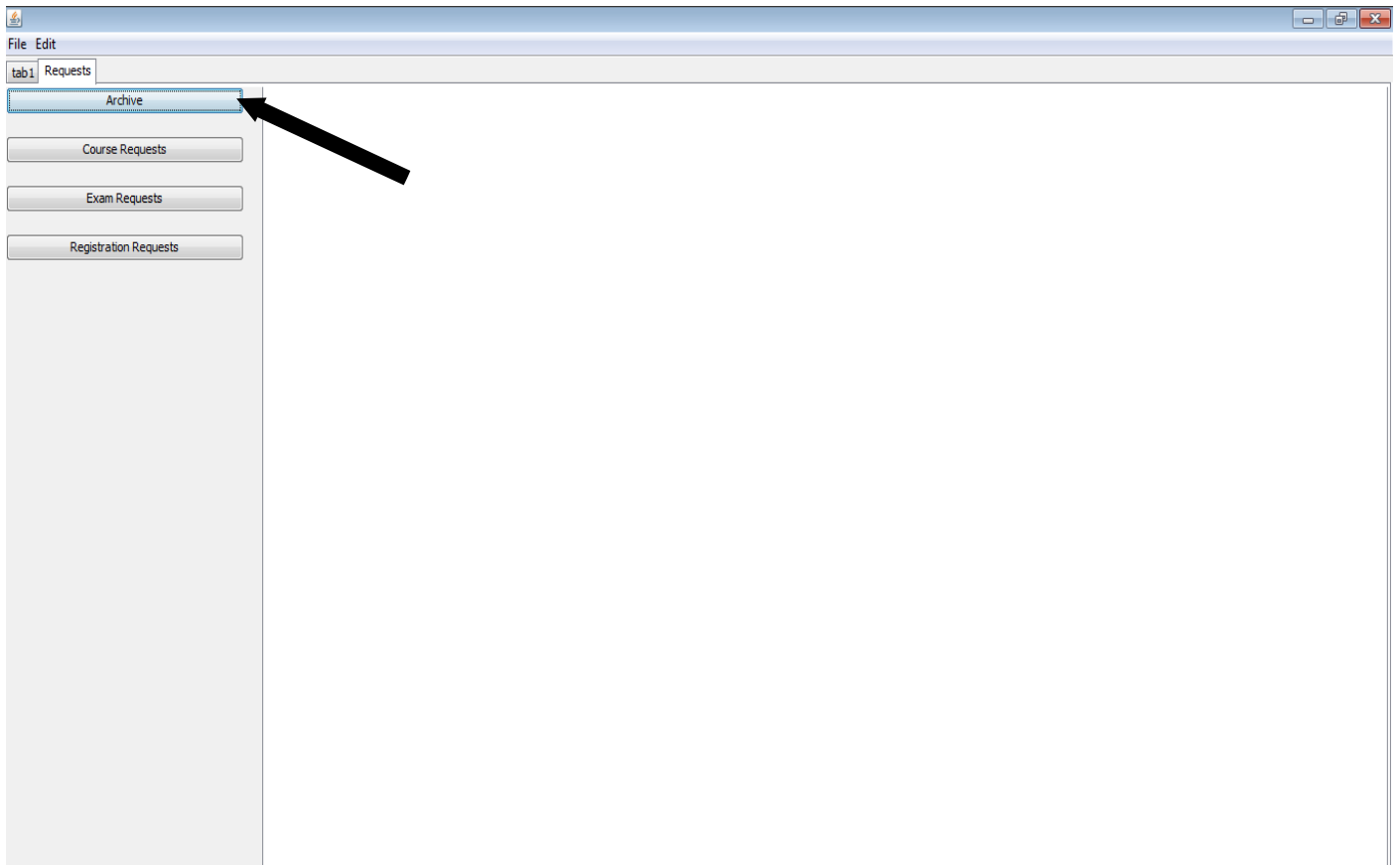




## 2.5.2- Viewing/Handling Exam Enrollment Requests:



### 2.5.3- Viewing Already Handled Request:

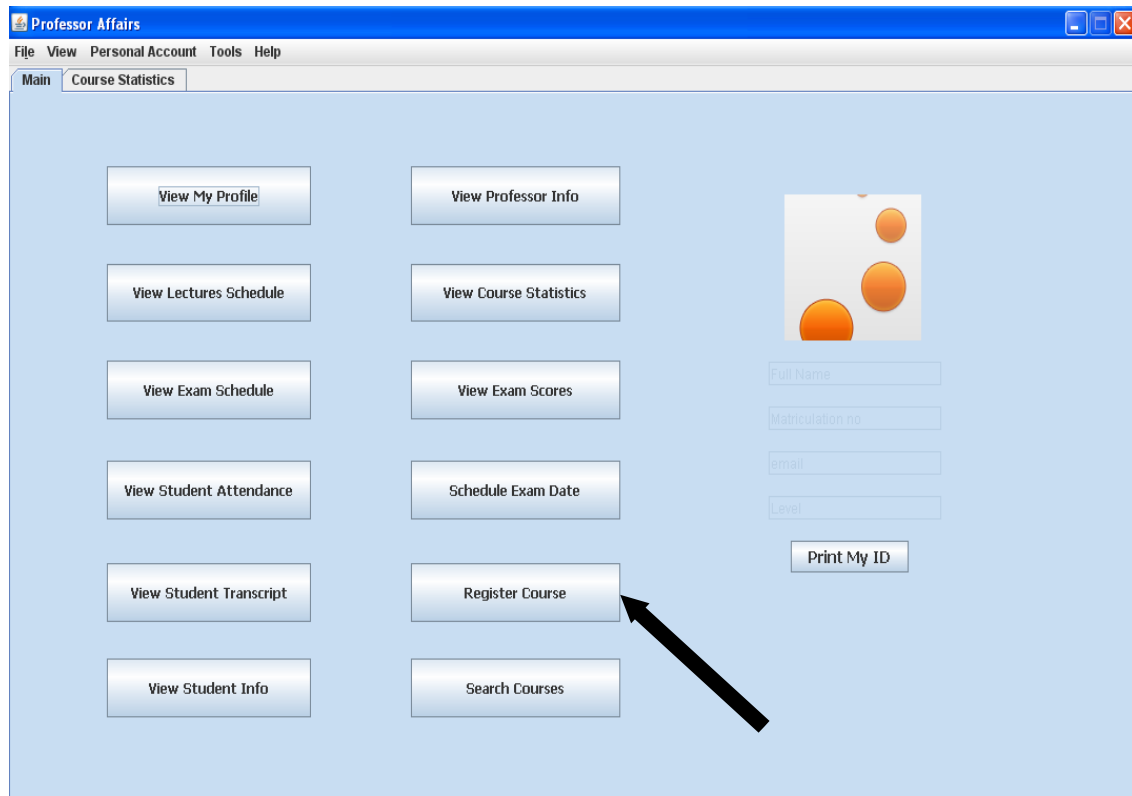


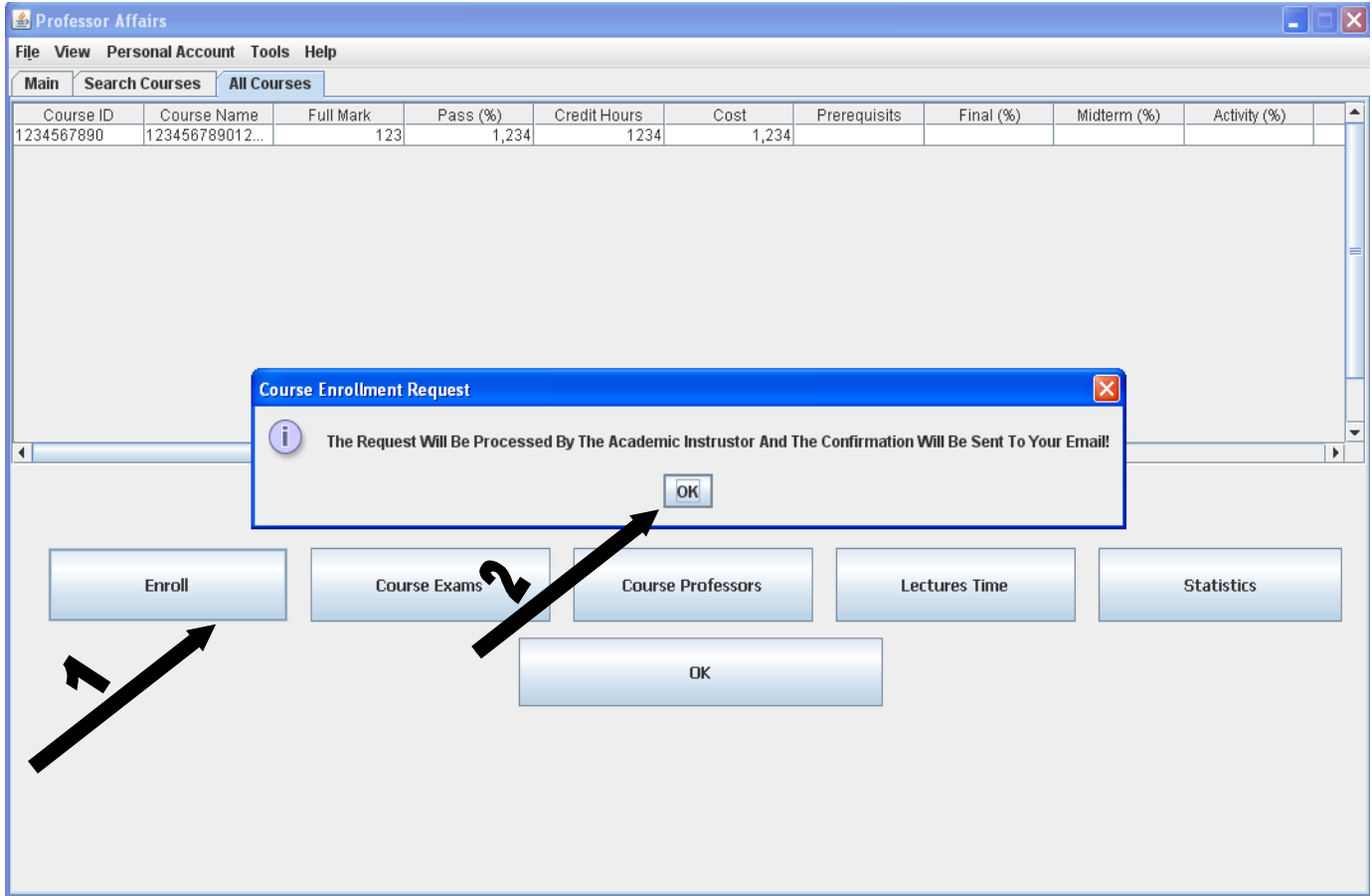
## Chapter 3 Ramy Saad Contribution

### 3.1- Some Functions of the Professor Module:

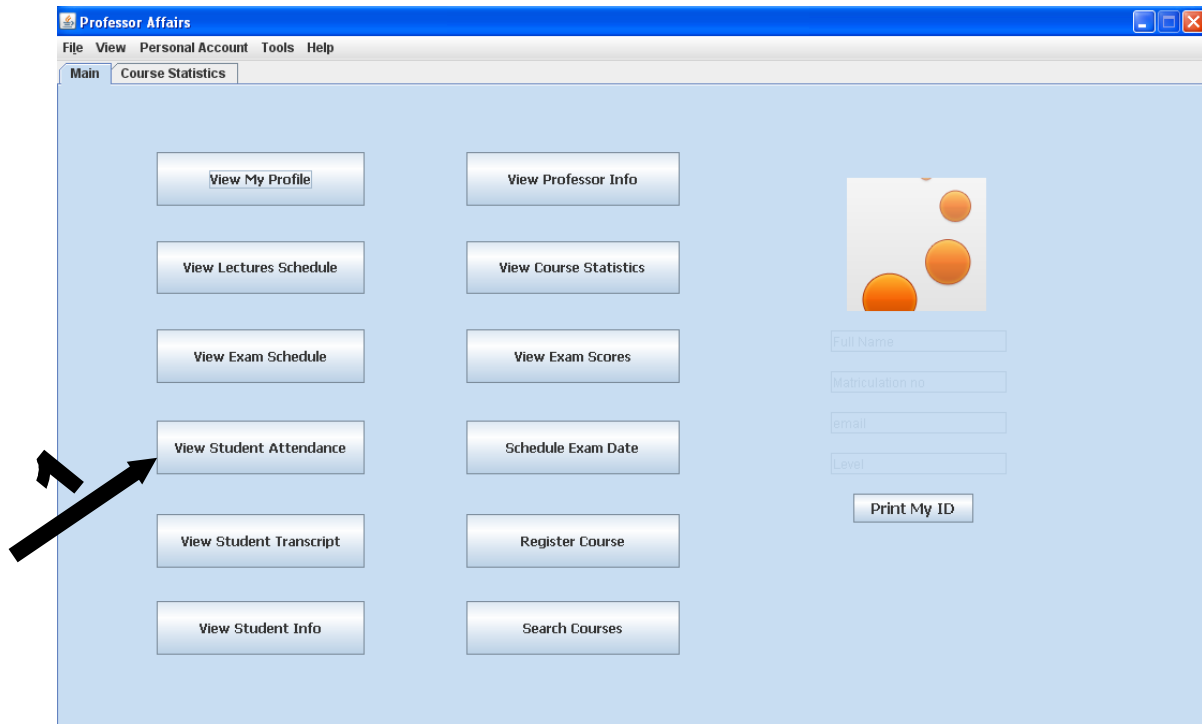
The Professor module contains the GUI and handling code that models the typical operations that would be initiated by a professor using the system. These operations reflect the privileges granted for the professor and hide the operations that are not allowed for a professor. For example the user that logged in as a professor cannot remove a student from the database or submit scores for a course that he is not teaching. Some of those functions are mentioned in the following sub-sections.

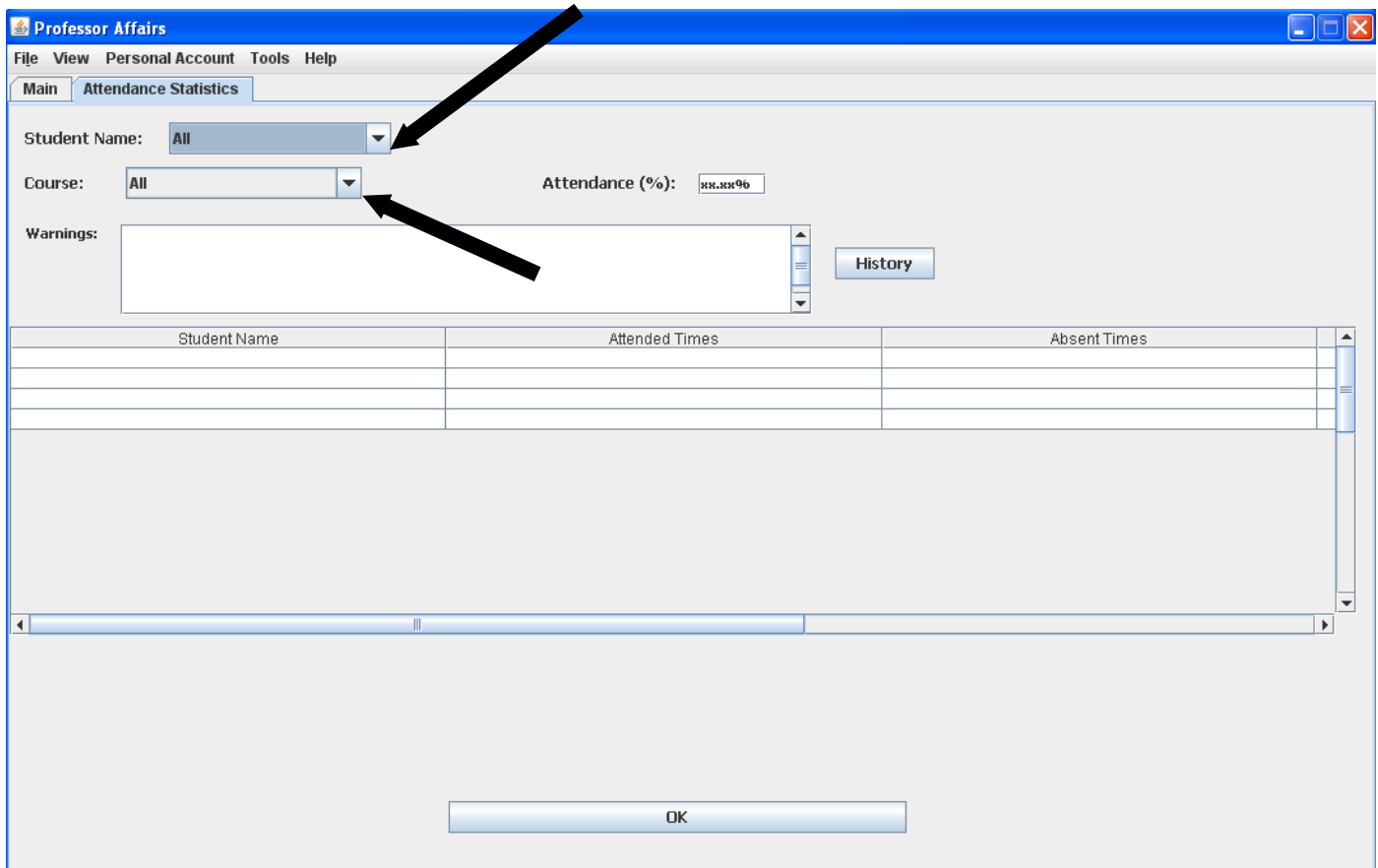
#### 3.1.1- Applying for Teaching a Course:



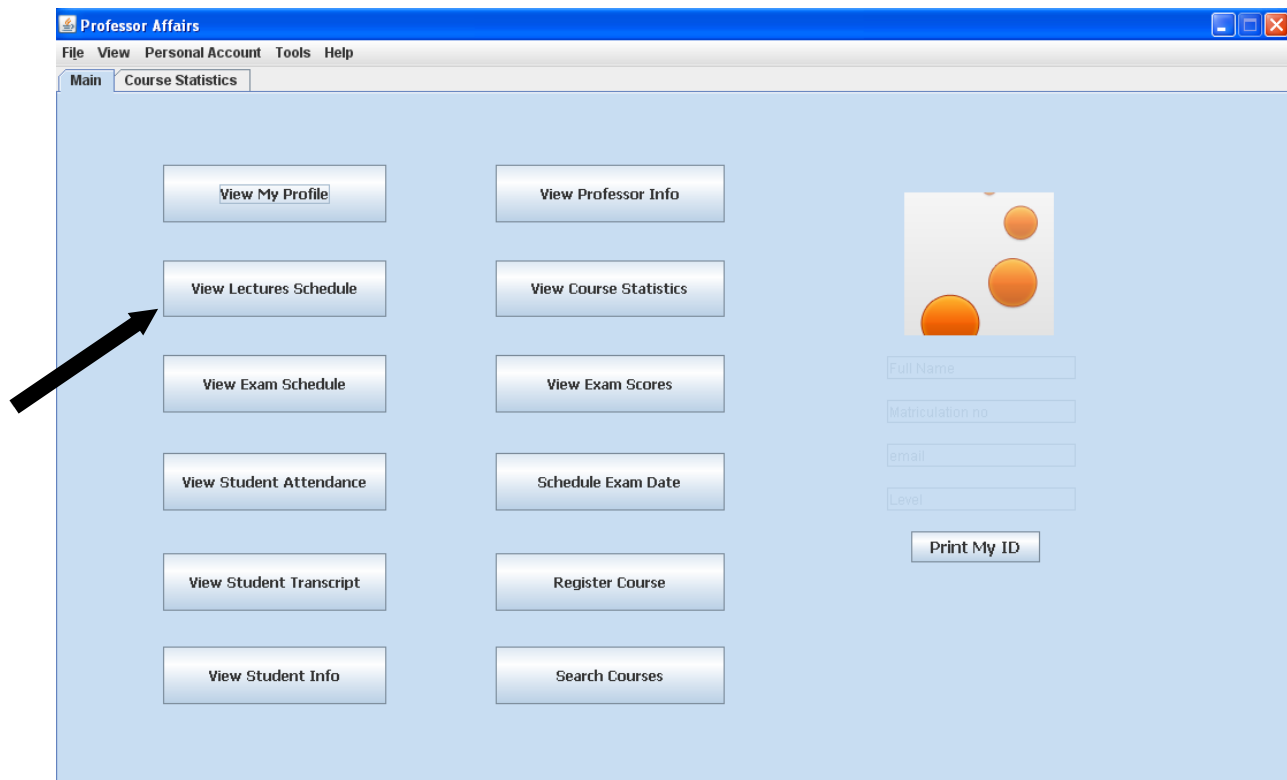


### 3.1.2- Reviewing the attendance of students:





### 3.1.3- Viewing Lectures Schedule:



Professor Affairs

File View Personal Account Tools Help

Main Search Courses All Courses Lectures Schedule

Course: math01

Day/ Lecture	Lecture 1	Lecture 2	Lecture 3	Lecture 4	Lecture 5
Saturday					
Sunday	true				
Monday					
Tuesday					
Wednesday	true				
Thursday		true			

OK

### 3.1.3- Generating Some Statistics for a Course:

Professor Affairs

File View Personal Account Tools Help

Main Course Statistics

View My Profile

View Lectures Schedule

View Exam Schedule

View Student Attendance

View Student Transcript

View Student Info

View Professor Info

View Course Statistics

View Exam Scores

Schedule Exam Date

Register Course

Search Courses

Full Name

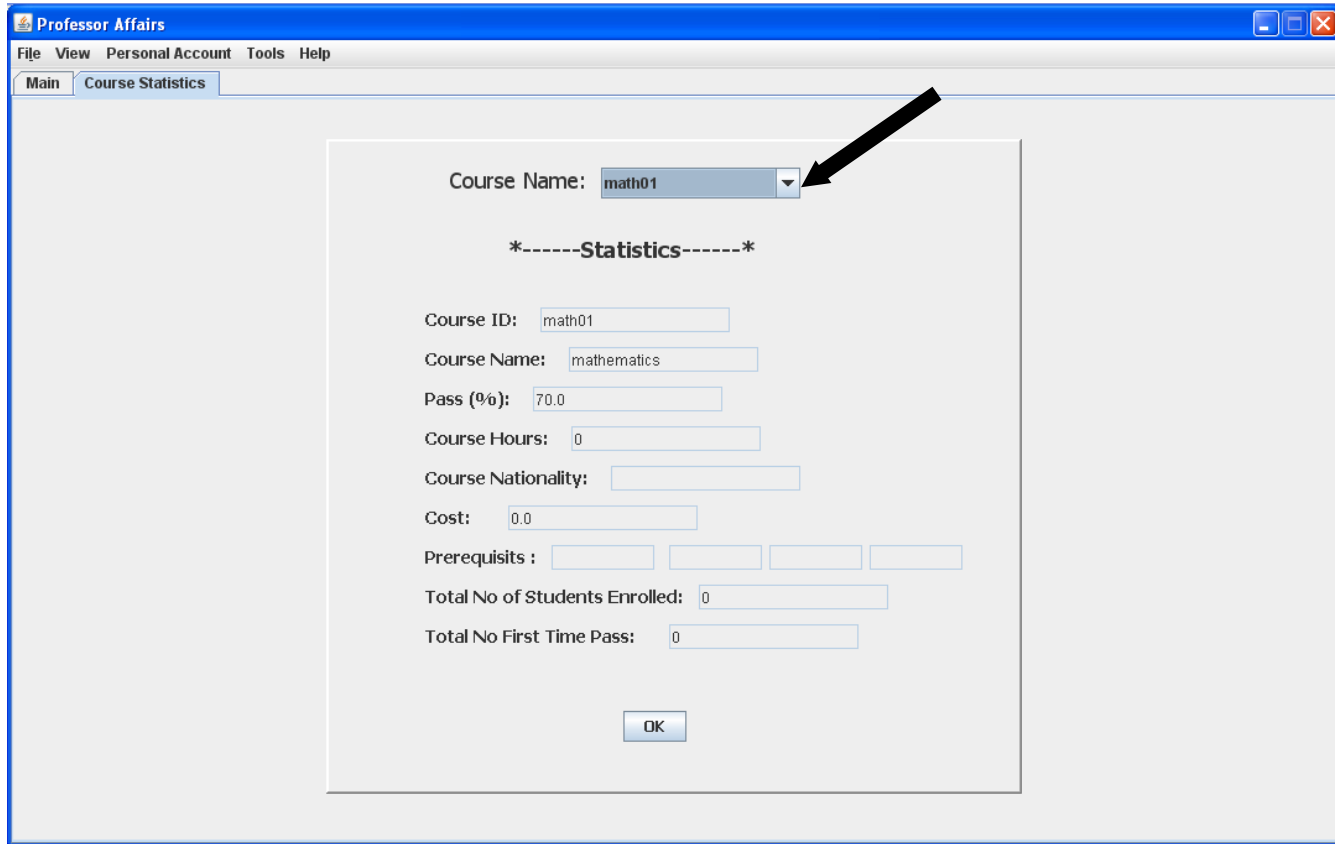
Matriculation no

email

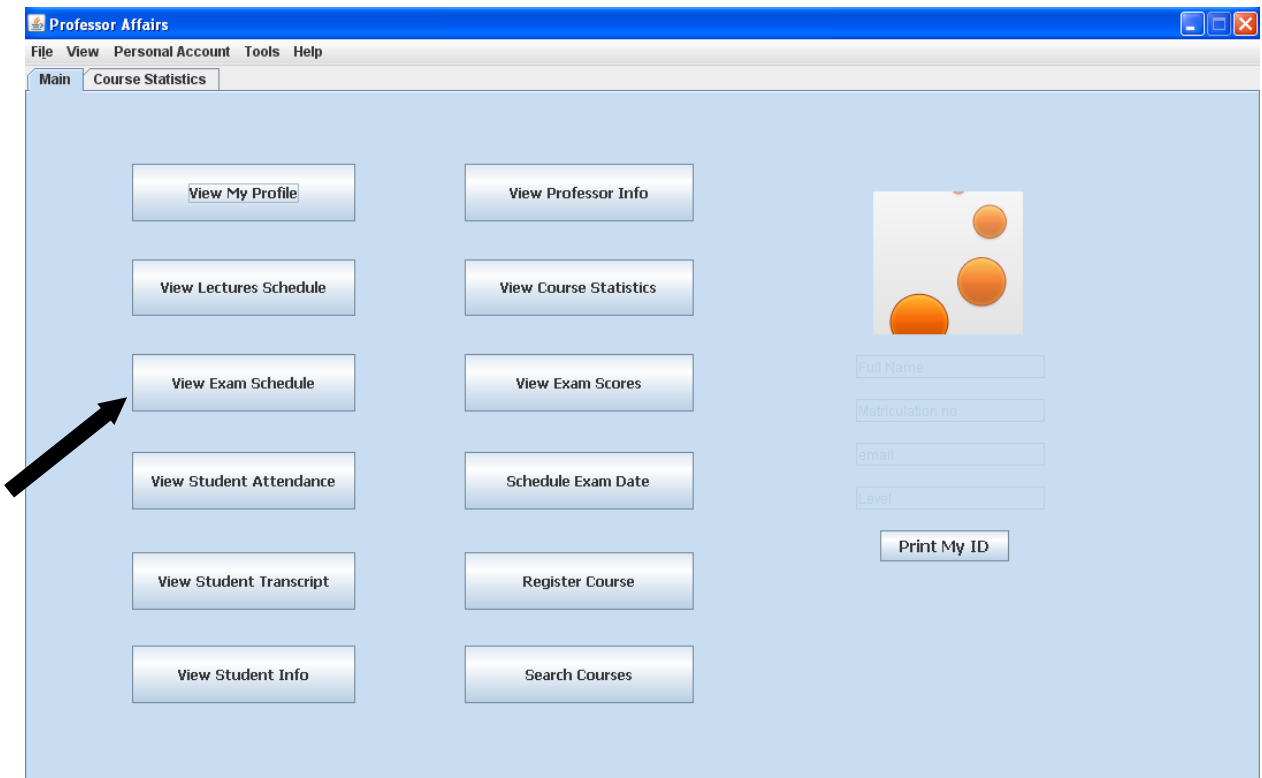
Level

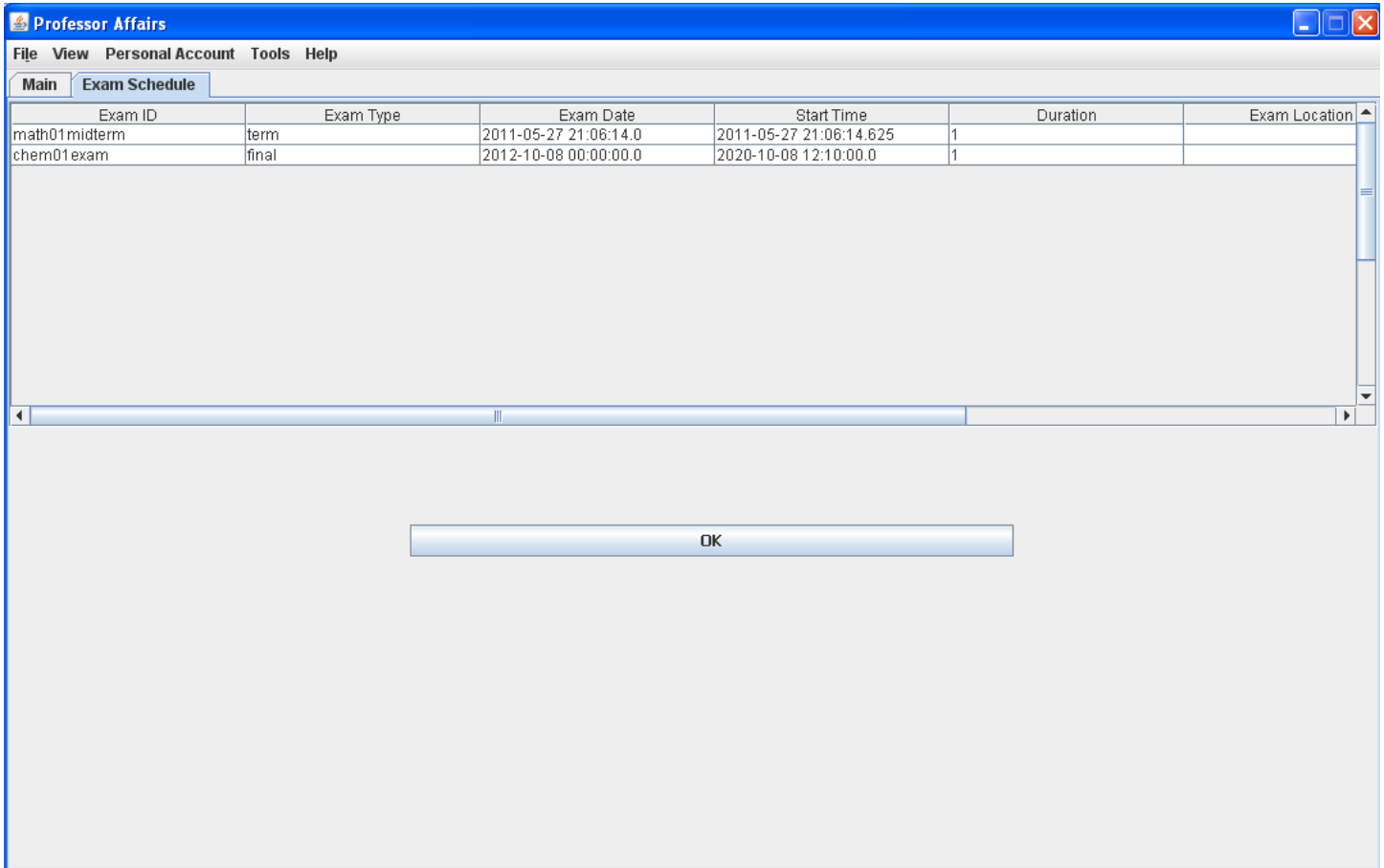
Print My ID





### 3.1.4- Viewing Exam Schedule:





### 3.2- Important Features of the Professor Module Java Code:

The features of the Professor module appear in how services reflect the operations that would be requested by a typical professor. The coordinates of these features are:

- 1- SQL code that could be translated by the database.
- 2- Java code that offers a mapping between the Java variables and the values of the columns. For example when a new professor requests to be added to the system, this operation is carried out by a services class called NewEmployee. :

```
package Services;

import java.sql.ResultSet;
import java.sql.SQLException;
import javax.swing.JOptionPane;
import session.DBManAPI;

public class NewEmployee {
```

```

        String username;
        String password;
        DBManAPI manipulate;
        String employeeID;

        public NewEmployee( DBManAPI manipulate,
String username, String password)
        {
            this.username = username;
            this.password = password;
            this.manipulate = manipulate;
        }

        public NewEmployee(DBManAPI manipulate,
String username, String password, String employeeID)
        {
            this( manipulate, username, password);
            this.employeeID = employeeID;
        }

        public void createUser() throws SQLException
        {
            try
            {
                manipulate.executeDML(String.format(
"create user %s identified by %s", username,
password));
                manipulate.executeDML(String.format(
"grant professor to %s", username));
            }
            catch( SQLException ex)
            {
                throw ex;
            }
        }

        public void removeUser() throws SQLException
        {
            try
            {
                manipulate.executeDML( String.format
("delete from dev.registration_requests where
upper(username) = upper('%s')", username));
            }
        }

```

```

        catch( SQLException ex)
        {
            throw ex;
        }
    }

    public void addHistory() throws SQLException
    {
        try
        {
            manipulate.executeDML( String.format
("insert into dev.request_archive( requesting_user,
request_time, handeled_by, handling_time)( select
'%s', request_time, user, sysdate from
dev.registration_requests where
upper(username)=upper('%s')", username, username));
        }
        catch( SQLException ex)
        {
            throw ex;
        }
    }

    public void addEmployeeToTable() throws
SQLException
    {
        try
        {
            manipulate.executeDML( String.format
( "insert into dev.employees ( first_name,
middle_name, "
                                + "last_name, nationality,
email, mobile_no, "
                                + "telephone_no, username,
street, city"
                                + ", state, country,
date_of_birth, place_of_birth, "
                                + " employee_id, image)
( select first_name, middle_name, "
                                + "last_name, nationality,
email, mobile_no, "
                                + "telephone_no, username,
addr_street, addr_city"
                                + ", addr_state,
addr_country, birth_date, "

```

```

        + "birth_place_country,
to_number('%s' ), to_number('%s'), image from
dev.registration_requests "
        + "where upper( username ) =
'%S' )", this.employeeID, username) );
        JOptionPane.showMessageDialog(null,
"The Employee Was Successfully Added to The System!",
"CONFIRMATION", JOptionPane.INFORMATION_MESSAGE);
    }

    catch( SQLException ex)
    {
        throw ex;
    }
}

void setEmployeeID(String id)
{
    employeeID = id;
}

public String getEmail()
{
    try
    {
        ResultSet email =
manipulate.executeSelect(String.format("select email
from dev.employees where username = '%s'",
username));
        email.first();
        return email.getString(1);
    }
    catch( SQLException ex )
    {
        return
JOptionPane.showInputDialog("Please Enter the Email
of This Employee:");
    }
}

public String getUsername()
{
    return username;
}

public String getPassword()
{

```

```

        return password;
    }
}

```

### 3.2.1- Feature createUser():

This feature creates a new login credentials for the professor using the username and the password already required by the user before. This is done by using the create user Oracle SQL command. The mapping between a Java string variable and a SQL literal is done by a `String.format()` static function.

### 3.2.2- Feature addEmployeeToTable():

This feature creates a new row for the professor in the employees table using the insert command. The insert command used in this feature is not a normal insert command, it uses a subquery to copy the user data from the registration table directly to the employees table. Finally after the professor is perfectly added to the employees in the system, a confirmation message will appear to the user using the following code snippet:

```

JOptionPane.showMessageDialog(null, "The Employee Was
Successfully Added to The System!", "CONFIRMATION",
JOptionPane.INFORMATION_MESSAGE);

```

### 3.2.3- Another example of the features is InitializeProfInfo class in package Services:

This class is used to retrieve all the basic information about professor from the database whenever the professor login the system, and keeps this information in the session.

```

package Services;

import java.sql.ResultSet;
import session.DBManAPI;

public class InitializeProfInfo {

    String firstName = "";
    String middleName = "";
    String lastName = "";
    DBManAPI manipulate;

    public InitializeProfInfo(DBManAPI manipulate,
String firstName, String middleName, String lastName)
{
        this.firstName = firstName;

```

```

        this.middleName = middleName;
        this.lastName = lastName;
        this.manipulate = manipulate;
    }

    public InitializeProfInfo(DBManAPI manipulate) {
        this.manipulate = manipulate;
    }

    public ResultSet getAllInfo() {
        if (firstName.equals("") ||
middleName.equals("") || lastName.equals("")) {
            return manipulate.executeSelect("select *
from dev.employees where username = user");
        } else {
            return
manipulate.executeSelect(String.format("select * from
dev.employees where first_name = '%s' and middle_name
= '%s' and last_name = '%s'", firstName, middleName,
lastName));
        }
    }

    public ResultSet getDay() {
        if (firstName.equals("") ||
middleName.equals("") || lastName.equals("")) {
            return manipulate.executeSelect("select
to_number(to_char(date_of_birth, 'DD')) from ( select
* from dev.employees where username = user)");
        } else {
            return
manipulate.executeSelect(String.format("select
to_number(to_char(date_of_birth, 'DD')) from ( %s )",
String.format("select * from dev.employees where
first_name = '%s' and middle_name = '%s' and
last_name = '%s'", firstName, middleName,
lastName)));
        }
    }

    public ResultSet getMonth() {
        if (firstName.equals("") ||
middleName.equals("") || lastName.equals("")) {
            return manipulate.executeSelect("select
to_number(to_char(date_of_birth, 'MM')) from ( select
* from dev.employees where username = user)");
        } else {

```

```

        return
        manipulate.executeSelect(String.format("select
to_number(to_char(date_of_birth, 'MM')) from ( %s)",
String.format("select * from dev.employees where
first_name = '%s' and middle_name = '%s' and
last_name = '%s'", firstName, middleName,
lastName)));
    }

}

    public ResultSet getYear() {
        if (firstName.equals("") ||
middleName.equals("") || lastName.equals("")) {
            return manipulate.executeSelect("select
to_number(to_char(date_of_birth, 'YYYY')) from
( select * from dev.employees where username =
user)");
        } else {
            return
            manipulate.executeSelect(String.format("select
to_number(to_char(date_of_birth, 'YYYY')) from
( %s)", String.format("select * from dev.employees
where first_name = '%s' and middle_name = '%s' and
last_name = '%s'", firstName, middleName,
lastName)));
        }
    }
}

```



### 3.3- Important Features of the Admin Module:

Admin module's features are the Java code that offers services that maps the different processes requested by the admin into database operations. For example, in order for the student to be enrolled in a course, an administrator must confirm this request. This operation is handled by a services class called RequestsInsertion in Services package. :

```
package Services;

import java.sql.ResultSet;
import java.sql.SQLException;
import session.DBManAPI;

public class CourseConfirmation
{
    DBManAPI manipulate;
    String username;
    String courseID;
    String courseName;

    public CourseConfirmation( DBManAPI
manipulate, String username, String courseID )
    {
        this.manipulate = manipulate;
        this.username = username;
        this.courseID = courseID;
        this.courseName = courseName();
    }

    public void changingStatus()
    {
        try
        {
            manipulate.executeDML(String.format
( "update student_courses set status = 'currently"
+ "
enrolled', num_of_enteries = num_of_enteries + 1
where student_id= ( select "
+ " s.student_id
from dev.students s where upper(s.username) = '%S' )
and "
+
"upper( course_id ) = '%S' ", username, courseID ));
        }
    }
}
```

```

        catch( SQLException ex )
        {
            ex.printStackTrace();
        }
    }

    public void addingHistory()
    {
        try
        {
            manipulate.executeDML( String.format
            ("insert into dev.request_archive( requesting_user,
            request_time, handeled_by, handling_time)( select
            '%s', request_date, user, sysdate from
            dev.course_requests where
            upper(username)=upper('%s')", username, username));
            manipulate.executeDML( String.format
            ("delete from dev.course_requests where
            upper(username) = upper('%s')", username));
        }
        catch( SQLException ex )
        {
            ex.printStackTrace();
        }
    }

    public String courseName()
    {
        ResultSet courseName =
        manipulate.executeSelect( String.format( "select
        course_name from dev.courses"
        + " where upper( course_id ) =
        %S", courseID ));
        try
        {
            courseName.first();
            return
            courseName.getString( 1 );
        }
        catch( SQLException ex )
        {
            ex.printStackTrace();
            return "";
        }
    }
}

```

```

public String getCourseName()
{
    return courseName;
}

public String getCourseID()
{
    return courseID;
}

public String getStudentEmail()
{
    ResultSet email =
manipulate.executeSelect( String.format( "select
email from students "
                                + "where upper( username ) =
%S", username) );
    try
    {
        email.first();
        return email.getString( 1 );
    }
    catch(SQLException ex)
    {
        ex.printStackTrace();
        return "";
    }
}
}

```

When the admin confirms course enrollment request, this class is instantiated. `changingStatus()` changes the status of the student regarding the specific course from 'Never Enrolled' to 'Currently Enrolled' using an update query, it also increments the number of trials by one. This feature recognizes the admin confirming the request using Oracle proprietary 'USER' function which returns the username of the user currently connected user to the database.

### 3.4- The Java Mail 1.4.4 API:

JavaMail 1.4.4 release contains several bug fixes and enhancements, including:

- Ability to cache POP3 messages on disk.
- In-memory POP3 message cache now uses soft references.
- NTLM authentication support is now integrated. SASL authentication support for SMTP.
- New demo classes showing how to handle old non-MIME Outlook messages. Note: Unless you're using Java SE 6, you will also need the JavaBeans Activation Framework (JAF) extension that provides the javax.activation package. We suggest you use version 1.1.1 of JAF, the latest release. JAF is included with Java SE 6.

The JavaMail API supports JDK 1.4 or higher.

Protocols supported:

---

This release supports the following Internet standard mail protocols:

IMAP - a message Store protocol, for reading messages from a server.

POP3 - a message Store protocol, for reading messages from a server.

SMTP - a message Transport protocol, for sending messages to a server.

The following table lists the names of the supported protocols (as used in the JavaMail API) and their capabilities:

Protocol Name	Store or Uses Transport?	Supports SSL?STARTTLS?
imap	Store	No Yes
imaps	Store	Yes Yes
pop3	Store	No No
pop3s	Store	Yes No
smtp	Transport	No Yes
smtps	Transport	Yes Yes

See our web page at <http://www.oracle.com/technetwork/java/javamail/> for the latest information on third party protocol providers.

---

## Installation

---

### Windows

---

1. Unzip the javamail1\_4\_4.zip archive. (You may have already done this.)

2. Set your CLASSPATH to include the "mail.jar" file obtained from the download, as well as the current directory. Assuming you unzipped javamail1\_4\_4.zip in c:\download the following would work:

```
set CLASSPATH=%CLASSPATH%;c:\download\javamail-1.4.4\mail.jar;
```

Also, if you're using JDK 1.5 or earlier, include the "activation.jar" file that you obtained from downloading the JavaBeans Activation Framework, in your CLASSPATH.

```
Set CLASSPATH=%CLASSPATH%;c:\download\activation\activation.jar
```

3. Go to the demo directory.

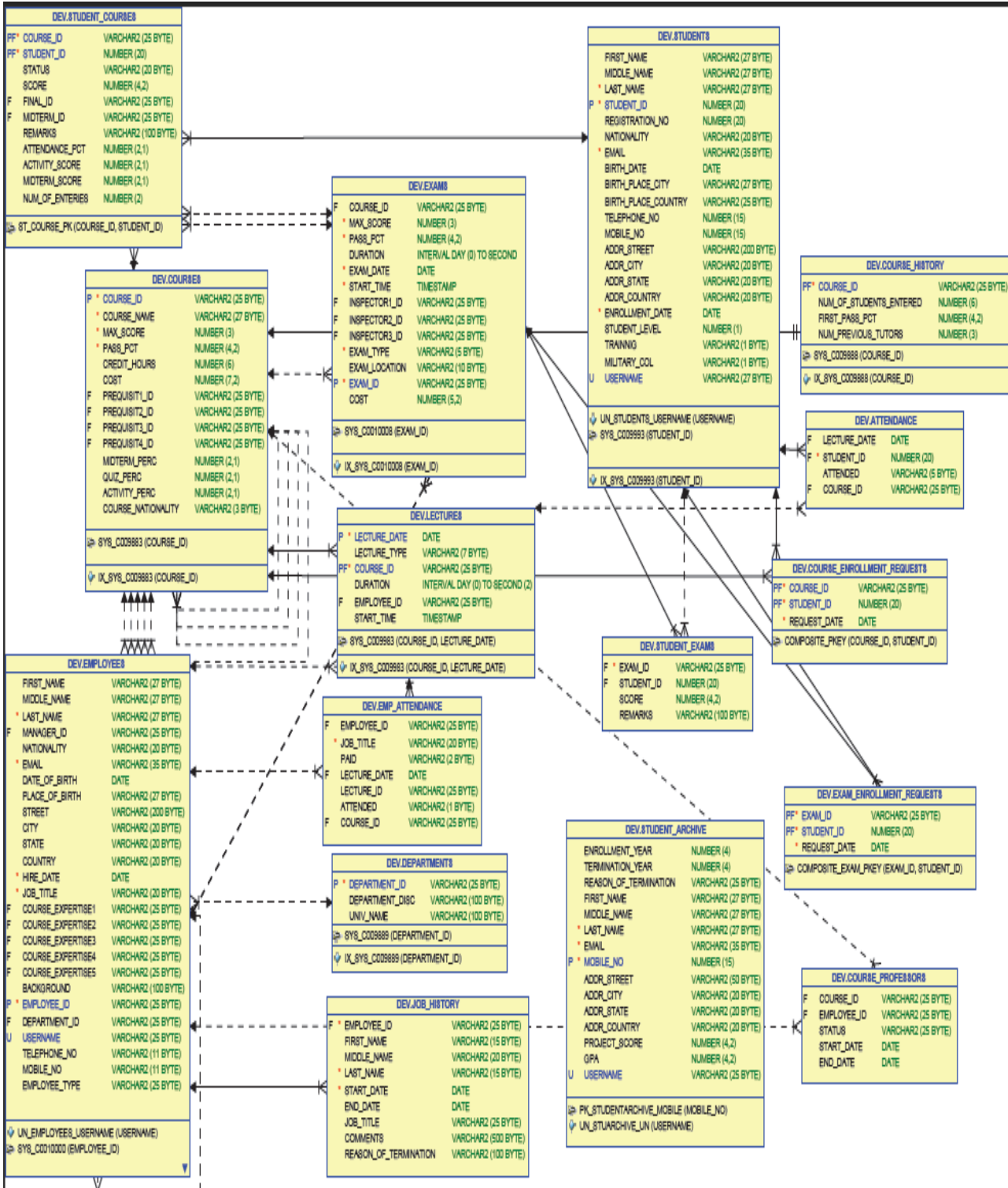
4. Compile any demo using your Java compiler. For example:  
javac msgshow.java.

5. Run the demo. The '-' option lists the required and optional command-line options to successfully run any demo. For example: java msgshow - lists the available options. And java msgshow -T imap -H <mailserver> -U <username> -P <passwd> -f INBOX 5 uses the IMAP protocol to display message number 5 from your INBOX.

(Additional instructions on how to run the simple mail reader demo and servlet demo are provided in demo/client/README.txt and demo/servlet/README.txt, respectively.)

# Chapter4 Hany Abd-EISamad Contribution

## 4.1- Creating Tables:



The ER model was studied perfectly and the plan of the ER model was created. Oracle SQL syntax was studied also some features of Oracle 10G was comprehended. Implementing the ER model requires to implicitly embed a diverse categories of statements in the tables creation scripts. In order to achieve a truly reflecting implementation of the ER model, I had to keep track of the variation of several variables including types of the columns, precision and scale of the columns, and finally the types of constraints that perfectly fit the visualization of the ER model. About 16 creation scripts were required to completely satisfy the specification comprehended from the ER model. Through this 16 scripts I changed several parameters to fit the implementation of each table independently from the others. One of these scripts is as following:

```

CREATE TABLE "DEV"."EMPLOYEES"
(
  "FIRST_NAME" VARCHAR2(27 BYTE),
  "MIDDLE_NAME" VARCHAR2(27 BYTE),
  "LAST_NAME" VARCHAR2(27 BYTE),
  "MANAGER_ID" VARCHAR2(25 BYTE),
  "NATIONALITY" VARCHAR2(20 BYTE),
  "EMAIL" VARCHAR2(35 BYTE),
  "DATE_OF_BIRTH" DATE,
  "PLACE_OF_BIRTH" VARCHAR2(27 BYTE),
  "STREET" VARCHAR2(200 BYTE),
  "CITY" VARCHAR2(20 BYTE),
  "STATE" VARCHAR2(20 BYTE),
  "COUNTRY" VARCHAR2(20 BYTE),
  "HIRE_DATE" DATE,
  "JOB_TITLE" VARCHAR2(20 BYTE),
  "COURSE_EXPERTISE1" VARCHAR2(25 BYTE),
  "COURSE_EXPERTISE2" VARCHAR2(25 BYTE),
  "COURSE_EXPERTISE3" VARCHAR2(25 BYTE),
  "COURSE_EXPERTISE4" VARCHAR2(25 BYTE),
  "COURSE_EXPERTISE5" VARCHAR2(25 BYTE),
  "BACKGROUND" VARCHAR2(100 BYTE),
  "EMPLOYEE_ID" VARCHAR2(25 BYTE),
  "DEPARTMENT_ID" VARCHAR2(25 BYTE),
  "USERNAME" VARCHAR2(25 BYTE),
  "TELEPHONE_NO" VARCHAR2(11 BYTE),
  "MOBILE_NO" VARCHAR2(11 BYTE),
  "EMPLOYEE_TYPE" VARCHAR2(25 BYTE),
  "IMAGE" BLOB
) STORAGE(INITIAL 65536 NEXT 1048576
MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT)
TABLESPACE "USERS"

```

```

      LOB ("IMAGE") STORE AS BASICFILE (
        TABLESPACE "USERS" ENABLE STORAGE IN ROW CHUNK
8192
        STORAGE (INITIAL 65536 NEXT 1048576 MINEXTENTS
1 MAXEXTENTS 2147483645
        PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT))  ENABLE ROW MOVEMENT ;

```

The previous script creates a table that models a employee (professor or tutor) user. It contains all the attributes needed to model a real professor into a tuple in the database. The most important services I have used in this creation scripts are listed in the following points:

#### 4.1.1- The CREATE TABLE statement:

The create table statement creates a table by specifying the columns and the data types of these columns and possible default values for these columns in a comma separated list after the table name. It also can specify some features about the table that could make the administrator's job easier after the comma separated list.

#### 4.1.2- The BLOB data type:

This data type is offered by the Oracle 10G database to hold a binary file. Oracle 10G offers two data types that could represent binary files they are:

##### 1- BFILE:

It holds a pointer to a file located locally on the file system of the operating system running the database. In other words, BFILEs does not hold a file itself but it holds a pointer to the file located on the operating system. Manipulating this data type over the cloud requires the use of an FTP server since in order to transfer the file between the server and the client the file would be first copied from the local operating system of the server and sent over the cloud to the client.

##### 2- Binary Large Object (BLOB):

It holds a binary file rather than just a pointer to the file. This data type can be queried easily over the cloud without using FTP server, since the file does not need to be copied from any file system firstly.

So the BLOBs are used to hold the images or avatars of the professors. This helped in manipulating those images into input streams of the I/O API of the JDK.



## 4.2- Enable Row Movement:

Three flashback techniques are based on the use of undo segments. The first flashback capability was initially introduced with release 9i of the database and has been substantially enhanced subsequently. Flashback Query (the release 9i feature) lets you query the database as it was at some time in the past, either for one select statement or by taking your session temporarily back in time so that all its queries will be against a previous version of the database. This can be used to see the state of the data before a set of transactions was committed. What did the tables look like half an hour ago? This can be invaluable in tracking down the cause of business data corruptions, and can also be used to correct some mistakes: by comparing the current and previous versions of a table, you can identify what was done that was wrong. It is even possible to select all versions of a row over a period of time, to show a history of what has happened to the row, when it happened, who did it, and the identifiers of the transactions that made each change. The third flashback technique based on undo data is Flashback Table. Having determined that inappropriate work has been committed against one table, you can instruct Oracle to reverse all changes made to that table since a particular point in time, while leaving all other tables current.<sup>1</sup> The “Enable Row Movement” statements contained in the creation of the table enables us to make use of the flashback properties offered by Oracle 11G database.

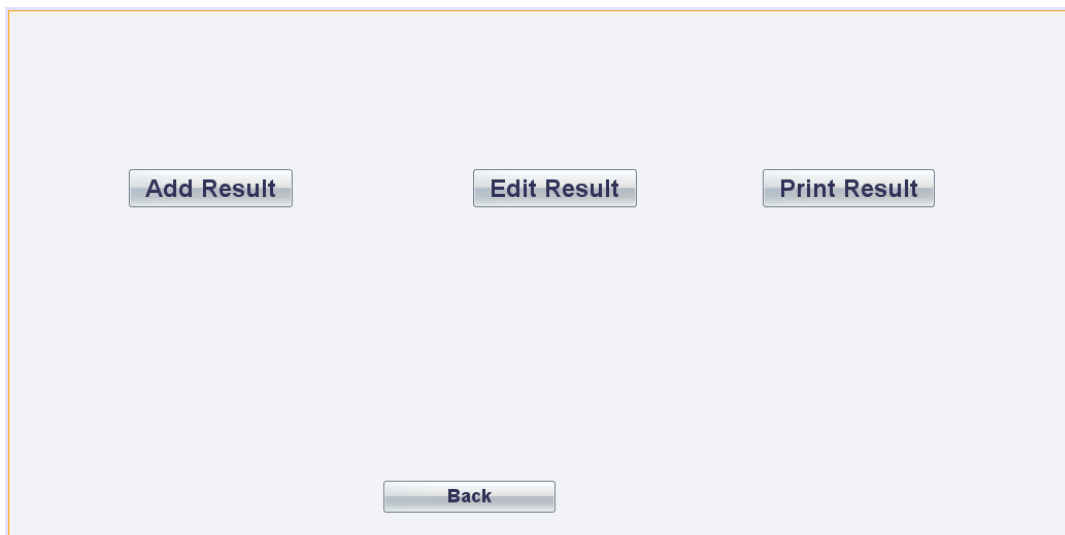
**1. Oracle Database Administration 11G Guide.**

## Chapter5 Ahmed Awad Contribution Control Role

### -The Functions:

a) The GUI Description of the control Functions:

#### 1-The main Frame:



This is the main window that is opened after the authentication is done and after that the password of the user is known to be verified to be belonged to the control department.

This window is composed of three buttons on the top and a button in the down of the window three buttons that implement the three functions that is needed from the control department this buttons are:

- \*Add Result button
- \*Edit Result button
- \*Print Result button

a) Add Result button:

This button is used to the result of a student to the database of the system.

b) Edit Result button:

This button is used to change the result of student in the system database.

c) Print Result button:

This button is used to print the script of a student or of a subject.

D) Back button:

The back button is used to return to the authentication window.

If the add button is pushed the following frame is opened which is:

### 2-Add Result Frame:

The screenshot shows a web form titled "Add Result Frame". At the top, there are four dropdown menus: "Student Name", "Course Name", "Exam Date", and "Exam Type". Each dropdown menu has "Item 1" selected. To the right of the "Course Name" dropdown is a small button with "\*\*\*\*\*" on it. To the right of the "Exam Type" dropdown is a small button with "\*\*\*\*" on it. Below the dropdowns are five text input fields, each with a label to its left: "Exam Score", "midTerm Score", "Attendance (%)", "Activity Score", and "Total Score". At the bottom of the form are two buttons: "OK" on the left and "Back" on the right.

The add Result Frame contain four combo boxes which are:

A) Student Name Combo Box:

This combo box is used to clarify the names of the students who are registered in a certain course from this combo box we choose the name of the student who his result will be edited this box is activated automatically after the choice of both the course name combo box and the exam date and type combo box.

B) Course Name Combo Box:

This combo box is used to select the name of the subject that will edit the result of its exam for the student name that exists in the combo box.

C) Exam Date and Type:

This button is activated when selecting the course name automatically to make the user select from the dates of the exams of this course and the type of the exam.

There are 5 Text Fields that are existed in the Add Result Frame which are:

A) Exam Score Field:

This field is used to take the exam score entered for that student in that subject.

B) Mid Term Score Field:

This field is used to take the mid term score of the specific student in that exam.

C) Attendance Field:

This field is used to take the attendance percentage of the student in that course.

D) Activity Score Field:

This Field is used to take the activity score of the student in this subject.

E) Total Score Field:

This Field is used to print the summation of the total of these scores.

There are two buttons in the Add Result Frame which are:

A) OK button:

This button when it is pushed it dumps all the data given in these fields into the database and prints the total score in the total score field.

B) Back button:

This button is used to back to the interface frame.

When pushing the Edit button in the interface Frame the following Frame is opened :

The screenshot shows a GUI window with the following components:

- Student Name:** A dropdown menu with 'Item 1' selected.
- Course Name:** A dropdown menu with 'Item 1' selected and a '\*\*\*\*\*' button to its right.
- Exam Date:** A dropdown menu with 'Item 1' selected.
- Exam Type:** A dropdown menu with 'Item 1' selected and a '\*\*\*\*' button to its right.
- Old Exam Score:** An input field.
- New Exam Score:** An input field.
- Old midTerm Score:** An input field.
- New midTerm Score:** An input field.
- Old Activity Score:** An input field.
- New Activity Score:** An input field.
- Old Attendance (%):** An input field.
- New Attendance (%):** An input field.
- Old Total Score:** An input field.
- New Total Score:** An input field.
- Generate:** A button located between the 'Old' and 'New' score fields.
- OK:** A button at the bottom left.
- Back:** A button at the bottom right.

This Window is composed of the following GUI components which are:

A) Four Combo boxes which are:

\* Student Name Combo Box: Is the combo box that contains the names of the students that are enrolled in a given Course name that is specified from the Course Name Combo Box and in a specific exam which has a specific date and a specific type that is chosen from the exam Date and exam type combo box .

\* Course Name Combo Box:

This is the combo box that contains the course names that are existed in the system.

\* Exam Date and Type Combo Boxes:

This Combo Box is used to contain all the exam dates of a given course and the type of the exams.

There are three buttons in the Edit Student Frame which are:

\* OK Button:

By pushing this button all the data from the new fields are dumped in the database.

\* Back Button:

This button is used to back to the interface frame.

\* Generate Button:

This button is used to retrieve the old data related to a given student that are existed in the database.

There 10 Fields in this frame which are:

1-Old Exam Score: this field contains the retrieved old exam score of a given student data that comes from the database.

2- Old midterm Score Field: this field contains the old midterm score of a given student that is retrieved from a database.

3- Old Activity Score Field: this field contains the old activity score of a given student.

4-Old Attendance Field: this field contains the old attendance of a given student.

5-Old total score Field: this field contains the Old total final score of a given student.

6-New Exam Score Field: this field contains the new exam score of a given student.

7-New midterm Score Field: this field contains the new mid term score of a given student that is retrieved from the database.

8-New Activity Score Field: this field contains the new activity score of a given student.

9-New Attendance Field: this field contains the new attendance of a given student.

10-New Total score Field: this field contains the new summation of the new scores of a given student.

When pushing the print button from the interface window the given frame is opened:



This is the Student Vs Subject Frame which consists of the following:

Three buttons which are:

a) Student Script button:

This button is used for printing a student results in the subjects that is opened in the currently semester (or term as will be described shortly).

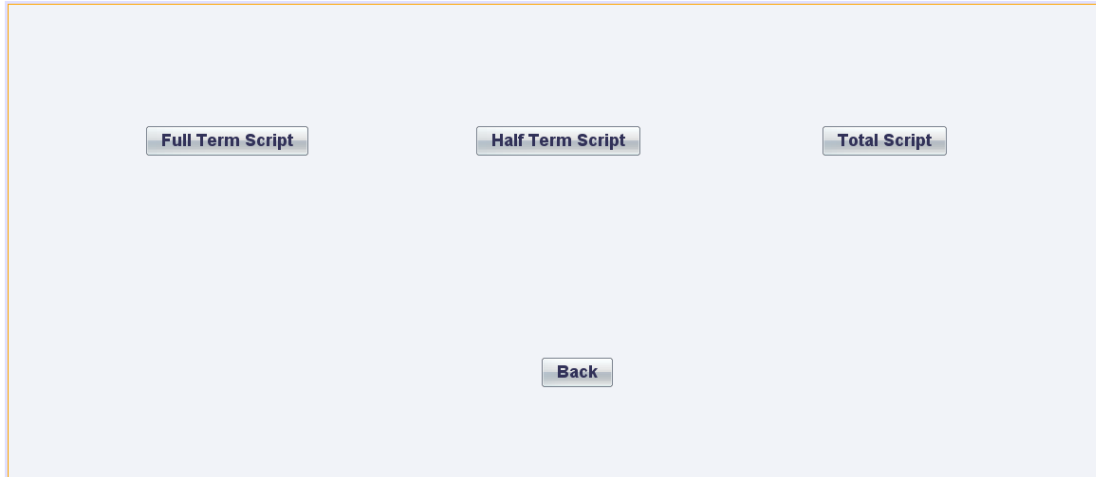
b) Subject Script button:

This button is used to print all the results of students in a given subject that is opened in the currently semester.

c) Back button:

This button is used to return to the interface window.

When pushing on the Student Script button the following window is opened:



This is the Print frame.

This frame consists of 4 buttons which are:

a) Full Term script:

This button lead to the printing of the results of a specific student in the subjects he takes in a full term (two semesters).

b) Half Term Script:

This button print the results of a specific student in the subjects he took in one semester (Half Term) .

c) Total Script:

This button prints the results of a given student in all the subjects he took.

D) Back button: this button is used to return to the Student Vs Subject frame.

If you press on the Half Semester button for example:







Subject: this header contains the subjects taken by this student.

CRH: this header contains the credit hours of every subject in the table.

ENT: this header contains the number of entrance the specific subject of a given student.

Deg: this header contains the total score of the student in each subject.

OF: this field contains the maximum score of each subject.

Grade: contains the Grade of the student in each subject.

Units: contain the units that are equivalent to each grade.

There are two buttons which are:

OK: this button is used to retrieve the data from the data base of a given student Known from its id to be visualized in the table.

Back: this button is used to return to the Print Frame.

If you Press the Total Script button the same frame will be appeared except the following little changes:

The screenshot shows a web form with the following components:

- Name:** A text input field.
- ID:** A button next to the Name field.
- Student ID:** A dropdown menu with "Item 1" selected.
- Hours Passed:** A text input field.
- Level:** A text input field.
- Total GPA:** A text input field.
- Table:** A table with 7 columns: Subjects, CRH, ENT, Deg, OF, Grade, and Units. The table has 10 rows, with the first row being the header.
- Buttons:** "OK" and "Back" buttons at the bottom of the form.

The Total script Frame consists of the following components:

Five Fields and Combo Box which are:

a) Name Field: this field takes the name of the student of this script.

b) Student Id Combo Box:

This combo box will contain the id of the students that has the name written in the name field

c) Hours Passed: this Field contains the total hours of the subjects that a given student is passed.

D) Level: this field contains the level of a given student.

E) Total GPA: this Field contains the total GPA of a given student.

The frame contains the table which shows the results of subjects taken Passed by a given student and has the following headers:

Subject: this header contains the subjects taken by this student.

CRH: this header contains the credit hours of every subject in the table.

ENT: this header contains the number of entrance the specific subject of a given student.

Deg: this header contains the total score of the student in each subject.

OF: this field contains the maximum score of each subject.

Grade: contains the Grade of the student in each subject.

Units: contain the units that are equivalent to each grade.

There are two buttons which are:

OK: this button is used to retrieve the data from the data base of a given student Known from its id to be visualized in the table.

Back: this button is used to return to the Print Frame.

If you press on the Subject Script Button on the Student Vs Subject Frame



The following frame is opened:

Student Name	Mid Term Score	Attendance	Activity Total Score	Exam Score	Total Score	Max Score	Grade

This is the subject frame and contains the following components:

One Combo Box which is:

Course Name Combo Box: which contain the name of all courses in the system.

Exam Date Combo Box: which contain the Dates of all exams specific to a given course name in the system.

Course Id Field: which contain the id number of the course being selected.

Two buttons which are:

- OK: this button supply the data from the data base to the table.
- Back: this button is used to return to the Subject Vs Student frame.
- Activation Button: this button is used when we want to activate the choose of the students that have an exam in a given subject for a specific date.
- Choose Button: this button is used when we want to choose of the students that have an exam in a given subject for a specific date.
- Course Id Field: this field contain the id of the course.

The Table that is used to show all the students that enter the test of this subject and their results for the following exam date.

Its header is :

Student Name: which contain the name of the students that have exam in this course(and in a specific date).

midterm Score: the score of the mid term score of each student in this course.  
Attendance: contain the attendance of each student in the course.  
Activity Score: contain the activity score of each student in the course.  
Exam Score: contain the exam score of each student in the course.  
Total Score: contain the summation of this scores of each student in the course.

## -Features:


1)

The screenshot shows a web form with four main sections: 'Student Name' with a dropdown menu, 'Course Name' with a dropdown menu and a search button (\*\*\*\*\*), 'Exam Date' with a dropdown menu, and 'Exam Type' with a dropdown menu and a search button (\*\*\*\*).

To prevent the entrance of a fault value or when the person forget the student name or the course name so we solve this problem by:  
Making a combo box for the courses names that contain the name of all courses in the system.  
That load automatically when starting the class.

Its Code is:

```
db = new DatabaseConnection();
    db.openDatabaseConnection();
    String st = String.format("select course_name from
dev.courses ");
    try {
        ResultSet rs =
db.con.createStatement().executeQuery(st);
        int counter = 0;
        while (rs.next()) {
            counter++;
        }
        rs = db.con.createStatement().executeQuery(st);
        cmbCoursesSelect.removeAllItems();
        for (int i = 0; i < counter; i++) {
            rs.next();
            cmbCoursesSelect.addItem(rs.getString(1));
        }
    } catch (SQLException ex) {
        System.err.println(ex.getMessage() + "in " +
this.getClass().getName());
        ex.printStackTrace();
    }
}
When a course name is selected and push the button
```

 which show all the dates of exams and the type of the exams of a specific course its code is:

```

String examId2=null;
    db = new DatabaseConnection();
    db.openDatabaseConnection();
    String courseSelected =
cmbCoursesSelect.getSelectedItemAt().toString();
    cmbDateSelection.removeAllItems();
    cmbExamTypeSelection.removeAllItems();

    try {

        String st = String.format("select course_id "
            + "from dev.courses where
course_name='%s' ", courseSelected);
        ResultSet rs =
db.con.createStatement().executeQuery(st);

        if (rs.next()) {
            //rs.next();
            courseId=rs.getString(1);
            String courseId2 = rs.getString(1);
            String st1 = String.format("select exam_id
from exams "
                + "where course_id='%s'",
courseId2);
            ResultSet rs1 =
db.con.createStatement().executeQuery(st1);
            while( rs1.next()){
                examId2 = rs1.getString(1);

                String st2 = String.format("select
to_char(exam_date,'DD-MM-YY') \"exam_date\",exam_type "
                    + "from dev.exams "
                    + "where exam_id='%s'", examId2);

                ResultSet rs2 =
db.con.createStatement().executeQuery(st2);
                while (rs2.next()) {
                    String exam_date =
rs2.getString("exam_date");
                    String exam_type =
rs2.getString("exam_type");
                    cmbDateSelection.addItem(exam_date);


```

```

        cmbExamTypeSelection.addItem(exam_type);

    }
}
}
} catch (SQLException ex) {
    System.err.println(ex.getMessage() + "in " +
this.getClass().getName());
    ex.printStackTrace();
}
}

```

When you select an exam date and push the button  the names of the students that submit in the exam of the course in this date are loaded in a combo box to enable the user to select among them freely, its code is:

```

Integer studentId2=0;
String
examType=cmbExamTypeSelection.getSelectedItem().toString();
String
examDate=cmbDateSelection.getSelectedItem().toString();
    String st = String.format("select student_id "
        + "from dev.student_exams "
        + "where exam_id=select exam_id from
dev.exams where exam_type='%s' AND exam_date='%s'
",examType,examDate);
    String examIdSelection=String.format("select
exam_id from dev.exams where exam_type='%s' AND
exam_date='%s' "
        + " AND
course_id='%s'",examType,examDate,courseId);
    cmbStudentName.removeAllItems();
    try {
        ResultSet rs =
db.con.createStatement().executeQuery(st);
        while(rs.next()){
            studentId2 = rs.getInt("student_id");
            String st1 = String.format("select
first_name,middle_name,last_name "
                + "from dev.students "
                + "where student_id='%d'", studentId2);
            ResultSet rs1 =
db.con.createStatement().executeQuery(st1);
            while (rs1.next()) {
                String name = String.format("%s %s %s",
rs1.getString("first_name"),

```

```

        rs1.getString("middle_name"),
rs1.getString("last_name"));
        cmbStudentName.addItem(name);
    }
}
ResultSet
rs2=db.con.createStatement().executeQuery(examIdSelection);
rs2.next();
examId=rs2.getString(1);
}
catch (SQLException ex) {
    System.err.println(ex.getMessage() + "in " +
this.getClass().getName());
    ex.printStackTrace();
}

```

The screenshot shows a light blue background with five rows of text labels and input fields. The labels are 'Old Exam Score', 'Old midTerm Score', 'Old Activity Score', 'Old Attendance', and 'Old Total Score'. Each label is followed by a white rectangular input field with a thin border. To the right of the 'Old Activity Score' input field is a button with a blue gradient and the word 'Generate' in white text.

2)

The generate button is used to provide the person with the old scores of given student before adding the new scores.

```

Double oldTotalScore = 0.0;
    Double oldExamScore = 0.0;
    Double oldMidTermScore = 0.0;
    Double oldAttendanceScore = 0.0;
    Double oldActivityScore = 0.0;

    db = new DatabaseConnection();
    db.openDatabaseConnection();

    try {
        String generation = String.format("select
score,attendance_pct,activity_score "
        + "from dev.student_courses where
student_id='%d' AND course_id='%s'", studentId, courseId);

```



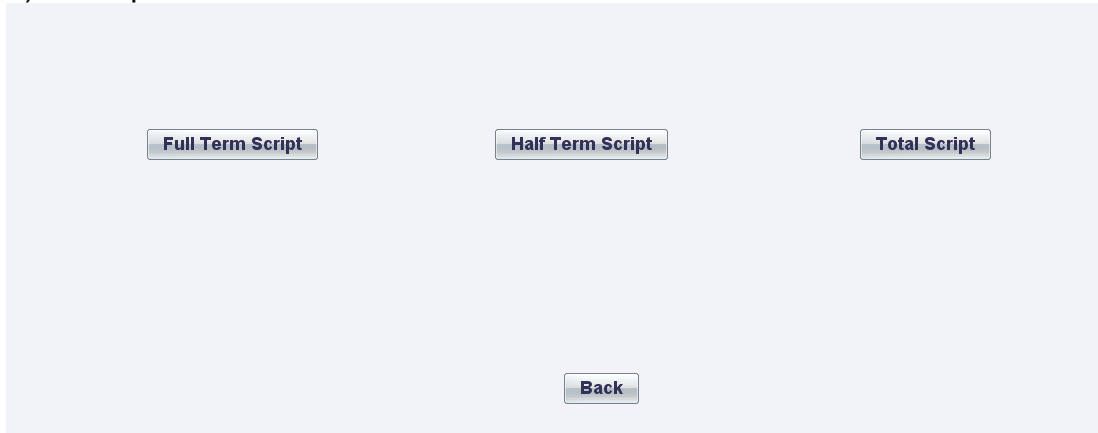
```

        String midTermGen=String.format("select score
from dev.student_exams where student_exams.student_id='%d'
AND "
        + "exam_id="
        + "select midterm_id from
dev.student_courses where "
        + "student_courses.student_id='%d' AND
student_courses.course_id='%s'",studentId,studentId,courseI
d);
        String ExamGen=String.format("select score from
dev.student_exams where student_exams.student_id='%d' AND "
        + "exam_id="
        + "select final_id from
dev.student_courses where "
        + "student_courses.student_id='%d' AND
student_courses.course_id='%s'",studentId,studentId,courseI
d);
        ResultSet rs =
db.con.createStatement().executeQuery(generation);
        ResultSet rs2 =
db.con.createStatement().executeQuery(midTermGen);
        ResultSet rs3 =
db.con.createStatement().executeQuery(ExamGen);
        if ((rs.next()) && (rs2.next()) &&
(rs3.next())) {
                oldExamScore = rs3.getDouble("score");
                oldMidTermScore = rs2.getDouble("score");
                oldAttendanceScore =
rs.getDouble("attendance_pct");
                oldActivityScore =
rs.getDouble("activity_score");
                oldTotalScore = oldExamScore +
oldMidTermScore + oldAttendanceScore + oldActivityScore;
                oldTotalScoreField.setText(oldTotalScore.to
String());
                oldExamScoreField.setText(oldExamScore.toSt
ring());
                oldMidTermField.setText(oldMidTermScore.toS
tring());
                oldAttendanceField.setText(oldAttendanceSco
re.toString());
                oldActivityScoreField.setText(oldActivitySc
ore.toString());
        }
    } catch (SQLException ex) {

```

```
        System.err.println(ex.getMessage() + "in " +
this.getClass().getName());
        ex.printStackTrace();
        this.dispose();
    }
}
```

3) in the print frame:



When you select half term script button or full term script button the same is opened except the label which is called semester hours must change into term hours

A screenshot of a student record form. At the top, there are fields for 'Name', 'ID', 'Student ID', and a dropdown menu labeled 'Item 1'. Below these are fields for 'Hours Passed', 'Level', 'Semester Hours', and 'Total GPA'. At the bottom, there is a table with columns: 'Subjects', 'CRH', 'ENT', 'Deg', 'OF', 'Grade', and 'Units'. The table has multiple empty rows. There are also some partially visible buttons at the very bottom of the form.



**Name**

The name of the student is entered in the field and by pushing the ID button it comparing all the names existed in the system to the entered name to be sure of the validation of this name if it exist it shows its id in the combo box

**Student ID**

If there is a repetition of this name it show all the ids that related to this repeated name.

If the name isn't exist it shows a warning message telling the user to reenter the name

Its code is:

```
cmbStudentId.removeAllItems();
    String studentName1 = nameField.getText();
    int studentId2=0;
    int counter = 0;
    String firstName = null;
    String middleName = null;
    String lastName = null;
    String reservedFirstName = null;
    String reservedMiddleName = null;
    String reservedLastName = null;
    try {
        String studentName = String.format("select
first_name,middle_name,last_name "
        + "from dev.students");
        ResultSet rs =
db.con.createStatement().executeQuery(studentName);
        while (rs.next()) {
            firstName = rs.getString("first_name");
            middleName = rs.getString("middle_name");
            lastName = rs.getString("last_name");
            String fullStudentName = String.format("%s
%s %s", rs.getString("first_name"),
rs.getString("middle_name"), rs.getString("last_name"));
            if
(studentName1.equalsIgnoreCase(fullStudentName)) {
                counter++;
                reservedFirstName = firstName;
                reservedMiddleName = middleName;
```

```

        reservedLastName = lastName;
    }
}
if (counter == 1) {
    String st2 = String.format("select
student_id "
        + "from dev.students "
        + "where first_name='%s' AND
middle_name='%s' AND last_name='%s'",
        reservedFirstName,
reservedMiddleName, reservedLastName);
    nameField.setEditable(false);
    ResultSet rs2 =
db.con.createStatement().executeQuery(st2);
    if (rs2.next()) {
        studentId2 = rs2.getInt("student_id");
        cmbStudentId.addItem(rs2.getInt("studen
t_id"));
        cmbStudentId.setEditable(false);
    }
}
if (counter >= 2) {
    JOptionPane.showMessageDialog(null, "the
there are a duplication in the name please choose an ID
from"
        + "the combo box", "Warning"
        + "", JOptionPane.PLAIN_MESSAGE);
    String st2 = String.format("select
student_id "
        + "from dev.students "
        + "where first_name='%s' AND
middle_name='%s' AND last_name='%s'",
        reservedFirstName,
reservedMiddleName, reservedLastName);
    ResultSet rs2 =
db.con.createStatement().executeQuery(st2);
    while (rs2.next()) {
        studentId2 = rs2.getInt("student_id");
        cmbStudentId.addItem(rs2.getInt("studen
t_id"));
    }
}
if (counter == 0) {
    JOptionPane.showMessageDialog(null, "Sorry
there isn't any student with this name please retype the
name again", "Warning"

```

```

        + "", JOptionPane.PLAIN_MESSAGE);
    }

    } catch (SQLException ex) {
        System.err.println(ex.getMessage() + "in " +
this.getClass().getName());
    }
}

```

so by this way we be sure that the user doesn't enter an invalid name or student id.

5)

In the exam frame :

Student Name	Mid Term Score	Attendance	Activity Total Score	Exam Score	Total Score	Max. Score	Grade

The course name is loaded in the combo box at the beginning of the program so as to avoid any probable mistake of entering invalid course name.

The code of loading is:

```


 initComponents();
    db = new DatabaseConnection();
    db.openDatabaseConnection();
    String st = String.format("select course_name from
dev.courses");
    try {
        ResultSet rs =
db.con.createStatement().executeQuery(st);
        int counter = 0;
        while (rs.next()) {
            counter++;
        }
    }
}

```


```

        rs = db.con.createStatement().executeQuery(st);
        cmbCourseName.removeAllItems();
        for (int i = 0; i < counter; i++) {
            rs.next();
            cmbCourseName.addItem(rs.getString(1));
        }
    } catch (SQLException ex) {
        System.err.println(ex.getMessage() + "in " +
this.getClass().getName());
    }
}

```

After choosing the course and pushing the button  the course id equivalent to the course name will appear in the course id field

**Course ID**

If you push  button the exam date combo box is activated

**Exam Date**

this combo box is used when you want to choose the students who entered the exam of this course in a chosen date the activate button code is:

```

cmbExamDate.setEditable(true);
cmbExamDate.removeAllItems();
try {
    String st = String.format("select exam_date
from dev.exams where course_id='%s'", courseId);
    ResultSet rs =
db.con.createStatement().executeQuery(st);
    while (rs.next()) {
        cmbExamDate.addItem(rs.getString("exam_date
"));
    }
} catch (SQLException ex) {
    System.err.println(ex.getMessage() + "in " +
this.getClass().getName());
}
}

```

After chosen a specific date push the choose button to select this date and choose the students according to this date:

```
String date = cmbExamDate.getSelectedItem().toString();
```

```

String examId2 = null;
Integer studentId = 0;
String name = null;
Double examScore = 0.0;
Double midTermScore = 0.0;
Double activity = 0.0;
Double Deg = 0.0;
Double attendance = 0.0;
String grade = null;
String tableHeader[] = {"Student Name", "Mid Term
Score", "Attendance", "Activity Total Score", "Exam Score",
"Total Score", "Max Score",
    "Grade"};
    tableModel = new DefaultTableModel(null,
tableHeader);
    table = new JTable(tableModel);
    try {

        String st = String.format("select student_id "
            + "from dev.student_courses "
            + "where course_id='%s' AND"
            + " upper(status)='%s' ", courseId,
"CURRENTLY ENROLLED");
        ResultSet rs =
db.con.createStatement().executeQuery(st);
        while (rs.next()) {
            studentId = rs.getInt("student_id");
            String st1 = String.format("select
first_name,middle_name,last_name from dev.students where
student_id='%d'", studentId);
            ResultSet rs2 =
db.con.createStatement().executeQuery(st1);
            if (rs2.next()) {
                name = String.format("%s %s %s",
rs2.getString("first_name"), rs2.getString("middle_name"),
rs2.getString("last_name"));
            }
            String st2 = String.format("select
attendance_pct,"
                + "activity_score"
                + "from dev.student_courses"
                + "where course_id='%s' AND
student_id='%d'", courseId, studentId);
            String midTerm=String.format("select score
from dev.student_exams where exam_id="

```



```

        + "select midterm_id from
dev.student_courses where course_id='%s' AND
student_id='%d' AND exam_date='%s'",
        courseId,studentId,date);
        String exam=String.format("select score
from dev.student_exams where exam_id="
        + "select final_id from
dev.student_courses where course_id='%s' AND
student_id='%d' AND exam_date='%s'",
        courseId,studentId,date);
        ResultSet rs1 =
db.con.createStatement().executeQuery(st2);
        ResultSet rs11 =
db.con.createStatement().executeQuery(midTerm);
        ResultSet rs21 =
db.con.createStatement().executeQuery(exam);
        if (rs1.next() && (rs11.next()) &&
(rs21.next())) {

                midtermScore =
rs11.getDouble("midterm_id");
                attendance =
rs1.getDouble("attendance_pct") * 10;
                activity =
rs1.getDouble("activity_score");
                examScore = rs21.getDouble("score");
                String st3 = String.format("select
score from dev.student_exams where student_id='%d'",
studentId);

                ResultSet rs7 =
db.con.createStatement().executeQuery(st3);
                if (rs7.next());
                Deg = rs7.getDouble("score");

                if (Deg < 50) {
                        grade = "F";
                }
                if ((50 <= Deg) && (Deg < 55)) {
                        grade = "D";
                }
                if ((55 <= Deg) && (Deg < 60)) {
                        grade = "D+";
                }
                if ((60 <= Deg) && (Deg < 65)) {
                        grade = "C-";
                }
}

```

```

        if ((65 <= Deg) && (Deg < 70)) {
            grade = "C";
        }
        if ((70 <= Deg) && (Deg < 75)) {
            grade = "C+";
        }
        if ((75 <= Deg) && (Deg < 80)) {
            grade = "B-";
        }
        if ((80 <= Deg) && (Deg < 85)) {
            grade = "B";
        }
        if ((85 <= Deg) && (Deg < 90)) {
            grade = "B+";
        }
        if ((90 <= Deg) && (Deg < 95)) {
            grade = "A-";
        }
        if ((95 <= Deg) && (Deg < 100)) {
            grade = "A";
        }
    }
    tableModel.addRow(new Object[]{name,
midTermScore, attendance, activity, examScore, Deg, 100,
grade});
    }
    jScrollPane1.setViewportViewView(table);
} catch (SQLException ex) {
    System.err.println(ex.getMessage() + "in " +
this.getClass().getName());
}
}

```

## **Chapter 6**

### **Ahmed Saad Contribution**

### **Graduate Module**

In the coding of the Graduate module the Java Persistence Query Language is used.

#### **JPQL**

The Java Persistence Query Language (JPQL) is a platform-independent object-oriented query language defined as part of the Java Persistence API specification.

JPQL is used to make queries against entities stored in a relational database. It is heavily inspired by SQL, and its queries resemble SQL queries in syntax, but operate against JPA entity objects rather than directly with database tables. In addition to retrieving objects (SELECT queries), JPQL supports bulk UPDATE and DELETE queries.

#### **Role of the Graduate**

The Graduate can view some data in Tables from the database & can edit his contact information.

The Graduate can view the archive and can Edit some data in his archive by using the following buttons (new, delete, refresh, save) in order to be able to update his contact information which facilitates the communication with his colleagues.

## A SNAPSHOT OF THE GRADUATE ARCHIVE

The screenshot shows a window titled 'Archive' with a menu bar containing 'File' and 'Help'. Below the menu bar is a table with the following columns: Mobile No, Enrollment ..., Termination..., Reason Of ..., First Name, Middle Name, Last Name, Email, Addr Street, Addr City, Addr State, Addr Country, Proj..., Gpa, and Us... The table contains two rows of data:

Mobile No	Enrollment ...	Termination...	Reason Of ...	First Name	Middle Name	Last Name	Email	Addr Street	Addr City	Addr State	Addr Country	Proj...	Gpa	Us...
122816189				Ahmed		Hisham	batman_danger@hotmail.com	Eldaher	Cairo		Egypt			
106761722				Ramy		saad	ramysaadm@hotmail.com	Nasr City	Cairo		Egypt			

## A SNAPSHOT OF THE GRADUATE ARCHIVE WHICH ALLOWS THE EDITING OF THE CONTACT INFORMATION

The screenshot shows a window titled 'Edit Archive' with a menu bar containing 'File' and 'Help'. Below the menu bar is a table with the following columns: First Name, Middle Name, Last Name, Email, Mobile No, Addr Street, Addr City, Addr State, Addr Country, and Username. The table contains two rows of data:

First Name	Middle Name	Last Name	Email	Mobile No	Addr Street	Addr City	Addr State	Addr Country	Username
Ahmed	Hisham	Saad	batman_danger@hotmail.com	106761722	Eldaher	Cairo		Egypt	ahmed
Ramy	Saad	Mohareb	ramysaadm@hotmail.com	122816189	Nasr City	Cairo		Egypt	ramy

Below the table is a form for editing contact information. The form contains the following fields:

- Enrollment Year:
- Termination Year:
- Reason Of Termination:
- Middle Name:
- Last Name:
- Email:
- Mobile No:
- Addr Street:
- Addr City:
- Addr State:
- Addr Country:
- Project Score:
- Gpa:
- Username:

At the bottom right of the form are four buttons: 'New', 'Delete', 'Refresh', and 'Save'.

The Graduate can view the list of professors and courses in order to be updated with the new trends in his technology researches.

## A SNAPSHOT OF THE LIST OF PROFESSORS

Emp...	Firs...	Mid...	Last...	Nati...	Email	Dat...	Plac...	Street	City	State	Cou...	Hire...	Job ...	Cou...	Cou...	Cou...	Cou...	Cou...	Bac...	Dep...	Use...	Tele...	Mob...	Emp...	Man...
ramy01	Ramy	Saad	Mohareb	Egyptian	ramys...	Oct 28...	Egypt	4b abr...	Cairo		Egypt	May 2...	Profes...	math01	chem01					PROF	02226...	01228...			
amro01	Amro	Salem	Hassan	egyptian	aliens3...	Dec 18...	usa	asffsa	cairo		egypt	Jun 3, ...	Profes...	chem01	chem01	math01				DEV	01666...	02222...	Professor	desкто...	

## A SNAPSHOT OF THE LIST OF COURSES

Course Id	Course Name	Max Score	Pass Pct	Credit Hours	Cost	Midterm Perc	Quiz Perc	Activity Perc	Course Natio...	Prerequisite2 Id	Prerequisite3 Id	Prerequisite4 Id	Prerequisite1 Id
chem01	chemistry	30	18										
math01	mathematics	30	70										

## FOR THE STUDENT ARCHIVE (VIEW WITHOUT EIDTING)

### The @NamedQueries:

→For example the following @NamedQuery annotation defines a query whose name is " StudentArchive.findAll" that retrieves all the StudentArchive objects in the database

```
[@NamedQuery (name = "StudentArchive.findAll", query =  
"SELECT s FROM StudentArchive s"),].
```

→Every @NamedQuery annotation is attached to exactly one entity class and attaching multiple named queries to the same entity class requires wrapping them in a @NamedQueries annotation.

→In the following select queries i use the path expressions, such as s.enrollment Year, which referred to as projection. The field values are extracted from (or projected out of) entity objects to form the query results.

→For example here in the below queries s.gpa where s represents a Student Archive entity object uses the GPA persistent field in the Student Archive class to navigate to the associated GPA entity object.

### The @NamedQueries are

```
@Entity  
@Table (name = "STUDENT_ARCHIVE", catalog = "", schema =  
"DEV")  
@NamedQueries({ @NamedQuery(name =  
"StudentArchive.findAll", query = "SELECT s FROM  
StudentArchive s"),  
@NamedQuery(name = "StudentArchive.findByEnrollmentYear",  
query = "SELECT s FROM StudentArchive s WHERE  
s.enrollmentYear = :enrollmentYear"),  
@NamedQuery(name = "StudentArchive.findByTerminationYear",  
query = "SELECT s FROM StudentArchive s WHERE  
s.terminationYear = :terminationYear"),
```

```

@NamedQuery(name =
"StudentArchive.findByReasonOfTermination", query = "SELECT
s FROM StudentArchive s WHERE s.reasonOfTermination =
:reasonOfTermination"),
@NamedQuery(name = "StudentArchive.findByFirstName", query =
"SELECT s FROM StudentArchive s WHERE s.firstName =
:firstName"),
@NamedQuery(name = "StudentArchive.findByMiddleName", query =
"SELECT s FROM StudentArchive s WHERE s.middleName =
:middleName"),
@NamedQuery(name = "StudentArchive.findByLastName", query =
"SELECT s FROM StudentArchive s WHERE s.lastName =
:lastName"),
@NamedQuery(name = "StudentArchive.findByEmail", query =
"SELECT s FROM StudentArchive s WHERE s.email = :email"),
@NamedQuery(name = "StudentArchive.findByMobileNo", query =
"SELECT s FROM StudentArchive s WHERE s.mobileNo =
:mobileNo"),
@NamedQuery(name = "StudentArchive.findByAddrStreet", query =
"SELECT s FROM StudentArchive s WHERE s.addrStreet =
:addrStreet"),
@NamedQuery(name = "StudentArchive.findByAddrCity", query =
"SELECT s FROM StudentArchive s WHERE s.addrCity =
:addrCity"),
@NamedQuery(name = "StudentArchive.findByAddrState", query =
"SELECT s FROM StudentArchive s WHERE s.addrState =
:addrState"),
@NamedQuery(name = "StudentArchive.findByAddrCountry",
query = "SELECT s FROM StudentArchive s WHERE s.addrCountry
= :addrCountry"),
@NamedQuery(name = "StudentArchive.findByProjectScore",
query = "SELECT s FROM StudentArchive s WHERE
s.projectScore = :projectScore"),
@NamedQuery(name = "StudentArchive.findByGpa", query =
"SELECT s FROM StudentArchive s WHERE s.gpa = :gpa")) .

```

**The annotations are defined as follows**

```

@Column (name = "ENROLLMENT_YEAR")
private Short enrollmentYear;
@Column(name = "TERMINATION_YEAR")
private Short terminationYear;
@Column(name = "REASON_OF_TERMINATION")
private String reasonOfTermination;
@Column(name = "FIRST_NAME")
private String firstName;
@Column(name = "MIDDLE_NAME")
private String middleName;
@Basic(optional = false)

```

```

@Column(name = "LAST_NAME")
private String lastName;
@Basic(optional = false)
@Column(name = "EMAIL")
private String email;
@Id
@Basic(optional = false)
@Column(name = "MOBILE_NO")
private Long mobileNo;
@Column(name = "ADDR_STREET")
private String addrStreet;
@Column(name = "ADDR_CITY")
private String addrCity;
    @Column(name = "ADDR_STATE")
    Private String addrState;
    @Column(name = "ADDR_COUNTRY")
    private String addrCountry;
    @Column(name = "PROJECT_SCORE")
    private BigDecimal projectScore;
    @Column(name = "GPA")
    private BigDecimal gpa;

```

## WITH GETTER AND SETTER METHODS AS FOLLOWS

```

public StudentArchive() {
}
    public StudentArchive(Long mobileNo) {
        this.mobileNo = mobileNo;
    }
    public StudentArchive(Long mobileNo, String lastName,
String email) {
        this.mobileNo = mobileNo;
        this.lastName = lastName;
        this.email = email;
    }
    public Short getEnrollmentYear() {
return enrollmentYear;
    }
    public void setEnrollmentYear(Short enrollmentYear) {
        Short oldEnrollmentYear = this.enrollmentYear;
        this.enrollmentYear = enrollmentYear;
        changeSupport.firePropertyChange("enrollmentYear",
oldEnrollmentYear, enrollmentYear);
    }
    public Short getTerminationYear() {
        return terminationYear;
    }
}

```



```

        public void setTerminationYear(Short terminationYear) {
            Short oldTerminationYear = this.terminationYear;
            this.terminationYear = terminationYear;
            changeSupport.firePropertyChange("terminationYear",
            oldTerminationYear, terminationYear);
        }
        public String getReasonOfTermination() {
            return reasonOfTermination;
        }
        public void setReasonOfTermination(String
            reasonOfTermination) {
            String oldReasonOfTermination = this.reasonOfTermination;
            this.reasonOfTermination = reasonOfTermination;
            changeSupport.firePropertyChange("reasonOfTermination",
            oldReasonOfTermination, reasonOfTermination);
        }
        public String getFirstName() {
            return firstName;
        }
        public void setFirstName(String firstName) {
            String oldFirstName = this.firstName;
            this.firstName = firstName;
            changeSupport.firePropertyChange("firstName",
            oldFirstName, firstName);
        }
        public String getMiddleName() {
            return middleName;
        }
        public void setMiddleName(String middleName) {
            String oldMiddleName = this.middleName;
            this.middleName = middleName;
            changeSupport.firePropertyChange("middleName",
            oldMiddleName, middleName);
        }
        public String getLastName() {
            return lastName;
        }
        public void setLastName(String lastName) {
            String oldLastName = this.lastName;
            this.lastName = lastName;
            changeSupport.firePropertyChange("lastName",
            oldLastName, lastName);
        }
        public String getEmail() {
            return email;
        }

```

```

    }
    public void setEmail(String email) {
        String oldEmail = this.email;
        this.email = email;
        changeSupport.firePropertyChange("email", oldEmail,
email);
    }
    public Long getMobileNo() {
        return mobileNo;
    }
    public void setMobileNo(Long mobileNo) {
        Long oldMobileNo = this.mobileNo;
        this.mobileNo = mobileNo;
        changeSupport.firePropertyChange("mobileNo",
oldMobileNo, mobileNo);
    }
    public String getAddrStreet() {
        return addrStreet;
    }
    public void setAddrStreet(String addrStreet) {
        String oldAddrStreet = this.addrStreet;
        this.addrStreet = addrStreet;
        changeSupport.firePropertyChange("addrStreet",
oldAddrStreet, addrStreet);
    }
    public String getAddrCity() {
        return addrCity;
    }
    public void setAddrCity(String addrCity) {
        String oldAddrCity = this.addrCity;
        this.addrCity = addrCity;
        changeSupport.firePropertyChange("addrCity",
oldAddrCity, addrCity);
    }
    public String getAddrState() {
        return addrState;
    }
    public void setAddrState(String addrState) {
        String oldAddrState = this.addrState;
        this.addrState = addrState;
        changeSupport.firePropertyChange("addrState",
oldAddrState, addrState);
    }

    public String getAddrCountry() {
        return addrCountry;
    }

```

```

    public void setAddrCountry(String addrCountry) {
        String oldAddrCountry = this.addrCountry;
        this.addrCountry = addrCountry;
        changeSupport.firePropertyChange("addrCountry",
oldAddrCountry, addrCountry);
    }
    public BigDecimal getProjectScore() {
        return projectScore;
    }
    public void setProjectScore(BigDecimal projectScore) {
        BigDecimal oldProjectScore = this.projectScore;
        this.projectScore = projectScore;
        changeSupport.firePropertyChange("projectScore",
oldProjectScore, projectScore);
    }
    public BigDecimal getGpa() {
        return gpa;
    }
    public void setGpa(BigDecimal gpa) {
        BigDecimal oldGpa = this.gpa;
        this.gpa = gpa;
        changeSupport.firePropertyChange("gpa", oldGpa, gpa);}

```

## Class Grd\_stdAboutBox

```

java.lang.Object
├ java.awt.Component
│   └ java.awt.Container
│       └ java.awt.Window
│           └ java.awt.Dialog
│               └ javax.swing.JDialog
│                   └ grd_std.Grd_stdAboutBox

```

## All Implemented Interfaces

java.awt.image.ImageObserver, java.awt.MenuContainer,  
java.io.Serializable, javax.accessibility.Accessible,  
javax.swing.RootPaneContainer, javax.swing.WindowCons

## Nested Class Summary

### Nested classes/interfaces inherited from class javax.swing.JDialog

javax.swing.JDialog.AccessibleJDialog

### Nested classes/interfaces inherited from class java.awt.Dialog

java.awt.Dialog.AccessibleAWTDialog, java.awt.Dialog.ModalExclusionType,  
java.awt.Dialog.ModalityType

#### Nested classes/interfaces inherited from class java.awt.Window

java.awt.Window.AccessibleAWTWindow

#### Nested classes/interfaces inherited from class java.awt.Container

java.awt.Container.AccessibleAWTContainer

#### Nested classes/interfaces inherited from class java.awt.Component

java.awt.Component.AccessibleAWTComponent,  
java.awt.Component.BaselineResizeBehavior,  
java.awt.Component.BlitBufferStrategy, java.awt.Component.FlipBufferStrategy

### Method Detail

closeAboutBox  
@Action  
public void **closeAboutBox**()

### Class Grd\_stdApp

java.lang.Object  
└─ org.jdesktop.application.AbstractBean  
    └─ org.jdesktop.application.Application  
        └─ org.jdesktop.application.SingleFrameApplication  
            └─ **Grd\_stdApp**  
public class **Grd\_stdApp**  
extends org.jdesktop.application.SingleFrameApplication

### Nested Class Summary

#### Nested classes/interfaces inherited from class org.jdesktop.application.Application

org.jdesktop.application.Application.ExitListener

### Constructor Summary

[Grd\\_stdApp](#)()

### Method Summary

protected void	<a href="#">configureWindow</a> (java.awt.Window root) This method is to initialize the specified window by injecting resources.
static <a href="#">Grd_stdApp</a>	<a href="#">getApplication</a> () A convenient static getter for the application instance.
static void	<a href="#">main</a> (java.lang.String[] args) Main method launching the application.

protected void

[startup\(\)](#)

At startup create and show the main frame of the application.

## Class Grd\_stdView

java.lang.Object

└ org.jdesktop.application.AbstractBean

└ org.jdesktop.application.View

└ org.jdesktop.application.FrameView

└ **grd\_std.Gr\_stdView**

public class **Gr\_stdView**

extends org.jdesktop.application.FrameView

The application's main frame.

### Constructor Summary

[Gr\\_stdView](#)(org.jdesktop.application.SingleFrameApplication app)

### Method Summary

void [showAboutBox](#)()

### Constructor Detail

#### Gr\_stdView

public **Gr\_stdView**(org.jdesktop.application.SingleFrameApplication app)

### Method Detail

#### showAboutBox

@Action

public void **showAboutBox**()

## Class StudentArchive

java.lang.Object

└ **grd\_std.StudentArchive**

**All Implemented Interfaces:**

java.io.Serializable

@Entity

public class **StudentArchive**

extends java.lang.Object

implements java.io.Serializable

### Constructor Summary

[StudentArchive](#)()

[StudentArchive](#)(java.lang.Long mobileNo)

[StudentArchive](#)(java.lang.Long mobileNo, java.lang.String lastName, java.lang.String email)

Method Summary	
<a href="#">addPropertyChangeListener</a> (java.beans.PropertyChangeListener listener)	void
<a href="#">equals</a> (java.lang.Object object)	boolean
<a href="#">getAddrCity</a> ()	java.lang.String
<a href="#">getAddrCountry</a> ()	java.lang.String
<a href="#">getAddrState</a> ()	java.lang.String
<a href="#">getAddrStreet</a> ()	java.lang.String
<a href="#">getEmail</a> ()	java.lang.String
<a href="#">getEnrollmentYear</a> ()	java.lang.Short
<a href="#">getFirstName</a> ()	java.lang.String
<a href="#">getGpa</a> ()	java.math.BigDecimal
<a href="#">getLastName</a> ()	java.lang.String
<a href="#">getMiddleName</a> ()	java.lang.String
<a href="#">getMobileNo</a> ()	java.lang.Long
<a href="#">getProjectScore</a> ()	java.math.BigDecimal
<a href="#">getReasonOfTermination</a> ()	java.lang.String
<a href="#">getTerminationYear</a> ()	java.lang.Short
<a href="#">hashCode</a> ()	int

<a href="#">removePropertyChangeListener</a> (java.beans.PropertyChangeListener listener)	void
<a href="#">setAddrCity</a> (java.lang.String addrCity)	void
<a href="#">setAddrCountry</a> (java.lang.String addrCountry)	void
<a href="#">setAddrState</a> (java.lang.String addrState)	void
<a href="#">setAddrStreet</a> (java.lang.String addrStreet)	void
<a href="#">setEmail</a> (java.lang.String email)	void
<a href="#">setEnrollmentYear</a> (java.lang.Short enrollmentYear)	void
<a href="#">setFirstName</a> (java.lang.String firstName)	void
<a href="#">setGpa</a> (java.math.BigDecimal gpa)	void
<a href="#">setLastName</a> (java.lang.String lastName)	void
<a href="#">setMiddleName</a> (java.lang.String middleName)	void
<a href="#">setMobileNo</a> (java.lang.Long mobileNo)	void
<a href="#">setProjectScore</a> (java.math.BigDecimal projectScore)	void
<a href="#">setReasonOfTermination</a> (java.lang.String reasonOfTermination)	void
<a href="#">setTerminationYear</a> (java.lang.Short terminationYear)	void
<a href="#">toString</a> ()	java.lang.String

#### Methods inherited from class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait, wait

#### Constructor Detail

##### StudentArchive

public **StudentArchive**()

##### StudentArchive

public **StudentArchive**(java.lang.Long mobileNo)

## StudentArchive

```
public StudentArchive(java.lang.Long mobileNo,  
    java.lang.String lastName,  
    java.lang.String email)
```

### Method Detail

#### getEnrollmentYear

```
public java.lang.Short getEnrollmentYear()
```

#### setEnrollmentYear

```
public void setEnrollmentYear(java.lang.Short enrollmentYear)
```

#### getTerminationYear

```
public java.lang.Short getTerminationYear()
```

#### setTerminationYear

```
public void setTerminationYear(java.lang.Short terminationYear)
```

#### getReasonOfTermination

```
public java.lang.String getReasonOfTermination()
```

#### setReasonOfTermination

```
public void setReasonOfTermination(java.lang.String reasonOfTermination)
```

#### getFirstName

```
public java.lang.String getFirstName()
```

#### setFirstName

```
public void setFirstName(java.lang.String firstName)
```

#### getMiddleName

```
public java.lang.String getMiddleName()
```

#### setMiddleName

```
public void setMiddleName(java.lang.String middleName)
```

#### getLastName

```
public java.lang.String getLastName()
```

#### setLastName

```
public void setLastName(java.lang.String lastName)
```

#### getEmail

```
public java.lang.String getEmail()
```



**setEmail**

```
public void setEmail(java.lang.String email)
```

**getMobileNo**

```
public java.lang.Long getMobileNo()
```

**setMobileNo**

```
public void setMobileNo(java.lang.Long mobileNo)
```

**getAddrStreet**

```
public java.lang.String getAddrStreet()
```

**setAddrStreet**

```
public void setAddrStreet(java.lang.String addrStreet)
```

**getAddrCity**

```
public java.lang.String getAddrCity()
```

**setAddrCity**

```
public void setAddrCity(java.lang.String addrCity)
```

**getAddrState**

```
public java.lang.String getAddrState()
```

**setAddrState**

```
public void setAddrState(java.lang.String addrState)
```

**getAddrCountry**

```
public java.lang.String getAddrCountry()
```

**setAddrCountry**

```
public void setAddrCountry(java.lang.String addrCountry)
```

**getProjectScore**

```
public java.math.BigDecimal getProjectScore()
```

**setProjectScore**

```
public void setProjectScore(java.math.BigDecimal projectScore)
```

**getGpa**

```
public java.math.BigDecimal getGpa()
```

**setGpa**

```
public void setGpa(java.math.BigDecimal gpa)
```

### **hashCode**

```
public int hashCode()
```

#### **Overrides:**

```
hashCode in class java.lang.Object
```

### **equals**

```
public boolean equals(java.lang.Object object)
```

#### **Overrides:**

```
equals in class java.lang.Object
```

### **toString**

```
public java.lang.String toString()
```

#### **Overrides:**

```
toString in class java.lang.Object
```

### **addPropertyChangeListener**

```
public void addPropertyChangeListener(java.beans.PropertyChangeListener listener)
```

### **removePropertyChangeListener**

```
public void removePropertyChangeListener(java.beans.PropertyChangeListener listener)
```

## For the student Archive(view with Editing)

The running application with a graphical user interface (GUI) that has the following features:

- Ability to view and modify values in nine columns of the StudentArchive database.
- Menu items.
- Persistence of its window state between sessions. When you close the application, the window position and size are remembered. So when you reopen the application, the window opens in the same position as it was when you closed it as well as the column's size are been remembered.

The connection between the master table (a JTable component) and the database is handled with a combination of the following mechanisms:-

The `StudentArchive.java` entity class, which is used to read and write data to the `StudentArchive` database table. Entity classes are a special type of class that enable you to interact with databases through Java code. Entity classes use Java annotations to map class fields to database columns.

- The `META-INF/persistence.xml` file, which defines a connection between the database and the entity class. This file is also known as the persistence unit.
- Using beans binding to connect the properties of the entity class with the properties of the `JTable` component.
- The `entityManager`, `query`, and `list` objects, which are defined in the `StudentArchive View` class:-
  - The entity manager object is used to retrieve and commit data within the defined persistence unit scope.
  - The query object defines how the particular data collection is retrieved from the entity manager.
  - The list object is an observable collection that holds the data from the query. An observable collection is a special kind of collection on which you can place a listener to find out when changes to the collection have been made.

Every Graduate has the ability for editing his own archive only. The `WHERE` clause adds filtering capability to the `FROM-SELECT` structure. Only `StudentArchive` objects for which the Boolean expression evaluates to `TRUE` are passed to the `SELECT` clause and then collected as query results.

```
@NamedQuery (name = "StudentArchive.findByUsername", query = "SELECT s  
FROM StudentArchive s WHERE s.username =: user")}
```

## The use of binding

Package `org.jdesktop.beansbinding` Provides support for defining properties and creating bindings between sets of two properties.  
properties

Property defines a uniform way to access the value of a property. A typical property implementation allows you to create an immutable representation of a way to derive some property from a source object. As such, all methods of this class take a source object as an argument.

A property implementation may, however, be designed such that the property itself is a mutable thing that stores a property value. In such a case, the property implementation may ignore the source object. property implementations should clearly document their behavior in this regard.

You can listen for changes in the state of a property by registering Property State Listeners on the property.

The method in `org.jdesktop.beansbinding` that return `ELProperty`  
`ELProperty.create(java.lang.String expression)`  
Creates an instance of `ELProperty` for the given expression.

## AutoBinding

An implementation of `Binding` that automatically syncs the source and target by refreshing and saving according to one of three update strategies. The update strategy is specified for an `Auto Binding` on creation, and is one of:

```
AutoBinding.UpdateStrategy.READ_ONCE
```

```
AutoBinding.UpdateStrategy.READ
```

```
AutoBinding.UpdateStrategy.READ_WRITE
```

In my code the `READ_WRITE` strategy is used which Tries to keep both the source and target in sync with each other.

## Refresh and Save Methods:

```
save()
```

Fetches the value of the target property for the target object and sets it as the value of the source property for the source object.

```
refresh()
```

Fetches the value of the source property for the source object and sets it as the value of the target property for the target object.

→The Graduate can edit specific columns (only in his archive) and this is achieved by binding only the editable columns.

JTableBinding class extends Auto Binding and Binds a List of objects to act as the rows of a JTable. Each object in the source List represents one row in the JTable. Mappings from properties of the source objects to columns are created by adding ColumnBinding to a JTableBinding. Instances of JTableBinding are obtained by calling one of the createJTableBinding methods in the SwingBindings class. The JTable target of a JTableBinding acts as a live view of the objects in the source List. JTableBinding listens to the properties specified for the ColumnBindings, for all objects in the List, and updates the values displayed in the JTable in response to change. All successful edits made to JTable cell values are immediately committed back to corresponding objects in the source List.

## The code

```
org.jdesktop.swingbinding.JTableBinding jTableBinding =
org.jdesktop.swingbinding.SwingBindings.createJTableBinding
(org.jdesktop.beansbinding.AutoBinding.UpdateStrategy.READ_
WRITE, list, masterTable);
    org.jdesktop.swingbinding.JTableBinding.ColumnBindi
ng columnBinding =
jTableBinding.addColumnBinding(org.jdesktop.beansbinding.EL
Property.create("${firstName}"));
    columnBinding.setColumnName("First Name");
    columnBinding.setColumnClass(String.class);
    columnBinding =
jTableBinding.addColumnBinding(org.jdesktop.beansbinding.EL
Property.create("${middleName}"));
    columnBinding.setColumnName("Middle Name");
    columnBinding.setColumnClass(String.class);
    columnBinding =
jTableBinding.addColumnBinding(org.jdesktop.beansbinding.EL
Property.create("${lastName}"));
    columnBinding.setColumnName("Last Name");
    columnBinding.setColumnClass(String.class);
    columnBinding =
jTableBinding.addColumnBinding(org.jdesktop.beansbinding.EL
Property.create("${email}"));
    columnBinding.setColumnName("Email");
    columnBinding.setColumnClass(String.class);
    columnBinding =
jTableBinding.addColumnBinding(org.jdesktop.beansbinding.EL
Property.create("${mobileNo}"));
    columnBinding.setColumnName("Mobile No");
    columnBinding.setColumnClass(Long.class);
    columnBinding =
jTableBinding.addColumnBinding(org.jdesktop.beansbinding.EL
Property.create("${addrStreet}"));
```

```

        columnBinding.setColumnName("Addr Street");
        columnBinding.setColumnClass(String.class);
        columnBinding =
jTableBinding.addColumnBinding(org.jdesktop.beansbinding.EL
Property.create("${addrCity}"));
        columnBinding.setColumnName("Addr City");
        columnBinding.setColumnClass(String.class);
        columnBinding =
jTableBinding.addColumnBinding(org.jdesktop.beansbinding.EL
Property.create("${addrState}"));
        columnBinding.setColumnName("Addr State");
        columnBinding.setColumnClass(String.class);
        columnBinding =
jTableBinding.addColumnBinding(org.jdesktop.beansbinding.EL
Property.create("${addrCountry}"));
        columnBinding.setColumnName("Addr Country");
        columnBinding.setColumnClass(String.class);
        bindingGroup.addBinding(jTableBinding);

```

**An example action method showing how to create asynchronous task running on background and how to show their progress**

```

@Action
public Task refresh() {
return new RefreshTask(getApplication());
}

private class RefreshTask extends Task {
RefreshTask(org.jdesktop.application.Application app) {
super(app);
}
@SuppressWarnings("unchecked")
@Override protected Void doInBackground() {
try {
setProgress(0, 0, 4);
setMessage("Rolling back the current changes...");
setProgress(1, 0, 4);
entityManager.getTransaction().rollback();
setProgress(2, 0, 4);

setMessage("Starting a new transaction...");
entityManager.getTransaction().begin();
setProgress(3, 0, 4);

setMessage("Fetching new data...");
java.util.Collection data = query.getResultList();
for (Object entity : data) {
entityManager.refresh(entity);
}
}
}

```

```

setProgress(4, 0, 4);

list.clear();
list.addAll(data);
} catch (InterruptedException ignore) { }
return null;
}
@Override protected void finished() {
setMessage("Done.");
setSaveNeeded(false);
}
}

```

### Class ArchiveAboutBox

java.lang.Object

└ java.awt.Component

└ java.awt.Container

└ java.awt.Window

└ java.awt.Dialog

└ javax.swing.JDialog

└ archive.ArchiveAboutBox

All Implemented Interfaces

java.awt.image.ImageObserver, java.awt.MenuContainer,

java.io.Serializable, javax.accessibility.Accessible,

javax.swing.RootPaneContainer, javax.swing.WindowConstants

public class ArchiveAboutBox

extends javax.swing.JDialog

#### Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JDialog

javax.swing.JDialog.AccessibleJDialog

Nested classes/interfaces inherited from class java.awt.Dialog

java.awt.Dialog.AccessibleAWTDialog, java.awt.Dialog.ModalExclusionType,  
java.awt.Dialog.ModalityType

Nested classes/interfaces inherited from class java.awt.Window

java.awt.Window.AccessibleAWTWindow

Nested classes/interfaces inherited from class java.awt.Container

java.awt.Container.AccessibleAWTContainer

## Nested classes/interfaces inherited from class java.awt.Component

java.awt.Component.AccessibleAWTComponent,  
java.awt.Component.BaselineResizeBehavior,  
java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Class Archive.extends  
java.lang.Object implements Serializable  
Serialized Fields

changeSupport  
java.beans.PropertyChangeSupport changeSupport

enrollmentYear  
java.lang.Short enrollmentYear

terminationYear  
java.lang.Short terminationYear

reasonOfTermination  
java.lang.String reasonOfTermination

firstName  
java.lang.String firstName

middleName  
java.lang.String middleName

lastName  
java.lang.String lastName

email  
java.lang.String email

mobileNo  
java.lang.Long mobileNo

addrStreet  
java.lang.String addrStreet

addrCity  
java.lang.String addrCity

addrState  
java.lang.String addrState



addrCountry

java.lang.String addrCountry

projectScore

java.math.BigDecimal projectScore

gpa

java.math.BigDecimal gpa

username

java.lang.String username

Class ArchiveApp

java.lang.Object

└ org.jdesktop.application.AbstractBean

└ org.jdesktop.application.Application

└ org.jdesktop.application.SingleFrameApplication

└ archive.ArchiveApp

**public class ArchiveApp**

extends org.jdesktop.application.SingleFrameApplication

The main class of the application.

▪ Nested Class Summary

Nested classes/interfaces inherited from class org.jdesktop.application.Application
---

org.jdesktop.application.Application.ExitListener
---

Method Detail
---------------

startup

protected void startup()

At startup create and show the main frame of the application.

Specified by:

startup in class org.jdesktop.application.Application

configureWindow

protected void configureWindow(java.awt.Window root)

This method is to initialize the specified window by injecting resources.

Windows shown in our application come fully initialized from the GUI builder, so this additional configuration is not needed.

Overrides:

configureWindow in class org.jdesktop.application.SingleFrameApplication

getApplication

public static [ArchiveApp](#) getApplication()

A convenient static getter for the application instance.

Returns:

the instance of ArchiveApp

main

```
public static void main(java.lang.String[] args)  
    Main method launching the application.
```

Class StudentArchive

```
java.lang.Object
```

```
└ archive.StudentArchive
```

All Implemented Interfaces

```
java.io.Serializable
```

```
public class StudentArchive
```

```
extends java.lang.Object
```

```
implements java.io.Serializable
```

```
public class StudentArchive
```

```
extends java.lang.Object
```

```
implements java.io.Serializable
```

#### Method Detail

getEnrollmentYear

```
public java.lang.Short getEnrollmentYear()
```

setEnrollmentYear

```
public void setEnrollmentYear(java.lang.Short enrollmentYear)
```

getTerminationYear

```
public java.lang.Short getTerminationYear()
```

setTerminationYear

```
public void setTerminationYear(java.lang.Short terminationYear)
```

getReasonOfTermination

```
public java.lang.String getReasonOfTermination()
```

setReasonOfTermination

```
public void setReasonOfTermination(java.lang.String reasonOfTermination)
```

getFirstName

```
public java.lang.String getFirstName()
```

setFirstName

```
public void setFirstName(java.lang.String firstName)
```

getMiddleName

```
public java.lang.String getMiddleName()
```

setMiddleName

```
public void setMiddleName(java.lang.String middleName)
```

getLastName

```
public java.lang.String getLastName()
```

setLastName

```
public void setLastName(java.lang.String lastName)
```

getEmail

```
public java.lang.String getEmail()
```

setEmail

```
public void setEmail(java.lang.String email)
```

getMobileNo

```
public java.lang.Long getMobileNo()
```

setMobileNo

```
public void setMobileNo(java.lang.Long mobileNo)
```

getAddrStreet

```
public java.lang.String getAddrStreet()
```

setAddrStreet

```
public void setAddrStreet(java.lang.String addrStreet)
```

getAddrCity

```
public java.lang.String getAddrCity()
```

setAddrCity

```
public void setAddrCity(java.lang.String addrCity)
```

getAddrState

```
public java.lang.String getAddrState()
```

setAddrState

```
public void setAddrState(java.lang.String addrState)
```

getAddrCountry

```
public java.lang.String getAddrCountry()
```

setAddrCountry

```
public void setAddrCountry(java.lang.String addrCountry)
```

getProjectScore

```
public java.math.BigDecimal getProjectScore()
```

setProjectScore

```
public void setProjectScore(java.math.BigDecimal projectScore)
```

getGpa

```
public java.math.BigDecimal getGpa()
```

setGpa

```
public void setGpa(java.math.BigDecimal gpa)
```

getUsername

```
public java.lang.String getUsername()
```

setUsername

```
public void setUsername(java.lang.String username)
```

equals

```
public boolean equals(java.lang.Object object)
```

Overrides:

equals in class java.lang.Object

toString

```
public java.lang.String toString()
```

Overrides:

toString in class java.lang.Object

addPropertyChangeListener

```
public void
```

addPropertyChangeListener

```
(java.beans.PropertyChangeListener listener)
```

removePropertyChangeListener

```
public void
```

removePropertyChangeListener

```
(java.beans.PropertyChangeListener listener)
```

## Bibliography

- 1- **Oracle® Database:** Recovery Guide
- 2- **Oracle® Database:** SQL Language Reference 11g Release 1
- 3- Database System concepts( Silberschatz, Korth and Sudarshan )
- 4- **Oracle/SQL Tutorial**( Michael Gertz )
- 5- **Oracle® Database:** Administrator's Guide 11g Release 1.
- 6- **Oracle® Java Tutorials**