**A Database for the**

**Management of University Database**

Project for the class Database Systems II

in the Summer Semester 2024

The following persons have contributed to this project:

Close-up of a handwritten word

Description automatically generated A signature on a white background

Description automatically generated

Sanjay Prabhu Kunjibettu Tanay Rajendrakumar Khilare

41kusa1mst@hft-stuttgart.de 41khta1mst@hft-stuttgart.de

**Acknowledgement**

We extend our sincere gratitude to Prof. Dorothee Koch and Dr. rer. nat. Manfred Besner for granting us the opportunity to work on and present our University Management System project as part of the Summer Semester 2024 pre-exam requirements. This project significantly enhanced our understanding of Database Management System as a subject and also provided us with valuable practical experience in MySQL and JAVA.

We also wish to express our appreciation to the staff of Hochschule für Technik Stuttgart for facilitating our access to the necessary credentials to work with our database on the LIDA Server smoothly.

TABLE OF CONTENTS

Contents

[**1.** **Data Model for the University Management Database** 6](#_Toc169278076)

[**1.1** **Explanation of the Data and the Application** 6](#_Toc169278077)

[**1.2** **The Data Model** 6](#_Toc169278078)

[**1.2.1** **Modern ER Diagram representing Relational Schema for Operations** 6](#_Toc169278079)

[**1.2.2** **Another version of ER Diagram ( The first ER Diagram captures this as well but in a better readable format)** 8](#_Toc169278080)

[**2. System Requirements** 9](#_Toc169278081)

[**2.1 Hardware Requirements** 9](#_Toc169278082)

[**2.2 Software Requirements** 9](#_Toc169278083)

[**2.3 Project Management Requirements** 10](#_Toc169278084)

[**3.** **Relational Design** 10](#_Toc169278085)

[**3.1** **Table Schemas** 10](#_Toc169278086)

[**3.2** **Database Tables with Data** 13](#_Toc169278087)

[**3.2.1** **ADMINISTRATOR table** 13](#_Toc169278088)

[**3.2.2** **COURSE table** 13](#_Toc169278089)

[**3.2.3** **DEPARTMENT table** 14](#_Toc169278090)

[**3.2.4** **INSTRUCTOR table** 14](#_Toc169278091)

[**3.2.5** **STUDENT table** 15](#_Toc169278092)

[**3.2.6** **FEES table** 15](#_Toc169278093)

[**3.2.7** **ENROLLMENT table** 15](#_Toc169278094)

[**3.2.8** **SUBJECT table** 16](#_Toc169278095)

[**3.2.9** **GRADES table** 16](#_Toc169278096)

[**3.3** **Normalization** 16](#_Toc169278097)

[**3.4** **Integrity Constraints** 17](#_Toc169278098)

[**4.** **Use Cases** 20](#_Toc169278099)

[**4.1** **Use Cases** 20](#_Toc169278100)

[**4.1.1 Consolidated UML Diagram for all primary use cases** 21](#_Toc169278101)

[**4.1.2** **Future Proposal – Restricted access to Parents / Guardians** 21](#_Toc169278102)

[**4.2 Description of the Graphical User Interface** 22](#_Toc169278103)

[**5. Transactions / Triggers** 22](#_Toc169278104)

[**5.1 Transactions** 22](#_Toc169278105)

[**5.1.1 DML Statements part of the stored procedure call** 22](#_Toc169278106)

[**5.1.2 Transactions as per the use cases** 24](#_Toc169278107)

[**5.2 Triggers** 37](#_Toc169278108)

[**5.2.1 Trigger for the ADMINISTRATOR** 37](#_Toc169278109)

[**5.2.2 Trigger for the INSTRUCTOR** 38](#_Toc169278110)

[**5.2.3 Trigger for the STUDENT** 38](#_Toc169278111)

[**5.2.4 Trigger for the FEES** 39](#_Toc169278112)

[**5.3 Stored Procedures** 39](#_Toc169278113)

[**5.3.1 Stored Procedure for the DDL Statements** 39](#_Toc169278114)

[**5.3.2 Stored Procedure for the DML Statements** 42](#_Toc169278115)

[**5.4 Declaration about used AI Tools** 44](#_Toc169278116)

[**6. List of References ( Other than class notes on Moodle )** 45](#_Toc169278117)

[**7. Appendix** 46](#_Toc169278118)

[**7.1 Source Code of the Application and User Interface** 46](#_Toc169278119)

[7.1.1 Custom Connection Class – Conn.java 46](#_Toc169278120)

[7.1.2 Mapping of Department to Course – DepartmentToCourseMapping.java 46](#_Toc169278121)

[7.1.3 Initial Pop Up Screen – Splash.java 47](#_Toc169278122)

[7.1.4 Login Page for Admin – AdminLogin.java 49](#_Toc169278123)

[7.1.5 Admin Desktop / Main Page – AdminMain.java 51](#_Toc169278124)

[7.1.6 Person Interface – Person.java 55](#_Toc169278125)

[7.1.7 Instructor Class implementing Person Interface – Instructor.java 55](#_Toc169278126)

[7.1.8 Student Class implementing Person Interface – Student.java 59](#_Toc169278127)

[7.1.9 Java Swings Class to add a new instructor record – AddInstructor.java 65](#_Toc169278128)

[7.1.10 Java Swings Class to update/delete an existing instructor record – UpdateInstructor.java 70](#_Toc169278129)

[7.1.11 Java Swings Class to view an existing instructor record – ViewInstructor.java 76](#_Toc169278130)

[7.1.12 Java Swings Class to insert a new student record – AddStudent.java 80](#_Toc169278131)

[7.1.13 Java Swings Class to update/deregister an existing student record – UpdateStudent.java 85](#_Toc169278132)

[7.1.14 Java Swings Class to view an existing student record – ViewStudent.java 91](#_Toc169278133)

[7.1.15 Java Swings Class to view the fee structures – FeeStructure.java 95](#_Toc169278134)

[7.1.16 Java Swings Class to pay the fees – FeesForm.java 97](#_Toc169278135)

[7.1.17 Java Swings Class to insert grades for subjects – InsertGrades.java 102](#_Toc169278136)

[7.1.18 Java Swings Class to view results – ViewGrades.java 107](#_Toc169278137)

[7.1.19 Stored Procedures with DDL and DML Statements 111](#_Toc169278138)

[7.1.20 Triggers 115](#_Toc169278139)

[**7.2 Other sources of inspiration for the project** 117](#_Toc169278140)

# **Data Model for the University Management Database**

## **1.1 Explanation of the Data and the Application**

The University Management Database[1][5] is designed to enhance and streamline the administrative processes involving students and instructors by providing a single platform to efficiently manage tasks such as adding, updating and remove student and instructor records, entering grades and viewing the results, and also viewing the free structures and handling simple fee payments.

The key entities are administrators (who are responsible for managing the database and currently overseeing the operations and transactions for instructors and students), students (aged above 16 who are enrolled at the university titled “Hochschule für Wissenschaft Europa”) and instructors (aged above 25 who are responsible for the courses and departments for the enrolled students). Additional entities are subjects, grades, courses, departments and enrolments each of which is responsible for holding the respective data concerned with the entity-relations between the primary three users.

The typical queries are to select data (retrieving specific information such as the student, instructor and course records), inserting data (adding new records for students and instructors), modifying data (modifying certain fields of existing records for students and instructors), deleting data (deregistering students and removing the instructor records), stored procedures[4] (one is responsible to run the DDL and the other is responsible to run the DML queries) and lastly, triggers[3] (automated routines handing actions against conditions like generating user\_ids, passwords and emails based on the inserted records into student and instructor tables).

By incorporating these elements, the project aims to provide a simple and user-friendly solution to manage simple university operations[1][5] within a unified platform.

## **1.2 The Data Model**

### **1.2.1 Modern ER Diagram representing Relational Schema for Operations**

The Entity-Relational Diagram showcased below[5] provides a clear representation of the relational schema governing the operations within the University Management System[1][5]. Each table, including those about administrators, is interconnected via integrity constraints, marked as Primary Keys (pk) and Secondary/Foreign Keys (sk). The ER Diagram is designed with the following objectives in mind:

1. The **administrator** table houses individuals with privileged access to the database, enabling them to manage **student** and **instructor** parent records, as well as associated child records such as **enrolment**, **grades**, and **courses**.
2. Robust data integrity constraints are implemented to ensure accurate relationships between tables, facilitating the automatic removal of orphaned records, and thereby maintaining data consistency and reliability.

A diagram of a computer

Description automatically generated

### **1.2.2 Another version of ER Diagram ( The first ER Diagram captures this as well but in a better readable format)**

A diagram of a network

Description automatically generated

# **2. System Requirements**

The project was designed and operated on a Windows 11 operating system with a 64-bit architecture, running on a laptop equipped with 16 GB of RAM and 500 GB SSD storage. Below is a summary of the hardware, software, and project management requirements necessary for the seamless operation of the project:

## **2.1 Hardware Requirements**

|  |  |
| --- | --- |
| Hardware Components | Requirements |
| Operating System | Windows 10 / 11 / 11 pro |
| Architecture | x64-bit |
| Processor | Intel i5 10th gen or higher |
| RAM | Recommended: at least 8GB |
| Storage | SSD with at least 500 GB capacity |
| Network | High Speed internet connection for accessing LIDA server |
| Database Servers | HFT LIDA Server |

## **2.2 Software Requirements**

|  |  |  |
| --- | --- | --- |
| Software Requirement | Tool | Purpose |
| Integrated Development Environment | Recommended: [IntelliJ IDEA Community Edition 2021.1.1](https://www.jetbrains.com/idea/download/other.html)  Alternative: Apache Netbeans IDE 21 | IDE for implementing GUI and connecting the GUI to the MySQL database on the LIDA server |
| Version Control | Optional: GitHub (be aware of the GDPR regulations) | Committing, pushing, pulling or rolling back the changes to the code |
| Database Management Tools | Recommended: [SQL Workbench/J](https://sql-workbench.eu/)  Alternative: PuTTy release 0.81 (x64 bit) (for Windows only) | GUI for handling MySQL database queries and transactions |
| Database | MySQL | SQL Database used for the project |
| JDBC Connectivity | mysql-connector-java-8.0.23.jar | MySQL JDBC driver required to connect to the MySQL database via the IDE or Thin Client |
| Database Scripts | Recommended: [Notepad++ (64-bit x64)](https://notepad-plus-plus.org/downloads/)  Alternative: Sublime Text 3 | Additional Text based IDEs to write SQL scripts and perform regex quality testing |
| Java Development Kit | Recommended: [JDK 21](https://www.oracle.com/de/java/technologies/downloads/) or higher | Required for JAVA as the main programming language |
| Graphical User Interface | Java Swings | Frontend development of the GUI |
| Virtual Network | Cisco Secure Client - AnyConnect VPN 5.1.3.62 | to access HFT LIDA server |
| Plugins | [jcalendar-1.4.zip](https://toedter.com/jcalendar/)  [rs2xml.jar](https://sourceforge.net/projects/finalangelsanddemons/files/rs2xml.jar/download) | These are external plugins to be placed inside the code/library repository to add the additional functionalities of date/time and JTable respectively. |

## **2.3 Project Management Requirements**

|  |  |
| --- | --- |
| Project Management Task | Tool |
| Agile Methodology | O365 Planner  Optional: Atlassian JIRA |
| Communication Management | O365 Teams accessible using university user ids  (data integrity) |
| Project Documentations | O365 Microsoft Word |
| Code Snippets | <https://carbon.now.sh/> |
| UML and Use Case Diagrams | <https://draw.io>  <https://excalidraw.com/> |

# **Relational Design**

## **3.1 Table Schemas**

The following are the snippets of the SQL code to create tables for our project. These statements are also a part of our stored procedure illustrated in the later chapter. The schemas are well equipped with data integrity constraints defined by the primary and foreign keys deemed as necessary as per the ER Diagram. Referential integrities have been carefully implemented to avoid having orphaned nodes when parent table records are removed.

/\* DDL STATEMENTS \*/  
  
/\* 1. DROP TABLES \*/  
  
/\*1.1\*/  
DROP TABLE IF EXISTS GRADES;  
DROP TABLE IF EXISTS SUBJECT;  
  
/\*1.2\*/  
DROP TABLE IF EXISTS FEES;  
DROP TABLE IF EXISTS ENROLLMENT;  
  
/\*1.3\*/  
DROP TABLE IF EXISTS COURSE;  
DROP table IF EXISTS DEPARTMENT;  
  
/\*1.4\*/  
DROP TABLE IF EXISTS INSTRUCTOR;  
DROP TABLE IF EXISTS STUDENT;  
  
/\*1.5\*/  
DROP TABLE IF EXISTS ADMINISTRATOR;  
  
/\* 2. SCHEMA CREATION \*/  
  
/\*2.1 ADMINISTRATOR TABLE \*/  
CREATE TABLE ADMINISTRATOR(  
 USER\_ID *VARCHAR*(16) PRIMARY KEY,  
 PSWD *VARCHAR*(16),  
 FIRST\_NAME *VARCHAR*(255),  
 LAST\_NAME *VARCHAR*(255),  
 CONTACT *VARCHAR*(17),  
 EMAIL *VARCHAR*(255) /\* Trigger to generate regex email -> firstname.lastname@hft-europa.com\*/  
);  
  
/\*2.2 INSTRUCTOR TABLE \*/  
CREATE TABLE INSTRUCTOR(  
 USER\_ID *VARCHAR*(16),  
 PSWD *VARCHAR*(16),  
 FIRST\_NAME *VARCHAR*(255),  
 LAST\_NAME *VARCHAR*(255),  
 CONTACT *VARCHAR*(15),  
 EMAIL *VARCHAR*(50), /\* Trigger to generate regex email -> firstname.lastname@hft-europa.com\*/  
 DOB *VARCHAR*(255),  
 HOME\_ADDRESS *VARCHAR*(255),  
 HFT\_CABIN *VARCHAR*(255),  
 EMP\_ID *VARCHAR*(16) PRIMARY KEY  
);  
  
/\*2.3 STUDENT TABLE \*/  
CREATE TABLE STUDENT(  
 USER\_ID *VARCHAR*(16),  
 PSWD *VARCHAR*(16),  
 FIRST\_NAME *VARCHAR*(255),  
 LAST\_NAME *VARCHAR*(255),  
 CONTACT *VARCHAR*(15),  
 EMAIL *VARCHAR*(50),  
 DOB *VARCHAR*(255),  
 HOME\_ADDRESS *VARCHAR*(255),  
 GRADE\_XII *VARCHAR*(16),  
 INTERNATIONAL\_STUDENT *VARCHAR*(16),  
 IMMA\_ID *VARCHAR*(16) PRIMARY KEY  
);  
  
/\*2.4 DEPARTMENT TABLE \*/  
CREATE TABLE DEPARTMENT(  
 DEPT\_ID *INT* PRIMARY KEY,  
 DEAN\_ID *VARCHAR*(16),  
 DEPT\_NAME *VARCHAR*(255),  
 FOREIGN KEY (DEAN\_ID) REFERENCES INSTRUCTOR(EMP\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);  
  
/\*2.5 COURSE TABLE \*/  
CREATE TABLE COURSE(  
 COURSE\_ID *INT* PRIMARY KEY,  
 DEPT\_ID *INT*,  
 INSTRUCTOR\_ID *VARCHAR*(16),  
 COURSE\_NAME *VARCHAR*(255),  
 COURSE\_CODE *VARCHAR*(10), /\* Master Software Technology -> mst \*/  
 CREDITS *INT*,  
 BUILDING\_NO *INT*,  
 ROOM\_NO *INT*,  
 FOREIGN KEY (DEPT\_ID) REFERENCES DEPARTMENT(DEPT\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE,  
 FOREIGN KEY (INSTRUCTOR\_ID) REFERENCES INSTRUCTOR(EMP\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);  
  
/\*2.6 FEES \*/  
CREATE TABLE FEES(  
 STUDENT\_ID *VARCHAR*(16),  
 TUITION\_FEES *VARCHAR*(16), /\* 1500 Euros for International Students \*/  
 SEMESTER\_FEES *VARCHAR*(16), /\* 200 Euros for all \*/  
 D\_TICKET *VARCHAR*(16), /\* Optional: 360 Euros for all \*/  
 RESEARCH\_VARIABLE *VARCHAR*(16), /\* Optional: Variable \*/  
 FEES\_PAID *VARCHAR*(3),  
 FOREIGN KEY (STUDENT\_ID) REFERENCES STUDENT(IMMA\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);  
  
/\*2.7 ENROLLMENT \*/  
CREATE TABLE ENROLLMENT(  
 IMMA\_ID *VARCHAR*(16),  
 DEPT\_ID *INT*,  
 COURSE\_ID *INT*,  
 IMMA\_DATE *VARCHAR*(255),  
 IMMA\_STATUS *VARCHAR*(100),  
 FOREIGN KEY (IMMA\_ID) REFERENCES STUDENT(IMMA\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE,  
 FOREIGN KEY (DEPT\_ID) REFERENCES DEPARTMENT(DEPT\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE,  
 FOREIGN KEY (COURSE\_ID) REFERENCES COURSE(COURSE\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);  
  
/\*2.8 SUBJECT TABLE \*/  
CREATE TABLE SUBJECT(  
 STUDENT\_ID *VARCHAR*(16),  
 COURSE\_ID *INT*,  
 SEMESTER *INT*,  
 SUBJECT1 *VARCHAR*(255),  
 SUBJECT2 *VARCHAR*(255),  
 SUBJECT3 *VARCHAR*(255),  
 SUBJECT4 *VARCHAR*(255),  
 SUBJECT5 *VARCHAR*(255),  
 ELECTIVE1 *VARCHAR*(255),  
 FOREIGN KEY (STUDENT\_ID) REFERENCES STUDENT(IMMA\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE,  
 FOREIGN KEY (COURSE\_ID) REFERENCES COURSE(COURSE\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);  
  
/\*2.9 GRADES TABLE \*/  
CREATE TABLE GRADES(  
 STUDENT\_ID *VARCHAR*(16),  
 COURSE\_ID *INT*,  
 SEMESTER *INT*,  
 GRADE1 *VARCHAR*(8),  
 GRADE2 *VARCHAR*(8),  
 GRADE3 *VARCHAR*(8),  
 GRADE4 *VARCHAR*(8),  
 GRADE5 *VARCHAR*(8),  
 GRADE6 *VARCHAR*(8),  
 FOREIGN KEY (STUDENT\_ID) REFERENCES STUDENT(IMMA\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE,  
 FOREIGN KEY (COURSE\_ID) REFERENCES COURSE(COURSE\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);

## **3.2 Database Tables with Data**

There are 9 tables defined in our ER-Diagram. Below are the snippets of all the tables. The values for all the tables except ADMINSTRATOR, COURSE and DEPARTMENT are dynamically added during transactions, while the former are added during the stored procedure runs. An exception to the COURSE table is the column ***instructor\_id*** where the fields are populated during transactions.

The following snippets illustrate the backend data present in our tables:

### **3.2.1 ADMINISTRATOR table**

SELECT \* FROM ADMINISTRATOR;

**A screenshot of a computer

Description automatically generated**

### **3.2.2 COURSE table**

**Note:** The ***instructor\_id*** value was updated during a transaction of inserting a new instructor record as illustrated in the coming chapters.

SELECT \* FROM COURSE;

**A screenshot of a computer

Description automatically generated**

### **3.2.3 DEPARTMENT table**

**Note:** The ***dean\_id*** field is for **future scope**. The goal is to add innovative triggers[3] in our database system and code to automatically or manually assigned deans.

SELECT \* FROM DEPARTMENT;

A screenshot of a computer

Description automatically generated

### **3.2.4 INSTRUCTOR table**

**Note:** Values displayed here **after the transactions have been committed**, illustrated in the later sections.

SELECT \* FROM INSTRUCTOR;

**A screenshot of a computer

Description automatically generated**

### **3.2.5 STUDENT table**

**Note:** Values displayed here **after the transactions have been committed**, illustrated in the later sections.

SELECT \* FROM STUDENT;

A screenshot of a computer

Description automatically generated

### **3.2.6 FEES table**

**Note:** Values displayed here **after the transactions have been committed**, illustrated in the later sections.

SELECT \* FROM FEES;

**A screenshot of a computer

Description automatically generated**

### **3.2.7 ENROLLMENT table**

**Note:** Values displayed here **after the transactions have been committed**, illustrated in the later sections. The working is as follows –

1. The Administrator adds a new record for a student, and the status is ***admitted***.
2. After the student has paid the fees, the matriculation status is updated as ***enrolled.***
3. The matriculation date is the date when the student was admitted into the university.
4. Every enrolment of a student record is tagged against a course and the corresponding department (here, course refers to the main branch / programme, not to be confused with the subjects and the grades).

SELECT \* FROM ENROLLMENT;

**A screenshot of a computer

Description automatically generated**

### **3.2.8 SUBJECT table**

**Note:** Values displayed here **after the transactions have been committed**, illustrated in the later sections.

SELECT \* FROM SUBJECT;

**A screenshot of a computer

Description automatically generated**

### **3.2.9 GRADES table**

**Note:** Values displayed here **after the transactions have been committed**, illustrated in the later sections.

SELECT \* FROM GRADES;

A screenshot of a computer

Description automatically generated

## **3.3 Normalization**

Overall, all the schemas are in 2NF, with some tables also achieving 3NF forms as summarized in the table below -

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Normalization | Description | Justification |
| ADMINISTRATOR, INSTRUCTOR, STUDENT, DEPARTMENT, FEES | 3NF | 1. 3NF - There are no transitive dependencies 2. 2NF – All non key attributes are fully functionally dependent on the primary key 3. 1NF - All fields are atomic values. | Most of these tables are dimensional tables (except for department and fees which are connected to instructor and student but without a transitive dependency). Such tables maintain 3NF as they only contain additional information with a primary key serving as the prime purpose of relational integration. |
| COURSE, ENROLLMENT, SUBJECT, GRADES | 2NF | 1. 2NF – All non key attributes are fully functionally dependent on the primary key 2. 1NF - All fields are atomic values. | These tables have a relational transitive dependency on the other 3NF tables as they contain the main summarized fields, gradually turning into fact tables. |

**Future Scope:** To enhance the efficiency and normalization of the database, the tables currently in 2NF can be further decomposed into 3NF, provided additional fact tables are introduced. In practice, maintaining a large database effectively requires a greater number of dimension tables and fewer fact tables. This approach helps optimize database performance and manageability.

## **3.4 Integrity Constraints**

Referential integrity constraints have been established to ensure data integrity and consistency, thereby maintaining appropriate relationships between tables and fields. The table below summarizes all the referential integrity constraints:

|  |  |  |
| --- | --- | --- |
| Table | Referential Integrity | Comments |
| ADMINISTRATOR | No referential integrity constraints defined | 1. Dimension table which maintains admin user data and is standalone. |
| INSTRUCTOR | No referential integrity constraints defined | 1. Dimension table which maintains instructor user data and is standalone. |
| STUDENT | No referential integrity constraints defined | 1. Dimension table which maintains student user data and is standalone. |
| DEPARTMENT | FOREIGN KEY (DEAN\_ID) REFERENCES INSTRUCTOR(EMP\_ID) ON DELETE SET NULL ON UPDATE CASCADE | 1. Referential integrity between the *dean\_id* of the **department** table and the primary key *instructor\_id* of the **instructor** table. 2. Changes in the **instructor** table are reflected in the **department** table 3. **Designed as a future scope** to explore how universities assign deans on a rotational / round robin basis |
| COURSE | FOREIGN KEY (DEPT\_ID) REFERENCES DEPARTMENT(DEPT\_ID) ON DELETE SET NULL ON UPDATE CASCADE | 1. Referential integrity between the *dept\_id* of the **course** table and the primary key *dept\_id* of the parent table **department**. 2. Changes in the department table are reflected in the child course table. 3. Prevents the case of orphaned course records who have no linkage to any of the parent departments. |
| FOREIGN KEY (INSTRUCTOR\_ID) REFERENCES INSTRUCTOR(EMP\_ID) ON DELETE SET NULL ON UPDATE CASCADE | 1. Referential integrity between the *instructor\_id* of the **course** table and the primary key *emp\_id* of the **instructor** table. 2. Changes in the instructor table are reflected in the course table. 3. Ensures every instructor is associated with a course being taught. |
| FEES | FOREIGN KEY (STUDENT\_ID) REFERENCES STUDENT(IMMA\_ID) ON DELETE SET NULL ON UPDATE CASCADE | 1. Referential integrity between the *student\_id* of the **fees** table and the primary key *imma\_id* of the parent **student** table. 2. Changes in the student table are reflected in the child fees table. 3. Prevents the case of orphaned fees records who have no linkage to any of the parent student matriculations. |
| ENROLLMENT | FOREIGN KEY (IMMA\_ID) REFERENCES STUDENT(IMMA\_ID) ON DELETE SET NULL ON UPDATE CASCADE | 1. Referential integrity between the *imma\_id* of the **enrollment** table and the primary key *imma\_id* of the **student** parent table. 2. Changes in the student table are reflected in the child fees table. 3. Prevents the case of orphaned enrollment records who have no linkage to any of the parent student matriculations. |
| FOREIGN KEY (DEPT\_ID) REFERENCES DEPARTMENT(DEPT\_ID) ON DELETE SET NULL ON UPDATE CASCADE | 1. Referential integrity between the *dept\_id* of the **enrollment** table and the primary key *dept\_id* of the **department** table. 2. Changes in the department table are reflected in the enrollment table. 3. Ensures every enrolled student is associated with a department. |
| FOREIGN KEY (COURSE\_ID) REFERENCES COURSE(COURSE\_ID) ON DELETE SET NULL ON UPDATE CASCADE | 1. Referential integrity between the *course\_id* of the **enrollment** table and the primary key *course\_id* of the **course** table. 2. Changes in the course table are reflected in the enrollment table. 3. Ensures every enrolled student is associated with a course. |
| SUBJECT | FOREIGN KEY (STUDENT\_ID) REFERENCES STUDENT(IMMA\_ID)  ON DELETE SET NULL ON UPDATE CASCADE | 1. Referential integrity between the *student\_id* of the **subject** table and the primary key *imma\_id* of the **student** parent table. 2. Changes in the student table are reflected in the child subject table. 3. Prevents the case of orphaned subject records who have no linkage to any of the parent student matriculations. |
| FOREIGN KEY (COURSE\_ID) REFERENCES COURSE(COURSE\_ID)  ON DELETE SET NULL ON UPDATE CASCADE | 1. Referential integrity between the *course\_id* of the **subject** table and the primary key *course\_id* of the **course** table. 2. Changes in the course table are reflected in the subject table. 3. Ensures every subject that is inserted against grades for an enrolled student is associated with a course. |
| GRADES | FOREIGN KEY (STUDENT\_ID) REFERENCES STUDENT(IMMA\_ID)  ON DELETE SET NULL ON UPDATE CASCADE | 1. Referential integrity between the *student\_id* of the **grades** table and the primary key *imma\_id* of the **student** parent table. 2. Changes in the student table are reflected in the child grades table. 3. Prevents the case of orphaned grade records who have no linkage to any of the parent student matriculations. 4. **Future scope** is to improve on the decomposition to make them 3NF. |
|  | FOREIGN KEY (COURSE\_ID) REFERENCES COURSE(COURSE\_ID)  ON DELETE SET NULL ON UPDATE CASCADE | 1. Referential integrity between the *course\_id* of the **grades** table and the primary key *course\_id* of the **course** table. 2. Changes in the course table are reflected in the grades table. 3. Ensures every grade that is inserted against a subject for an enrolled student is associated with a course. 4. **Future scope** is to improve on the decomposition to make them 3NF. |

# **Use Cases**

## **Use Cases**

The use cases below depict primarily around how administrators can manage the entire ecosystem of the UMS. These use cases form the superset of all the other use cases described above in the UML code.

### **4.1.1 Consolidated UML Diagram for all primary use cases**

**A diagram of a college

Description automatically generated**

The above diagram illustrates a consolidated UML use case for the primary users: administrators, instructors, and students.

1. **Administrators:** As previously discussed, administrators have the highest level of access control. They are responsible for managing the administration and operations of the database.
2. **Students:** Students can use the GUI to view their details and results. As a future scope with the help of row level security, we will also enable the students to be able to pay the fees which pertains to their unique matriculation ids.
3. **Instructors:** Instructors can use the GUI to view their details, enter grades for students in their subjects, and view their results.

### **4.1.2 Future Proposal – Restricted access to Parents / Guardians**

**A diagram of a person with text

Description automatically generated**

* A potential future use case could be included for guardians or parents to view restricted data, such as their ward's grades and fee status.
* **Current Access:** Currently, guardians can access the Student UMS via their ward’s credentials, eliminating the need for additional database resources and overhead.
* **GDPR Compliance and Future Scope:** To comply with GDPR regulations, we propose offering guardians the option to consent to saving their data and creating a Guardian account. This would help students manage financial stability and facilitate the payment of tuition and semester fees.

## **4.2 Description of the Graphical User Interface**

The Graphical User Interface (GUI) is built using **Java Swing[2]**, a powerful toolkit for creating user-friendly interfaces. A ***Person*** interface is defined to include basic data requirements like first name, last name, contact details, and address. This interface is implemented by two classes, ***Student*** and ***Instructor***, each with additional fields relevant to their roles, such as room numbers, cabins, courses, and departments.

Java Swing was chosen for its ease of use in creating forms[2], which makes collecting and managing user information straightforward. The design ensures that all classes are neatly organized into packages, promoting clean and maintainable code.

User input collected via the Swing forms is stored in the HFT LIDA server database. This process uses the MySQL driver provided in Moodle to connect, store and retrieve the data for the project.

# **5. Transactions / Triggers**

## **5.1 Transactions**

### **5.1.1 DML Statements part of the stored procedure call**

These SQL queries are part of the second stored procedure which is called right before the GUI is started[2]. These update the database with static values for the administrator, department and course to cover the prerequisites for the use cases illustrated above. The administrator queries insert three admin users for the main database system to work, the department and the course queries pre-fill the database with static values of the departments and their respective courses which are then updated or aligned against instructors or students depending on their enrolment or registration. There are also cases of **complex** **multiple joins** in Transactions 3,6, and 9 respectively.

#### **5.1.1.1 Inserting values into the ADMINISTRATOR table**

/\*3.1 ADMINISTRATOR \*/  
  
INSERT INTO ADMINISTRATOR (FIRST\_NAME, LAST\_NAME, CONTACT) VALUES ('Ryan', 'Gosling', '+49 15511234567');  
INSERT INTO ADMINISTRATOR (FIRST\_NAME, LAST\_NAME, CONTACT) VALUES ('Patrick', 'Bateman', '+49 15512345671');  
INSERT INTO ADMINISTRATOR (FIRST\_NAME, LAST\_NAME, CONTACT) VALUES ('Lou', 'Bloom', '+49 15513456712');

#### **5.1.1.2 Inserting values into the DEPARTMENT table**

/\* 3.2 DEPARTMENT \*/  
  
INSERT INTO DEPARTMENT (DEPT\_ID, DEPT\_NAME, DEAN\_ID) VALUES (1,'Computer Science', NULL);  
INSERT INTO DEPARTMENT (DEPT\_ID, DEPT\_NAME, DEAN\_ID) VALUES (2,'Mathematics', NULL);  
INSERT INTO DEPARTMENT (DEPT\_ID, DEPT\_NAME, DEAN\_ID) VALUES (3,'Business', NULL);  
INSERT INTO DEPARTMENT (DEPT\_ID, DEPT\_NAME, DEAN\_ID) VALUES (4,'Building Physics', NULL);

#### **5.1.1.3 Inserting values into the COURSE table**

The ***course*** table is the child table for the ***department*** table as every course is linked to a parent department. The data integrity constraints do not allow orphaned courses to exist, hence, these values have been added to the initial call to pre-fill the database with specific courses to which students or instructors can be aligned with. The goal is to have every instructor and student linked to a department and its corresponding course ( or programme ).

/\* 3.3 COURSE \*/  
  
/\* Computer Science \*/  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (11, 1, NULL, 'Bachelor of Computer Science', 'bcs', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (12, 1, NULL, 'Bachelor of Augmented Reality', 'bar', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (13, 1, NULL, "Bachelor's degree in digitalization and information management", 'bdi', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (14, 1, NULL, 'Bachelor Surveying and Geoinformatics', 'bsg', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (15, 1, NULL, 'Master Photogrammetry and Geoinformatics', 'mpg', 90, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (16, 1, NULL, 'Master Software Technology', 'mst', 90, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (17, 1, NULL, 'Master Digital Processes and Technologies', 'mdp', 90, NULL, NULL);  
  
/\* Mathematics \*/  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (21, 2, NULL, 'Bachelor Applied Mathematics and AI', 'bam', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (22, 2, NULL, 'Master Mathematics', 'mam', 90, NULL, NULL);  
  
/\* Business \*/  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (31, 3, NULL, 'Bachelor of Business Administration', 'bba', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (32, 3, NULL, 'Business Administration International Business course', 'bai', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (33, 3, NULL, 'Bachelor of Infrastructure Management', 'bim', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (34, 3, NULL, 'Bachelor of Business Informatics', 'bbi', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (35, 3, NULL, 'Master General Management', 'mgm', 90, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (36, 3, NULL, 'Master Environmentally Oriented Logistics', 'meo', 90, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (37, 3, NULL, "Master of Business Psychology", 'mdb', 90, NULL, NULL);  
  
/\* Building Physics \*/  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (41, 4, NULL, 'Bachelor of Building Physics', 'bbp', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (42, 4, NULL, 'Bachelor of Climate Engineering', 'bce', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (43, 4, NULL, 'Master Building Physics', 'mbp', 90, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (44, 4, NULL, 'Master in Sustainable Energy Competence', 'mse', 90, NULL, NULL);

### **5.1.2 Transactions as per the use cases**

Administrators have the highest privileges to add/modify/delete student and instructor records, while also maintaining the basic student and instructor privileges to add/view grades against subjects and/or pay fees that automatically update the registration fields in the enrolment as “fees paid” because of the integrity constraints applied in the DDL statements of the initial stored procedure call. Commit and Rollback scenarios are also included in the following snippets[6][7][8][9] to take care of the transactions:

**Transaction #1: Adding Instructor Records**

The Administrators can add new instructors to the DBMS provided the contact follows the German Phone notation (used Regex) and the age is above 25 years.

The queries are committed only if they follow both the preliminary conditions, else the transactions are rolled back to avoid saving the incorrect query results. Necessary joins have also been taken care inside the SQL queries with appropriate data integrity constraints for the table COURSE which has a relationship with the table INSTRUCTOR via *instructor\_id* field.

**A screen shot of a computer

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Transaction #2: Updating / Removing Instructor Records**

The Administrators can add update instructors to the DBMS but only the contact (which again follows the German Phone notation) and the address. The queries are committed only if they follow the preliminary conditions, else the transactions are rolled back to avoid saving the incorrect query results. The Administrators can also remove the instructors but this transaction is permanent and it cannot be rolled back. However, the UI has been designed to give a warning dialogue before every transaction so that the transaction can be cancelled before it can begin.

A screen shot of a computer code

Description automatically generated A computer screen with text

Description automatically generated

A black screen with text

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer error

Description automatically generated

A screenshot of a computer

Description automatically generated

**Transaction #3: Viewing Instructor Records**

This is feature shared by the ADMINISTRATOR and INSTRUCTOR where instructor records can be viewed by both the parties. As a future scope, we are working on developing the row level security for the instructor access so that they can only view the data revolving around their *employee\_id*. We are also building on improving the UI of the table to improve readability[11][12]. Currently, the admins and the instructors have the option of either viewing or printing the required data to their desktop which can be useful for analysis. As there are no insertions or updations, there are no COMMIT or ROLLBACK scenarios applicable here. These queries **use multiple joins** and each join maintains data referential integrity as they have been predefined in the DDL statements.

A computer screen with green text

Description automatically generated

A screenshot of a computer

Description automatically generated

**Transaction #4: Inserting Student and Enrolment Records**

The Administrators can add new students to the DBMS provided the contact follows the German Phone notation (used Regex) and the age is above 16 years.

The queries are committed only if they follow both the preliminary conditions, else the transactions are rolled back to avoid saving the incorrect query results. Necessary joins have also been taken care inside the SQL queries with appropriate data integrity constraints for the table COURSE which has a relationship with the table STUDENT via *imma\_id* field (for matriculation).

A black screen with green text

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Transaction #5: Updating / Deregistering Student Records**

The Administrators can add update student records to the DBMS but only the contact (which again follows the German Phone notation) and the address. The queries are committed only if they follow the preliminary conditions, else the transactions are rolled back to avoid saving the incorrect query results. The Administrators can also remove the students (called as deregistration) but this transaction is permanent and it cannot be rolled back. However, the UI has been designed to give a warning dialogue before every transaction so that the transaction can be cancelled before it can begin.

A computer screen shot of text

Description automatically generatedA computer screen with text and numbers

Description automatically generated

A screenshot of a computer

Description automatically generated

A black screen with text on it

Description automatically generated

A screenshot of a computer

Description automatically generated

A close-up of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

**Transaction #6: Viewing Student Records**

This is feature shared by the ADMINISTRATOR and STUDENTs where student records can be viewed by both the parties. As a future scope, we are working on developing the row level security for the student access so that they can only view the data revolving around their *imma\_id*. We are also building on improving the UI of the table to improve readability[11][12]. Currently, the admins and the students have the option of either viewing or printing the required data to their desktop which can be useful for analysis. As there are no insertions or updations, there are **NO COMMIT or ROLLBACK scenarios applicable here**. These queries **use multiple joins** and each join maintains data referential integrity as they have been predefined in the DDL statements.

A computer screen with green text

Description automatically generated

A screenshot of a computer

Description automatically generated

**Transaction #7: Updating Student Fee Status**

The Administrators can add update student fee status to the DBMS. The tuition fees are applicable only to the international students, while semester fees are common to all. The form also gives the option of selecting the D-ticket (Deutschland Fahrkarte) at a discounted price to the students. The total amount can then be paid by simply clicking a button. The queries **are committed only** if the fee status for the selected student shows as “NOT PAID”, **else the transactions are rolled back** to avoid saving the incorrect query results. There is also an option for the administer to manually reset the paid fees status to “NOT PAID”.

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer screen

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

**Transaction #8: Inserting Grades for a student by an instructor**

The Administrators and Instructors can insert grades to the students in the DBMS but **only once (commit only once)**. Rollback happens automatically if the system detects a case of multiple grades assigned for a semester. Necessary joins have also been taken care inside the SQL queries with appropriate data integrity constraints for the tables SUBJECT and GRADES which have a relationship with the table STUDENT via *imma\_id* field (for matriculation).

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Transaction #9: Viewing Subjects and Grades Records**

This is feature shared by the ADMINISTRATOR and STUDENTS where student records can be viewed by both the parties. As a future scope, we are working on developing the row level security for the student access so that they can only view the data revolving around their *imma\_id*. We are also building on improving the UI of the table to improve readability[11][12]. Currently, the admins and the students have the option of either viewing or printing the required data to their desktop which can be useful for analysis. As there are no insertions or updations, there are **NO COMMIT or ROLLBACK scenarios applicable here**. These queries **use multiple joins** and each join maintains data referential integrity as they have been predefined in the DDL statements.

String query = """  
 SELECT DISTINCT S.FIRST\_NAME, S.LAST\_NAME, C.COURSE\_NAME, SUB.\*, G.\*  
 FROM  
 SUBJECT SUB  
 LEFT JOIN GRADES G  
 ON SUB.STUDENT\_ID = G.STUDENT\_ID and SUB.COURSE\_ID = G.COURSE\_ID and SUB.SEMESTER = G.SEMESTER  
 LEFT JOIN STUDENT S  
 ON S.IMMA\_ID = SUB.STUDENT\_ID  
 LEFT JOIN ENROLLMENT E  
 ON S.IMMA\_ID = E.IMMA\_ID  
 LEFT JOIN COURSE C  
 ON E.COURSE\_ID = C.COURSE\_ID  
 """;

A screenshot of a computer

Description automatically generated

## **5.2 Triggers**

We have successfully implemented **4 triggers** which are run between the two stored procedures[3][4][10]. The first three triggers automatically take care of creating user\_ids, passwords, and email\_ids based on the regex pattern and inserts them dynamically into the respective admin, instructor and student records. The last trigger updates the tuition fees based on the international status of the new student enrolled into the student database – € 1500,00 in case they are an international student, else € 0. This is useful in order to calculate the total fees and also while paying the total fees which includes the tuition fees calculated dynamically.

### **5.2.1 Trigger for the ADMINISTRATOR**

The trigger automatically fires when a new administrator is added to create user\_ids, passwords and email based on the regex (firstName.lastName@hft-europa.de).

/\* 1. ADMINISTRATOR \*/  
DROP TRIGGER IF EXISTS trigger\_admin;  
  
DELIMITER $$  
  
CREATE TRIGGER trigger\_admin  
BEFORE INSERT ON ADMINISTRATOR  
FOR EACH ROW  
BEGIN  
SET NEW.PSWD = CONCAT(LEFT(UUID(), 8),LEFT(UUID(), 8));  
SET NEW.EMAIL = CONCAT(NEW.FIRST\_NAME, '.', NEW.LAST\_NAME, '@hft-europa.de');  
SET NEW.USER\_ID = CONCAT(RIGHT(YEAR(CURRENT\_DATE()), 1),IF(MONTH(CURRENT\_DATE()) <= 6, '1', '2'),LEFT(NEW.LAST\_NAME, 2),LEFT(NEW.FIRST\_NAME, 2));  
END;  
$$  
  
DELIMITER ;

### **5.2.2 Trigger for the INSTRUCTOR**

The trigger automatically fires when a new instructor is added to create user\_ids, passwords and email based on the regex (firstName.lastName@hft-europa.de).

/\* 2. INSTRUCTOR \*/  
DROP TRIGGER IF EXISTS trigger\_instructor;  
  
DELIMITER $$  
  
CREATE TRIGGER trigger\_instructor  
BEFORE INSERT ON INSTRUCTOR  
FOR EACH ROW  
BEGIN  
SET NEW.EMAIL = CONCAT(NEW.FIRST\_NAME, '.', NEW.LAST\_NAME, '@hft-europa.de');  
SET NEW.USER\_ID = CONCAT(RIGHT(YEAR(CURRENT\_DATE()), 1),IF(MONTH(CURRENT\_DATE()) <= 6, '1', '2'),LEFT(NEW.LAST\_NAME, 2),LEFT(NEW.FIRST\_NAME, 2));  
SET NEW.PSWD = CONCAT(LEFT(UUID(), 8),LEFT(UUID(), 8));  
END;  
$$  
  
DELIMITER ;

### **5.2.3 Trigger for the STUDENT**

The trigger automatically fires when a new student is added to create user\_ids, passwords and email based on the regex (firstName.lastName@hft-europa.de).

/\* 3. STUDENT \*/  
DROP TRIGGER IF EXISTS trigger\_student;  
  
DELIMITER $$  
  
CREATE TRIGGER trigger\_student  
BEFORE INSERT ON STUDENT  
FOR EACH ROW  
BEGIN  
SET NEW.EMAIL = CONCAT(NEW.FIRST\_NAME, '.', NEW.LAST\_NAME, '@hft-europa.de');  
SET NEW.USER\_ID = CONCAT(RIGHT(YEAR(CURRENT\_DATE()), 1),IF(MONTH(CURRENT\_DATE()) <= 6, '1', '2'),LEFT(NEW.LAST\_NAME, 2),LEFT(NEW.FIRST\_NAME, 2));  
SET NEW.PSWD = CONCAT(LEFT(UUID(), 8),LEFT(UUID(), 8));  
END;  
$$  
  
DELIMITER ;

### **5.2.4 Trigger for the FEES**

The trigger automatically fires when a new student is added to the database. The goal is to set the tuition fees to either 0 or 1500 depending on the international status of the student whilst also preparing the fields for the other fee variables necessary for the fee payment.

/\* 4. FEES \*/  
DROP TRIGGER IF EXISTS trigger\_fees\_insert;  
  
DELIMITER $$  
  
CREATE TRIGGER trigger\_fees\_insert  
AFTER INSERT ON STUDENT  
FOR EACH ROW  
BEGIN  
INSERT INTO FEES (STUDENT\_ID, TUITION\_FEES, SEMESTER\_FEES, D\_TICKET, RESEARCH\_VARIABLE, FEES\_PAID)  
VALUES (NEW.IMMA\_ID, CASE WHEN UPPER(NEW.INTERNATIONAL\_STUDENT) = 'YES' THEN 1500 ELSE 0 END, 200, 0, 0, 'NO');  
END;  
$$  
  
DELIMITER ;

## **5.3 Stored Procedures**

We have successfully created **2 stored procedures** – the first one implements all the DDL statements and the second one implements all the DML statements[4]. The order of execution is as follows:

1. Stored procedure 1 for the DDL
2. Triggers
3. Stored procedure 2 for the DML

The stored procedures can be called by simply following the format[4]:

A black screen with white text

Description automatically generated

### **5.3.1 Stored Procedure for the DDL Statements**

/\* 1. STORED PROCEDURE \*/  
  
/\* DDL STATEMENTS \*/  
  
DROP PROCEDURE IF EXISTS UMS\_Schema\_Creation;  
  
DELIMITER $$  
  
CREATE PROCEDURE UMS\_Schema\_Creation()  
BEGIN  
  
/\* DDL STATEMENTS \*/  
  
/\* 1. DROP TABLES \*/  
  
/\*1.1\*/  
DROP TABLE IF EXISTS GRADES;  
DROP TABLE IF EXISTS SUBJECT;  
  
/\*1.2\*/  
DROP TABLE IF EXISTS FEES;  
DROP TABLE IF EXISTS ENROLLMENT;  
  
/\*1.3\*/  
DROP TABLE IF EXISTS COURSE;  
DROP table IF EXISTS DEPARTMENT;  
  
/\*1.4\*/  
DROP TABLE IF EXISTS INSTRUCTOR;  
DROP TABLE IF EXISTS STUDENT;  
  
/\*1.5\*/  
DROP TABLE IF EXISTS ADMINISTRATOR;  
  
/\* 2. SCHEMA CREATION \*/  
  
/\*2.1 ADMINISTRATOR TABLE \*/  
CREATE TABLE ADMINISTRATOR(  
 USER\_ID *VARCHAR*(16) PRIMARY KEY,  
 PSWD *VARCHAR*(16),  
 FIRST\_NAME *VARCHAR*(255),  
 LAST\_NAME *VARCHAR*(255),  
 CONTACT *VARCHAR*(17),  
 EMAIL *VARCHAR*(255) /\* Trigger to generate regex email -> firstname.lastname@hft-europa.com\*/  
);  
  
/\*2.2 INSTRUCTOR TABLE \*/  
CREATE TABLE INSTRUCTOR(  
 USER\_ID *VARCHAR*(16),  
 PSWD *VARCHAR*(16),  
 FIRST\_NAME *VARCHAR*(255),  
 LAST\_NAME *VARCHAR*(255),  
 CONTACT *VARCHAR*(15),  
 EMAIL *VARCHAR*(50), /\* Trigger to generate regex email -> firstname.lastname@hft-europa.com\*/  
 DOB *VARCHAR*(255),  
 HOME\_ADDRESS *VARCHAR*(255),  
 HFT\_CABIN *VARCHAR*(255),  
 EMP\_ID *VARCHAR*(16) PRIMARY KEY  
);  
  
/\*2.3 STUDENT TABLE \*/  
CREATE TABLE STUDENT(  
 USER\_ID *VARCHAR*(16),  
 PSWD *VARCHAR*(16),  
 FIRST\_NAME *VARCHAR*(255),  
 LAST\_NAME *VARCHAR*(255),  
 CONTACT *VARCHAR*(15),  
 EMAIL *VARCHAR*(50),  
 DOB *VARCHAR*(255),  
 HOME\_ADDRESS *VARCHAR*(255),  
 GRADE\_XII *VARCHAR*(16),  
 INTERNATIONAL\_STUDENT *VARCHAR*(16),  
 IMMA\_ID *VARCHAR*(16) PRIMARY KEY  
);  
  
/\*2.4 DEPARTMENT TABLE \*/  
CREATE TABLE DEPARTMENT(  
 DEPT\_ID *INT* PRIMARY KEY,  
 DEAN\_ID *VARCHAR*(16),  
 DEPT\_NAME *VARCHAR*(255),  
 FOREIGN KEY (DEAN\_ID) REFERENCES INSTRUCTOR(EMP\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);  
  
/\*2.5 COURSE TABLE \*/  
CREATE TABLE COURSE(  
 COURSE\_ID *INT* PRIMARY KEY,  
 DEPT\_ID *INT*,  
 INSTRUCTOR\_ID *VARCHAR*(16),  
 COURSE\_NAME *VARCHAR*(255),  
 COURSE\_CODE *VARCHAR*(10), /\* Master Software Technology -> mst \*/  
 CREDITS *INT*,  
 BUILDING\_NO *INT*,  
 ROOM\_NO *INT*,  
 FOREIGN KEY (DEPT\_ID) REFERENCES DEPARTMENT(DEPT\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE,  
 FOREIGN KEY (INSTRUCTOR\_ID) REFERENCES INSTRUCTOR(EMP\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);  
  
/\*2.6 FEES \*/  
CREATE TABLE FEES(  
 STUDENT\_ID *VARCHAR*(16),  
 TUITION\_FEES *VARCHAR*(16), /\* 1500 Euros for International Students \*/  
 SEMESTER\_FEES *VARCHAR*(16), /\* 200 Euros for all \*/  
 D\_TICKET *VARCHAR*(16), /\* Optional: 360 Euros for all \*/  
 RESEARCH\_VARIABLE *VARCHAR*(16), /\* Optional: Variable \*/  
 FEES\_PAID *VARCHAR*(3),  
 FOREIGN KEY (STUDENT\_ID) REFERENCES STUDENT(IMMA\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);  
  
/\*2.7 ENROLLMENT \*/  
CREATE TABLE ENROLLMENT(  
 IMMA\_ID *VARCHAR*(16),  
 DEPT\_ID *INT*,  
 COURSE\_ID *INT*,  
 IMMA\_DATE *VARCHAR*(255),  
 IMMA\_STATUS *VARCHAR*(100),  
 FOREIGN KEY (IMMA\_ID) REFERENCES STUDENT(IMMA\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE,  
 FOREIGN KEY (DEPT\_ID) REFERENCES DEPARTMENT(DEPT\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE,  
 FOREIGN KEY (COURSE\_ID) REFERENCES COURSE(COURSE\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);  
  
/\*2.8 SUBJECT TABLE \*/  
CREATE TABLE SUBJECT(  
 STUDENT\_ID *VARCHAR*(16),  
 COURSE\_ID *INT*,  
 SEMESTER *INT*,  
 SUBJECT1 *VARCHAR*(255),  
 SUBJECT2 *VARCHAR*(255),  
 SUBJECT3 *VARCHAR*(255),  
 SUBJECT4 *VARCHAR*(255),  
 SUBJECT5 *VARCHAR*(255),  
 ELECTIVE1 *VARCHAR*(255),  
 FOREIGN KEY (STUDENT\_ID) REFERENCES STUDENT(IMMA\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE,  
 FOREIGN KEY (COURSE\_ID) REFERENCES COURSE(COURSE\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);  
  
/\*2.9 GRADES TABLE \*/  
CREATE TABLE GRADES(  
 STUDENT\_ID *VARCHAR*(16),  
 COURSE\_ID *INT*,  
 SEMESTER *INT*,  
 GRADE1 *VARCHAR*(8),  
 GRADE2 *VARCHAR*(8),  
 GRADE3 *VARCHAR*(8),  
 GRADE4 *VARCHAR*(8),  
 GRADE5 *VARCHAR*(8),  
 GRADE6 *VARCHAR*(8),  
 FOREIGN KEY (STUDENT\_ID) REFERENCES STUDENT(IMMA\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE,  
 FOREIGN KEY (COURSE\_ID) REFERENCES COURSE(COURSE\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);  
  
END;  
  
$$  
  
DELIMITER ;

### **5.3.2 Stored Procedure for the DML Statements**

/\* 3. STORED PROCEDURE \*/  
  
/\* DML STATEMENTS \*/  
  
DROP PROCEDURE IF EXISTS UML\_DML\_Statements;  
  
DELIMITER $$  
  
CREATE PROCEDURE UML\_DML\_Statements()  
BEGIN  
  
/\* DML STATEMENTS \*/  
  
/\*3.1 ADMINISTRATOR \*/  
  
INSERT INTO ADMINISTRATOR (FIRST\_NAME, LAST\_NAME, CONTACT) VALUES ('Ryan', 'Gosling', '+49 15511234567');  
INSERT INTO ADMINISTRATOR (FIRST\_NAME, LAST\_NAME, CONTACT) VALUES ('Patrick', 'Bateman', '+49 15512345671');  
INSERT INTO ADMINISTRATOR (FIRST\_NAME, LAST\_NAME, CONTACT) VALUES ('Lou', 'Bloom', '+49 15513456712');  
  
/\* 3.2 DEPARTMENT \*/  
  
INSERT INTO DEPARTMENT (DEPT\_ID, DEPT\_NAME, DEAN\_ID) VALUES (1,'Computer Science', NULL);  
INSERT INTO DEPARTMENT (DEPT\_ID, DEPT\_NAME, DEAN\_ID) VALUES (2,'Mathematics', NULL);  
INSERT INTO DEPARTMENT (DEPT\_ID, DEPT\_NAME, DEAN\_ID) VALUES (3,'Business', NULL);  
INSERT INTO DEPARTMENT (DEPT\_ID, DEPT\_NAME, DEAN\_ID) VALUES (4,'Building Physics', NULL);  
  
/\* 3.3 COURSE \*/  
  
/\* Computer Science \*/  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (11, 1, NULL, 'Bachelor of Computer Science', 'bcs', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (12, 1, NULL, 'Bachelor of Augmented Reality', 'bar', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (13, 1, NULL, "Bachelor's degree in digitalization and information management", 'bdi', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (14, 1, NULL, 'Bachelor Surveying and Geoinformatics', 'bsg', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (15, 1, NULL, 'Master Photogrammetry and Geoinformatics', 'mpg', 90, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (16, 1, NULL, 'Master Software Technology', 'mst', 90, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (17, 1, NULL, 'Master Digital Processes and Technologies', 'mdp', 90, NULL, NULL);  
  
/\* Mathematics \*/  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (21, 2, NULL, 'Bachelor Applied Mathematics and AI', 'bam', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (22, 2, NULL, 'Master Mathematics', 'mam', 90, NULL, NULL);  
  
/\* Business \*/  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (31, 3, NULL, 'Bachelor of Business Administration', 'bba', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (32, 3, NULL, 'Business Administration International Business course', 'bai', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (33, 3, NULL, 'Bachelor of Infrastructure Management', 'bim', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (34, 3, NULL, 'Bachelor of Business Informatics', 'bbi', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (35, 3, NULL, 'Master General Management', 'mgm', 90, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (36, 3, NULL, 'Master Environmentally Oriented Logistics', 'meo', 90, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (37, 3, NULL, "Master of Business Psychology", 'mdb', 90, NULL, NULL);  
  
/\* Building Physics \*/  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (41, 4, NULL, 'Bachelor of Building Physics', 'bbp', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (42, 4, NULL, 'Bachelor of Climate Engineering', 'bce', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (43, 4, NULL, 'Master Building Physics', 'mbp', 90, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (44, 4, NULL, 'Master in Sustainable Energy Competence', 'mse', 90, NULL, NULL);  
  
END;  
  
$$  
  
DELIMITER ;

## **5.4 Declaration about used AI Tools**

While we have used references and have listed them in the coming chapter with citations, we declare that we have usedChatGPT for the project for the following reasons with justifications:

|  |  |  |
| --- | --- | --- |
| Category | Use Case | Justification |
| Stored Procedure  Delimitters | SQL Workbench/J testing | 1. We used ChatGPT to understand a sample stored procedure on SQL Workbench/J as it was our primary GUI for connecting to the LIDA server. 2. No codes were copied as the tool was **only** used to understand the MySQL Syntax structure as it varies for different softwares used. |
| Before vs After Triggers | SQL Workbench/J testing | 1. We used ChatGPT to understand the differences between ***before*** *and* ***after*** triggers. 2. No codes were copied as the tool was **only** used to understand the MySQL Snytax structure for delimitters and before triggers as it varies for different softwares used. |
| DDL Statement | **ON DELETE SET NULL ON UPDATE CASCADE** | 1. We used ChatGPT to **understand this code** (which was alsoon the moodle). 2. We wanted to understand how this line affects our referential integrities and we **only** used the tool to understand the impact on a general DDL statement. |

# **6. List of References ( Other than class notes on Moodle )**

[1] <https://www.geeksforgeeks.org/student-information-management-system/>

[2] <https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html>

[3] <https://www.tutorialspoint.com/mysql/mysql_create_trigger.htm>

[4] <https://www.tutorialspoint.com/mysql/mysql_create_procedure.htm>

[5] <https://www.tutorialjinni.com/university-management-system-erd.html>

[6] <https://stackoverflow.com/questions/65213422/is-it-necessary-to-run-rollback-in-mysql>

[7] <https://stackoverflow.com/questions/61534177/is-it-bad-to-call-rollback-without-any-start-transaction>

[8] <https://www.geeksforgeeks.org/difference-between-commit-and-rollback-in-sql/>

[9] <https://www.javatpoint.com/transaction-management-in-jdbc>

[10] <https://www.javatpoint.com/trigger-in-sql>

[11] <https://stackoverflow.com/questions/33981939/jtable-not-visible-when-added-to-jpanel>

[12] <https://stackoverflow.com/questions/14789884/make-scrollable-table>

# **7. Appendix**

## **7.1 Source Code of the Application and User Interface**

### 7.1.1 Custom Connection Class – Conn.java

package connections;  
  
// General Libraries  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
import java.sql.Statement;  
  
public class Conn {  
 public Connection conn;  
 public Statement stmt;  
  
 public Conn(){  
 try{  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 conn = DriverManager.*getConnection*(  
 "jdbc:mysql://193.196.143.168/dk4s\_41kusa1mst", "dk4s\_41kusa1mst", "bitteaendern"  
 );  
  
 stmt = conn.createStatement();  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
}

### 7.1.2 Mapping of Department to Course – DepartmentToCourseMapping.java

package connections;  
  
*/\*\*  
 @author Sanjay Prabhu Kunjibettu  
 @author Tanay Khilare  
 \*/*// General Classes  
import java.sql.ResultSet;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
  
// Importing Custom Classes  
import connections.Conn;  
  
public class DeptToCourseMapping {  
 HashMap<String, List<String>> deptToCourse;  
  
 public DeptToCourseMapping(){  
 deptToCourse = new HashMap<>();  
  
 try{  
 Conn c = new Conn();  
 String query = """  
 SELECT dept\_name, course\_name  
 FROM DEPARTMENT D  
 LEFT JOIN COURSE C  
 ON D.dept\_id = C.dept\_id;  
 """;  
  
 ResultSet rs = c.stmt.executeQuery(query);  
 while(rs.next()){  
 String dept = rs.getString("dept\_name");  
 String course = rs.getString("course\_name");  
  
 // Custom JAVA method to maintain dynamic mapping of dept-course  
 insert(dept, course);  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
  
 public void insert(String key, String value){  
 // 1. Check if the key already exists  
 List<String> allValues = deptToCourse.get(key);  
  
 // 2. If the key doesn't exist, create a new mapping  
 if(allValues == null){  
 allValues = new ArrayList<>();  
 deptToCourse.put(key, allValues);  
 }  
  
 // 3. Add the values to the new/existing list  
 allValues.add(value);  
 }  
  
 public HashMap getDeptToCourseMapping(){  
 return this.deptToCourse;  
 }  
  
 public String[] getDeptList(){  
 return this.deptToCourse.keySet().toArray(new String[0]);  
 }  
}

### 7.1.3 Initial Pop Up Screen – Splash.java

package de.stuttgart.europa;  
  
// Splash Screen is the initial screen that appears for 10 miliseconds  
// It gives the system time to load the configurations and U.I.  
  
*/\*\*  
 @author Sanjay Prabhu Kunjibettu  
 @author Tanay Khilare  
 \*/*// General Libraries  
import javax.swing.\*;  
import java.awt.\*;  
  
// Importing Custom Classes  
import admin.AdminLogin;  
  
public class SplashScreen extends JFrame implements Runnable {  
 Thread t;  
  
 SplashScreen(){  
 ImageIcon imageIcon1 = new ImageIcon(ClassLoader.*getSystemResource*("images/splash\_screen\_university.jpg"));  
 Image image1 = imageIcon1.getImage().getScaledInstance(1000, 500, Image.*SCALE\_DEFAULT*);  
 ImageIcon imageIcon2 = new ImageIcon(image1);  
  
 JLabel lblImage = new JLabel(imageIcon2);  
 this.add(lblImage);  
  
 // Start Thread  
 t = new Thread(this);  
 t.start();  
  
 try {  
 Thread.*sleep*(10);  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
  
 // JFrame Configurations  
 this.setTitle("Hochschule für Wissenschaft Europa");  
 this.setSize(1000,500);  
 this.setVisible(true);  
 this.setLocation(300,200);  
  
 }  
  
 @Override  
 public void run(){  
 try{  
 Thread.*sleep*(5000);  
 setVisible(false);  
 new AdminLogin();  
 }  
 catch(Exception e){  
 e.printStackTrace();  
 }  
 }  
  
 public static void main(String[] args) {  
 new SplashScreen();  
 }  
}

### 7.1.4 Login Page for Admin – AdminLogin.java

package admin;  
  
// Page is followed after SplashScreen.java  
// Admin Login Page  
  
*/\*\*  
 @author Sanjay Prabhu Kunjibettu  
 @author Tanay Khilare  
 \*/*// General Libraries  
import java.awt.\*;  
import javax.swing.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
  
// Importing Custom Classes  
import connections.Conn;  
  
public class AdminLogin extends JFrame implements ActionListener {  
 JLabel lblUserName, lblPassword, lblBG;  
 JTextField tfUserName;  
 JPasswordField pfPassword;  
 JButton btnLogin;  
  
 ImageIcon icon1, iconBG, icon2, iconAdmin;  
 Image imgBG, imgAdmin;  
  
 public AdminLogin(){  
  
 /\* 1. Admin User Name \*/  
  
 // 1.1 Label  
 lblUserName = new JLabel("Username: ");  
 lblUserName.setBounds(70, 20, 100, 20);  
 this.add(lblUserName);  
  
 // 1.2 Text Field  
 tfUserName = new JTextField();  
 tfUserName.setBounds(180, 20, 150, 20);  
 this.add(tfUserName);  
  
 /\* 2. Admin Password \*/  
  
 // 2.1 Label  
 lblPassword = new JLabel("Password: ");  
 lblPassword.setBounds(70, 70, 100, 20);  
 this.add(lblPassword);  
  
 // 1.2 Text Field  
 pfPassword = new JPasswordField();  
 pfPassword.setBounds(180, 70, 150, 20);  
 this.add(pfPassword);  
  
 /\* 3. Buttons \*/  
  
 // 3.1 Login Button  
 btnLogin = new JButton("Login");  
 btnLogin.setBounds(100, 140, 120, 30);  
 btnLogin.setBackground(Color.*black*);  
 btnLogin.setForeground(Color.*white*);  
 btnLogin.addActionListener(this);  
 this.add(btnLogin);  
  
 /\* 4. Images \*/  
  
 // 4.1 Background  
 icon1 = new ImageIcon(ClassLoader.*getSystemResource*("./images/admin\_login\_bg.jpg"));  
 imgBG = icon1.getImage().getScaledInstance(500, 350, Image.*SCALE\_DEFAULT*);  
 iconBG = new ImageIcon(imgBG);  
 lblBG = new JLabel(iconBG);  
 lblBG.setBounds(0, 0, 500, 350);  
 this.add(lblBG);  
  
 // JFrame Configurations  
 this.setTitle("Admin Login");  
 this.setSize(500,350);  
 this.setLocation(400,300);  
 this.setLayout(null);  
 this.setVisible(true);  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent ae){  
 if(ae.getSource() == btnLogin){  
 String adminUserName = tfUserName.getText();  
 String adminPassword = new String(pfPassword.getPassword());  
  
 System.*out*.println(adminUserName + " " + adminPassword);  
  
 String query = "SELECT \* FROM ADMINISTRATOR WHERE USER\_ID = ? AND PSWD = ?";  
  
 try{  
 Conn c = new Conn();  
 PreparedStatement preparedStatement = c.conn.prepareStatement(query);  
 preparedStatement.setString(1, adminUserName);  
 preparedStatement.setString(2, adminPassword);  
  
 ResultSet resultSet = preparedStatement.executeQuery();  
  
 if(resultSet.next()){  
 JOptionPane.*showMessageDialog*(null, "You have successfully logged in !");  
 setVisible(false);  
 new AdminMain();  
 }  
 else{  
 JOptionPane.*showMessageDialog*(null, "Invalid username or password !");  
 }  
 }catch(Exception e){  
 e.printStackTrace();  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 new AdminLogin();  
 }  
}

### 7.1.5 Admin Desktop / Main Page – AdminMain.java

package admin;  
  
*/\*\*  
 @author Sanjay Prabhu Kunjibettu  
 @author Tanay Khilare  
 \*/*// General Libraries  
import java.awt.\*;  
import javax.swing.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
// Importing Custom Classes  
import grades.\*;  
import instructor.\*;  
import student.\*;  
  
public class AdminMain extends JFrame implements ActionListener {  
 ImageIcon icon1, iconBG;  
 Image imgBG;  
 JLabel lblBG;  
  
 JMenuBar menuBar;  
 JMenu menuNew, menuView, menuUpdate, menuRemove, menuExam, menuFees, menuUtility, menuReset, menuExit;  
 JMenuItem newInstructorInfo, newStudentInfo, viewInstructorInfo, viewStudentInfo,  
 updateInstructorInfo, updateStudentInfo, removeInstructorInfo, removeStudentInfo,  
 examResults, examEnterMarks, feesStructure, feesForm, utilityCalculator, utilityNotepad,  
 exitMenuItem;  
  
 AdminMain(){  
 /\* 1. Images \*/  
  
 // 1.1 Desktop Background  
 icon1 = new ImageIcon(ClassLoader.*getSystemResource*("images/desktop\_bg.jpeg"));  
 imgBG = icon1.getImage().getScaledInstance(1540,750, Image.*SCALE\_DEFAULT*);  
 iconBG = new ImageIcon(imgBG);  
 lblBG = new JLabel(iconBG);  
 this.add(lblBG);  
  
 /\* 2. Menu Bar \*/  
  
 menuBar = new JMenuBar();  
  
 // 2.1 New  
 menuNew = new JMenu("Add");  
 menuNew.setForeground(Color.*BLACK*);  
 menuBar.add(menuNew);  
  
 // // 2.1.1 New Instructor Information  
 newInstructorInfo = new JMenuItem("New Instructor");  
 newInstructorInfo.setBackground(Color.*WHITE*);  
 newInstructorInfo.addActionListener(this);  
 menuNew.add(newInstructorInfo);  
  
 // // 2.1.2 New Student Information  
 newStudentInfo = new JMenuItem("New Student");  
 newStudentInfo.setBackground(Color.*WHITE*);  
 newStudentInfo.addActionListener(this);  
 menuNew.add(newStudentInfo);  
  
 // 2.2 View Existing  
 menuView = new JMenu("View");  
 menuView .setForeground(Color.*BLACK*);  
 menuBar.add(menuView );  
  
 // // 2.2.1 View Existing Instructor Information  
 viewInstructorInfo = new JMenuItem("Instructor Details");  
 viewInstructorInfo.setBackground(Color.*WHITE*);  
 viewInstructorInfo.addActionListener(this);  
 menuView .add(viewInstructorInfo);  
  
 // // 2.2.2 View Existing Student Information  
 viewStudentInfo = new JMenuItem("Student Details");  
 viewStudentInfo.setBackground(Color.*WHITE*);  
 viewStudentInfo.addActionListener(this);  
 menuView .add(viewStudentInfo);  
  
 // 2.3 Update  
 menuUpdate = new JMenu("Update");  
 menuUpdate.setForeground(Color.*BLACK*);  
 menuBar.add(menuUpdate);  
  
 // // 2.3.1 Update Instructor Information  
 updateInstructorInfo = new JMenuItem("Update Instructor");  
 updateInstructorInfo.setBackground(Color.*WHITE*);  
 updateInstructorInfo.addActionListener(this);  
 menuUpdate.add(updateInstructorInfo);  
  
 // // 2.3.2 Update Student Information  
 updateStudentInfo = new JMenuItem("Update Student");  
 updateStudentInfo.setBackground(Color.*WHITE*);  
 updateStudentInfo.addActionListener(this);  
 menuUpdate.add(updateStudentInfo);  
  
 // 2.4 Remove  
 menuRemove= new JMenu("Remove");  
 menuRemove.setForeground(Color.*BLACK*);  
 menuBar.add(menuRemove);  
  
 // // 2.4.1 Remove Instructor Information  
 removeInstructorInfo = new JMenuItem("Remove Instructor");  
 removeInstructorInfo.setBackground(Color.*WHITE*);  
 removeInstructorInfo.addActionListener(this);  
 menuRemove.add(removeInstructorInfo);  
  
 // // 2.3.2 Deregister Student Information  
 removeStudentInfo = new JMenuItem("Deregister Student");  
 removeStudentInfo.setBackground(Color.*WHITE*);  
 removeStudentInfo.addActionListener(this);  
 menuRemove.add(removeStudentInfo);  
  
 // 2.5 Examinations  
 menuExam = new JMenu("Examinations");  
 menuExam.setForeground(Color.*BLACK*);  
 menuBar.add(menuExam);  
  
 // // 2.5.1 View Results  
 examResults = new JMenuItem("View Results");  
 examResults.setBackground(Color.*WHITE*);  
 examResults.addActionListener(this);  
 menuExam.add(examResults);  
  
 // // 2.5.2 Add Marks  
 examEnterMarks = new JMenuItem("Enter Grades");  
 examEnterMarks.setBackground(Color.*WHITE*);  
 examEnterMarks.addActionListener(this);  
 menuExam.add(examEnterMarks);  
  
 // 2.6 Fees  
 menuFees = new JMenu("Fees");  
 menuFees.setForeground(Color.*BLACK*);  
 menuBar.add(menuFees);  
  
 // // 2.6.1 Fee Structure  
 feesStructure = new JMenuItem("Fee Structure");  
 feesStructure.setBackground(Color.*WHITE*);  
 feesStructure.addActionListener(this);  
 menuFees.add(feesStructure);  
  
 // // 2.6.2 Student Fee Form  
 feesForm = new JMenuItem("Student Form");  
 feesForm.setBackground(Color.*WHITE*);  
 feesForm.addActionListener(this);  
 menuFees.add(feesForm);  
  
 // 2.7 Utilities  
 menuUtility = new JMenu("Utility");  
 menuUtility.setForeground(Color.*BLACK*);  
 menuBar.add(menuUtility);  
  
 // // 2.7.1 Calculator  
 utilityCalculator = new JMenuItem("Standard Calculator");  
 utilityCalculator.setBackground(Color.*WHITE*);  
 utilityCalculator.addActionListener(this);  
 menuUtility.add(utilityCalculator);  
  
 // // 2.7.2 Notepad  
 utilityNotepad= new JMenuItem("Standard Notepad");  
 utilityNotepad.setBackground(Color.*WHITE*);  
 utilityNotepad.addActionListener(this);  
 menuUtility.add(utilityNotepad);  
  
 // 2.9 Exit  
 menuExit = new JMenu("Exit");  
 menuExit.setForeground(Color.*BLACK*);  
 menuBar.add(menuExit);  
  
 // // 2.9.1 Logout  
 exitMenuItem = new JMenuItem("Logout");  
 exitMenuItem.setBackground(Color.*WHITE*);  
 exitMenuItem.addActionListener(this);  
 menuExit.add(exitMenuItem);  
  
 /\* 3. JFrame Configurations \*/  
 this.setJMenuBar(menuBar);  
 this.setSize(1540, 850);  
 this.setVisible(true);  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent ae){  
 String selectedMenuItem = ae.getActionCommand();  
 switch(selectedMenuItem){  
 case "New Instructor":  
 new AddInstructor();  
 break;  
 case "New Student":  
 new AddStudent();  
 break;  
 case "Instructor Details":  
 new ViewInstructor();  
 break;  
 case "Student Details":  
 new ViewStudent();  
 break;  
 case "Update Instructor", "Remove Instructor":  
 new UpdateInstructor();  
 break;  
 case "Update Student", "Deregister Student":  
 new UpdateStudent();  
 break;  
 case "View Results":  
 new ViewGrades();  
 break;  
 case "Enter Grades":  
 new InsertGrades();  
 break;  
 case "Fee Structure":  
 new FeeStructure();  
 case "Student Form":  
 new FeesForm();  
 break;  
 case "Standard Calculator":  
 try{  
 Runtime.*getRuntime*().exec("calc.exe");  
 break;  
 }  
 catch(Exception e){  
 e.printStackTrace();  
 }  
 case "Standard Notepad":  
 try{  
 Runtime.*getRuntime*().exec("notepad.exe");  
 break;  
 }  
 catch(Exception e){  
 e.printStackTrace();  
 }  
 case "Logout":  
 System.*exit*(15);  
 }  
 }  
  
 public static void main(String[] args) {  
 new AdminMain();  
 }  
}

### 7.1.6 Person Interface – Person.java

package oop;  
  
// Interface for a person object  
// Useful to be used as a template to add more users  
  
public interface Person {  
  
 // Getters  
 public String getFirstName();  
 public String getLastName();  
 public String getContact();  
 public String getDOB();  
 public String getAddress();  
  
 // Setters  
 public void setFirstName(String firstName);  
 public void setLastName(String lastName);  
 public void setContact(String contact);  
 public void setDOB(String dob);  
 public void setAddress(String address);  
  
 // Custom Functions  
 public boolean checkDOB(int threshold);  
  
 public boolean checkContact(String contact);  
}

### 7.1.7 Instructor Class implementing Person Interface – Instructor.java

package oop;  
  
*/\*\*  
 @author Sanjay Prabhu Kunjibettu  
 @author Tanay Khilare  
 \*/*// General Libraries  
import java.time.LocalDate;  
import java.time.Period;  
import java.time.format.DateTimeFormatter;  
import java.time.format.DateTimeParseException;  
import java.util.Random;  
import java.util.regex.Pattern;  
  
// Custom Libraries  
import connections.Conn;  
  
public class Instructor implements Person{  
  
 // Instance Variables  
 private String firstName;  
 private String lastName;  
 private String contact;  
 private String dob;  
 private String address;  
 private String hftCabin;  
 private String empID;  
 private String courseName;  
 private String deptName;  
  
 // Non-param constructor  
 public Instructor(){}  
  
 // Parameterized constructor  
 public Instructor(String firstName, String lastName, String contact, String dateOfBirth, String address,  
 String hftCabin, String courseName, String deptName) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.contact = contact;  
 this.dob = dateOfBirth;  
 this.address = address;  
 this.hftCabin = hftCabin;  
 this.empID = getRandomEmpID();  
 this.courseName = courseName;  
 this.deptName = deptName;  
 }  
  
 // Getters  
  
 @Override  
 public String getFirstName() { return this.firstName; }  
  
 @Override  
 public String getLastName() { return this.lastName; }  
  
 @Override  
 public String getContact() { return this.contact; }  
  
 @Override  
 public String getDOB() { return this.dob; }  
  
 @Override  
 public String getAddress() { return this.address; }  
  
 public String getEmpID() { return this.empID; }  
  
 public String getHftCabin() { return this.hftCabin; }  
  
 public String getCourseName() { return this.courseName; }  
  
 public String getDeptName() { return this.deptName; }  
  
 // Setters  
  
 @Override  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 @Override  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
  
 @Override  
 public void setContact(String contact) {  
 this.contact = contact;  
 }  
  
 @Override  
 public void setDOB(String dob) {  
 this.dob = dob;  
 }  
  
 @Override  
 public void setAddress(String address) {  
 this.address = address;  
 }  
  
 public void setHftCabin(String hftCabin){  
 this.hftCabin = hftCabin;  
 }  
  
 public void setCourseName(String courseName){  
 this.courseName = courseName;  
 }  
  
 public void setDeptName(String deptName){  
 this.deptName = deptName;  
 }  
  
 // Custom Functions  
  
 public String getRandomEmpID(){  
 Random rnd = new Random();  
 int n = 100000 + rnd.nextInt(900000);  
 return String.*valueOf*(n);  
 }  
  
 @Override  
 public boolean checkDOB(int threshold){  
  
 // Check if the instructor is more than 25 years to be able to enroll  
  
 try {  
 DateTimeFormatter formatter = DateTimeFormatter.*ofPattern*("dd-MMM-yyyy");  
 LocalDate birthDate = LocalDate.*parse*(this.dob, formatter);  
 LocalDate currentDate = LocalDate.*now*();  
 Period age = Period.*between*(birthDate, currentDate);  
 return age.getYears() > threshold;  
 } catch (DateTimeParseException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 @Override  
 public boolean checkContact(String contact){  
 // German Phone Number Regex  
 String regex = "^(\\+49|0)[1-9][0-9]{1,14}$";  
  
 // Compile  
 Pattern pattern = Pattern.*compile*(regex);  
  
 // Match  
 return pattern.matcher(contact).matches();  
 }  
  
 public int insertIntoDB(Conn c, int threshold){  
 if(!firstName.isEmpty() && !lastName.isEmpty() && !contact.isEmpty() && !dob.isEmpty() && !address.isEmpty() && !hftCabin.isEmpty() && !empID.isEmpty()) {  
 try {  
 String query = """  
 INSERT INTO INSTRUCTOR (FIRST\_NAME, LAST\_NAME, CONTACT, DOB, HOME\_ADDRESS, HFT\_CABIN, EMP\_ID) VALUES ('"""  
 + firstName + "','"  
 + lastName + "','"  
 + contact + "','"  
 + dob + "','"  
 + address + "','"  
 + hftCabin + "','"  
 + empID + "')";  
  
 c.stmt.executeUpdate(query);  
  
 if (!checkDOB(threshold) || !checkContact(contact)) {  
 return -1; // values failed error  
 }  
 }  
 catch (Exception e) {  
 e.printStackTrace();  
 return -1;  
 }  
  
 return 1; // Insertion successful  
 }  
  
 return 0; // Values are missing  
 }  
  
 public int updateContactDB(Conn c, String empID, String newContact, String newAddress){  
 try{  
 if(!newContact.isEmpty()){  
 String query = """  
 UPDATE INSTRUCTOR  
 SET CONTACT = '  
 """ + newContact + "'"  
 +  
 """  
 WHERE EMP\_ID =  
 """  
 + empID + ";";  
 c.stmt.executeUpdate(query);  
  
 if(!checkContact(newContact)){  
 return -1; // rollback in the outside function  
 }  
 else{  
 c.conn.commit();  
 }  
 }  
  
 if(!newAddress.isEmpty()){  
 String query = """  
 UPDATE INSTRUCTOR  
 SET HOME\_ADDRESS = '  
 """ + newAddress + "'"  
 +  
 """  
 WHERE EMP\_ID =  
 """  
 + empID + ";";  
 c.stmt.executeUpdate(query);  
  
 c.conn.commit();  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 return -1;  
 }  
  
 return 1; // worked successfully  
 }  
}

### 7.1.8 Student Class implementing Person Interface – Student.java

package oop;  
  
*/\*\*  
 @author Sanjay Prabhu Kunjibettu  
 @author Tanay Khilare  
 \*/*// General Libraries  
import java.sql.ResultSet;  
import java.time.LocalDate;  
import java.time.Period;  
import java.time.format.DateTimeFormatter;  
import java.time.format.DateTimeParseException;  
import java.util.Random;  
import java.util.regex.Pattern;  
  
// Custom Libraries  
import connections.Conn;  
  
public class Student implements Person{  
  
 // Instance Variables  
 private String firstName;  
 private String lastName;  
 private String contact;  
 private String dob;  
 private String address;  
 private String gradeXII;  
 private String internationalStudent;  
 private String matID;  
 private String courseName;  
 private String deptName;  
  
 // Non-param Constructor  
 public Student(){}  
  
 // Parameterized constructor  
 public Student(String firstName, String lastName, String contact, String dateOfBirth, String address,  
 String gradeXII, String internationalStudent, String courseName, String deptName) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.contact = contact;  
 this.dob = dateOfBirth;  
 this.address = address;  
 this.gradeXII = gradeXII;  
 this.internationalStudent = internationalStudent;  
 this.matID = getRandomMatID();  
 this.courseName = courseName;  
 this.deptName = deptName;  
 }  
  
 // Getters  
  
 @Override  
 public String getFirstName() {  
 return this.firstName;  
 }  
  
 @Override  
 public String getLastName() {  
 return this.lastName;  
 }  
  
 @Override  
 public String getContact() {  
 return this.contact;  
 }  
  
 @Override  
 public String getDOB() {  
 return this.dob;  
 }  
  
 @Override  
 public String getAddress() {  
 return this.address;  
 }  
  
 public String getMatID() { return this.matID; }  
  
 public String getCourseName() { return this.courseName; }  
  
 public String getDeptName() { return this.deptName; }  
  
 // Setters  
  
 @Override  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 @Override  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
  
 @Override  
 public void setContact(String contact) {  
 this.contact = contact;  
 }  
  
 @Override  
 public void setDOB(String dob) {  
 this.dob = dob;  
 }  
  
 @Override  
 public void setAddress(String address) {  
 this.address = address;  
 }  
  
 public void setCourseName(String courseName){  
 this.courseName = courseName;  
 }  
  
 public void setDeptName(String deptName){  
 this.deptName = deptName;  
 }  
  
 // Custom Functions  
  
 public String getRandomMatID(){  
 Random rnd = new Random();  
 int n = 100000 + rnd.nextInt(900000);  
 return String.*valueOf*(n);  
 }  
  
 @Override  
 public boolean checkDOB(int threshold){  
  
 // Check if the student is more than 16 years to be able to enroll  
  
 try {  
 DateTimeFormatter formatter = DateTimeFormatter.*ofPattern*("dd-MMM-yyyy");  
 LocalDate birthDate = LocalDate.*parse*(this.dob, formatter);  
 LocalDate currentDate = LocalDate.*now*();  
 Period age = Period.*between*(birthDate, currentDate);  
 return age.getYears() > threshold;  
 } catch (DateTimeParseException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 @Override  
 public boolean checkContact(String contact){  
 // German Phone Number Regex  
 String regex = "^(\\+49|0)[1-9][0-9]{1,14}$";  
  
 // Compile  
 Pattern pattern = Pattern.*compile*(regex);  
  
 // Match  
 return pattern.matcher(contact).matches();  
 }  
  
 public boolean checkIfFeesPaid(String immaID){  
 Conn c = new Conn();  
 System.*out*.println("Checking for student: " + immaID);  
  
 try{  
 String query = """  
 SELECT FEES\_PAID  
 FROM FEES F  
 WHERE F.STUDENT\_ID =  
 """ + immaID + ";";  
  
 ResultSet rs = c.stmt.executeQuery(query);  
 String result = "";  
  
 while (rs.next()) {  
 result = rs.getString("FEES\_PAID");  
 }  
  
 System.*out*.println("Did the student pay the fees: " + result + " ?");  
 return result.equals("YES");  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
  
 return false;  
 }  
  
 public int insertIntoDB(Conn c, int threshold){  
 if(!firstName.isEmpty() && !lastName.isEmpty() && !contact.isEmpty() && !dob.isEmpty() && !address.isEmpty() && !gradeXII.isEmpty() && !internationalStudent.isEmpty() && !matID.isEmpty()){  
 try{  
 String query = """  
 INSERT INTO STUDENT (FIRST\_NAME, LAST\_NAME, CONTACT, DOB, HOME\_ADDRESS, GRADE\_XII, INTERNATIONAL\_STUDENT, IMMA\_ID) VALUES ('"""  
 + firstName + "','"  
 + lastName + "','"  
 + contact + "','"  
 + dob + "','"  
 + address + "','"  
 + gradeXII + "','"  
 + internationalStudent + "','"  
 + matID + "')";  
  
 c.stmt.executeUpdate(query);  
  
 if(!checkDOB(threshold) || !checkContact(contact)){  
 return -1;  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 return -1;  
 }  
  
 return 1; // Insertion successful  
 }  
  
 return 0; // Values are missing  
 }  
  
 public int updateContactDB(Conn c, String stuID, String newContact, String newAddress){  
 try{  
 if(!newContact.isEmpty()){  
 System.*out*.println("Empty?" + newContact);  
 String query = """  
 UPDATE STUDENT  
 SET CONTACT = '  
 """ + newContact + "'"  
 +  
 """  
 WHERE IMMA\_ID =  
 """  
 + stuID + ";";  
 c.stmt.executeUpdate(query);  
  
 if(!checkContact(newContact)){  
 return -1; // rollback in the outside function  
 }  
 else{  
 c.conn.commit();  
 }  
 }  
  
 if(!newAddress.isEmpty()){  
 System.*out*.println("Empty for " + stuID + " ?" + newAddress);  
 String query = """  
 UPDATE STUDENT  
 SET HOME\_ADDRESS = '  
 """ + newAddress + "'"  
 +  
 """  
 WHERE IMMA\_ID =  
 """  
 + stuID + ";";  
 c.stmt.executeUpdate(query);  
  
 c.conn.commit();  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 return -1;  
 }  
  
 return 1;  
 }  
  
 public int updateFeesDB(Conn c, String immaID, String d\_ticket, String variables){  
 try{  
 c.conn.setAutoCommit(false);  
  
 if(!d\_ticket.isEmpty()){  
  
 String query = """  
 UPDATE FEES  
 SET D\_TICKET = '  
 """ + d\_ticket + "', FEES\_PAID = 'YES' WHERE STUDENT\_ID = '" + immaID + "';";  
 c.stmt.executeUpdate(query);  
  
 if(checkIfFeesPaid(immaID)){  
 return -1; // rollback in the outside function  
 }  
 else{  
 c.conn.commit();  
 }  
 }  
  
 if(!variables.isEmpty()){  
  
 String query = """  
 UPDATE FEES  
 SET RESEARCH\_VARIABLE = '  
 """ + variables + "', FEES\_PAID = 'YES' WHERE STUDENT\_ID = '" + immaID + "';";  
 c.stmt.executeUpdate(query);  
  
 if(checkIfFeesPaid(immaID)){  
 return -1; // rollback in the outside function  
 }  
 else{  
 c.conn.commit();  
 }  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 return -1;  
 }  
  
 return 1;  
 }  
}

### 7.1.9 Java Swings Class to add a new instructor record – AddInstructor.java

package instructor;  
  
*/\*\*  
 @author Sanjay Prabhu Kunjibettu  
 @author Tanay Khilare  
 \*/*// General Libraries  
import java.awt.\*;  
import javax.swing.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Random;  
  
// Plugins  
import com.toedter.calendar.JDateChooser;  
  
// Importing Custom Classes  
import connections.Conn;  
import connections.DeptToCourseMapping;  
import oop.Instructor;  
  
public class AddInstructor extends JFrame implements ActionListener{  
 JLabel lblHeading, lblDepartment, lblCourse,  
 lblFirstName, lblLastName, lblContact,  
 lblDOB, lblAddressStreet, lblAddressBuilding, lblAddressPostCode, lblAddressCity, lblBuilding, lblRoomNumber;  
 TextField tfFirstName, tfLastName, tfContact,  
 tfAddressBuilding, tfAddressStreet, tfAddressPostCode, tfAddressCity, tfBuilding, tfRoomNumber;  
 JDateChooser dateChooserDOB;  
 JComboBox cbDepartment;  
 JComboBox<String> cbCourse;  
 JButton btnSubmit, btnCancel;  
  
 DeptToCourseMapping dcm;  
  
 public AddInstructor(){  
  
 /\* 1. Form Heading \*/  
  
 // 1. Heading  
 lblHeading = new JLabel("Form: Add a New Instructor");  
 lblHeading.setBounds(270, 30, 500, 50);  
 lblHeading.setFont(new Font("serif",Font.*BOLD*,30));  
 this.add(lblHeading);  
  
 /\* 2. Personal Details \*/  
  
 JLabel lblPersonalDetails = new JLabel("Personal Details ");  
 lblPersonalDetails.setBounds(350, 100, 250, 30);  
 lblPersonalDetails.setFont(new Font("serif", Font.*ITALIC*, 20));  
 this.add(lblPersonalDetails);  
  
 // 2.1 First Name  
 lblFirstName = new JLabel("First Name: ");  
 lblFirstName.setBounds(50, 150, 150, 30);  
 lblFirstName.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblFirstName);  
  
 tfFirstName = new TextField();  
 tfFirstName.setBounds(200, 150, 150, 30);  
 this.add(tfFirstName);  
  
 // 2.2 Last Name  
 lblLastName = new JLabel("Last Name: ");  
 lblLastName.setBounds(400, 150, 150, 30);  
 lblLastName.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblLastName);  
  
 tfLastName = new TextField();  
 tfLastName.setBounds(600, 150, 150, 30);  
 this.add(tfLastName);  
  
 // 2.3 Contact  
 lblContact = new JLabel("Contact: ");  
 lblContact.setBounds(50, 200, 150, 30);  
 lblContact.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblContact);  
  
 tfContact = new TextField();  
 tfContact.setBounds(200, 200, 150, 30);  
 this.add(tfContact);  
  
 // 2.4 Date of Birth  
 lblDOB = new JLabel("Date of Birth");  
 lblDOB.setBounds(400, 200, 150, 30);  
 lblDOB.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblDOB);  
  
 dateChooserDOB = new JDateChooser();  
 dateChooserDOB.setBounds(600, 200, 150, 30);  
 this.add(dateChooserDOB);  
  
 // 2.5 Address  
  
 // // 2.5.1 Street  
 lblAddressStreet = new JLabel("Street: ");  
 lblAddressStreet.setBounds(50, 250, 150, 30);  
 lblAddressStreet.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblAddressStreet);  
  
 tfAddressStreet = new TextField();  
 tfAddressStreet.setBounds(200, 250, 150, 30);  
 this.add(tfAddressStreet);  
  
 // // 2.5.2 Building Number  
 lblAddressBuilding = new JLabel("Building: ");  
 lblAddressBuilding.setBounds(400, 250, 150, 30);  
 lblAddressBuilding.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblAddressBuilding);  
  
 tfAddressBuilding = new TextField();  
 tfAddressBuilding.setBounds(600, 250, 150, 30);  
 this.add(tfAddressBuilding);  
  
 // // 2.5.3 Post Code  
 lblAddressPostCode = new JLabel("Post Code: ");  
 lblAddressPostCode.setBounds(50, 300, 150, 30);  
 lblAddressPostCode.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblAddressPostCode);  
  
 tfAddressPostCode = new TextField();  
 tfAddressPostCode.setBounds(200, 300, 150, 30);  
 this.add(tfAddressPostCode);  
  
 // // 2.5.4 City  
 lblAddressCity = new JLabel("City: ");  
 lblAddressCity.setBounds(400, 300, 150, 30);  
 lblAddressCity.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblAddressCity);  
  
 tfAddressCity = new TextField();  
 tfAddressCity.setBounds(600, 300, 150, 30);  
 this.add(tfAddressCity);  
  
 /\* 3. Hochschule Details \*/  
  
 JLabel lblHFWDetails = new JLabel("Hochschule Details");  
 lblHFWDetails.setBounds(350, 350, 250, 30);  
 lblHFWDetails.setFont(new Font("serif", Font.*ITALIC*, 20));  
 this.add(lblHFWDetails);  
  
 // 3.1 Hochschule Cabin  
 lblBuilding = new JLabel("HFW Building: ");  
 lblBuilding.setBounds(50, 400, 150, 30);  
 lblBuilding.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblBuilding);  
  
 tfBuilding = new TextField();  
 tfBuilding.setBounds(200, 400, 150, 30);  
 this.add(tfBuilding);  
  
 // 3.2 Room Number  
 lblRoomNumber = new JLabel("Room: #");  
 lblRoomNumber.setBounds(400, 400, 150, 30);  
 lblRoomNumber.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblRoomNumber);  
  
 tfRoomNumber = new TextField();  
 tfRoomNumber.setBounds(600, 400, 150, 30);  
 this.add(tfRoomNumber);  
  
 // 3.3 Department  
 lblDepartment = new JLabel("Department");  
 lblDepartment.setBounds(50,450,150,30);  
 lblDepartment.setFont(new Font("serif",Font.*BOLD*,20));  
 this.add(lblDepartment);  
  
 dcm = new DeptToCourseMapping(); // Dynamic Department to Course Mapping  
 String[] dept = dcm.getDeptList();  
 cbDepartment = new JComboBox(dept);  
 cbDepartment.setBounds(200,450,150,30);  
 cbDepartment.setBackground(Color.*WHITE*);  
 this.add(cbDepartment);  
  
 // 3.3 Course  
 lblCourse = new JLabel("Course: ");  
 lblCourse.setBounds(400,450,150,30);  
 lblCourse.setFont(new Font("serif",Font.*BOLD*,20));  
 this.add(lblCourse);  
  
 cbCourse = new JComboBox<>();  
 cbCourse.setBounds(500,450,250,30);  
 cbCourse.setBackground(Color.*WHITE*);  
 this.add(cbCourse);  
  
 // Dynamic Mapping of Departments to Courses  
 HashMap<String, List<String>> deptToCourse = dcm.getDeptToCourseMapping();  
  
 cbDepartment.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 String deptStr = (String) cbDepartment.getSelectedItem();  
 cbCourse.removeAllItems();  
 for (String course : deptToCourse.get(deptStr)) {  
 cbCourse.addItem(course);  
 }  
 }  
 });  
  
 cbDepartment.setSelectedIndex(0); // Default Selection to 1st Option  
  
 /\* 4. Buttons \*/  
  
 // 4.1 Submit  
 btnSubmit = new JButton("SUBMIT");  
 btnSubmit.setBounds(250,550,120,30);  
 btnSubmit.setBackground(Color.*BLACK*);  
 btnSubmit.setForeground(Color.*WHITE*);  
 btnSubmit.addActionListener(this);  
 this.add(btnSubmit);  
  
 // 4.2 Cancel  
 btnCancel = new JButton("CANCEL");  
 btnCancel.setBounds(450,550,120,30);  
 btnCancel.setBackground(Color.*BLACK*);  
 btnCancel.setForeground(Color.*WHITE*);  
 btnCancel.addActionListener(this);  
 this.add(btnCancel);  
  
 /\* 5. JFrame Configurations \*/  
 // this.getContentPane().setBackground(new Color(166,164,252));  
 this.setSize(900, 700);  
 this.setLocation(350, 50);  
 this.setLayout(null);  
 this.setVisible(true);  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent ae){  
 if(ae.getSource() == btnSubmit){  
 String firstName = tfFirstName.getText();  
 String lastName = tfLastName.getText();  
 String contact = tfContact.getText();  
 String dob = ((JTextField) dateChooserDOB.getDateEditor().getUiComponent()).getText();  
  
 // Address  
 String addressStreet = tfAddressStreet.getText();  
 String addressBuilding = tfAddressBuilding.getText();  
 String addressPostCode = tfAddressPostCode.getText();  
 String addressCity = tfAddressCity.getText();  
 String address = addressBuilding + ", " + addressStreet + ", " + addressPostCode + ", " + addressCity;  
  
 String building = tfBuilding.getText();  
 String roomNumber = tfRoomNumber.getText();  
 String hftCabin = building + "/" + roomNumber;  
  
 String dept = (String) cbDepartment.getSelectedItem();  
 String course = (String) cbCourse.getSelectedItem();  
  
 // 2. Create connection from the instructor template  
  
 Instructor newInstructor = new Instructor(firstName, lastName, contact, dob, address, hftCabin, course, dept);  
 Conn c = new Conn();  
  
 // 3. Transactions - COMMIT and ROLLBACK  
  
 try{  
 c.conn.setAutoCommit(false);  
  
 // Check if the new Instructor is more than 25 years old  
 int result = newInstructor.insertIntoDB(c, 25);  
 if(result==1) {  
 // COMMIT  
 c.conn.commit();  
  
 String query = """  
 update COURSE  
 set instructor\_id=  
 (  
 select emp\_id from INSTRUCTOR where first\_name='"""  
 + firstName +  
 """  
 ' and last\_name='"""  
 + lastName + "')" + "where upper(course\_name) = upper('" + course + "');";  
  
 c.stmt.executeUpdate(query);  
 c.conn.commit();  
  
 JOptionPane.*showMessageDialog*(null, "New Instructor Successfully added!");  
 this.setVisible(false);  
 }  
 else if (result == 0) {  
 JOptionPane.*showMessageDialog*(null, "ERROR: All values must be entered!");  
 }  
 else{  
 // ROLLBACK  
 // Instructor is less than or equal to 25 years and is NOT eligible  
 c.conn.rollback();  
 JOptionPane.*showMessageDialog*(null, "ERROR: Age>=25 and/or Contact must follow german phone standards!");  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
 else{  
 // Cancel Button  
 // Do nothing  
 this.setVisible(false);  
 }  
 }  
  
 public static void main(String[] args) {  
 new AddInstructor();  
 }  
}

### 7.1.10 Java Swings Class to update/delete an existing instructor record – UpdateInstructor.java

package instructor;  
  
*/\*\*  
 @author Sanjay Prabhu Kunjibettu  
 @author Tanay Khilare  
 \*/*// General Libraries  
import java.awt.\*;  
import javax.swing.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.ItemEvent;  
import java.awt.event.ItemListener;  
import java.sql.\*;  
  
// Plugins  
  
// Importing Custom Classes  
import connections.Conn;  
import oop.Instructor;  
  
public class UpdateInstructor extends JFrame implements ActionListener{  
 Choice empIDChoice;  
 JLabel lblMainHeading, lblHeading,  
 lblDepartment, lblCourse, lblDepartment2, lblCourse2,  
 lblFirstName, lblLastName, lblFirstName2, lblLastName2,  
 lblHFTCabin, lblHFTCabin2,  
 lblEmail, lblEmail2,  
 lblDOB, lblDOB2, lblContact, lblAddress;  
 TextField tfContact, tfAddress;  
 JButton btnUpdate, btnRemove, btnCancel;  
  
 public UpdateInstructor(){  
  
 /\* 1. Form Heading \*/  
  
 // 1.1 Heading  
 lblMainHeading = new JLabel("Form: Update/Remove an existing Instructor Record");  
 lblMainHeading.setBounds(100, 10, 700, 50);  
 lblMainHeading.setFont(new Font("serif",Font.BOLD,30));  
 this.add(lblMainHeading);  
  
 // 1.2 Choice  
 lblHeading = new JLabel("Choose an Employee ID: ");  
 lblHeading.setBounds(50,100,150,20);  
 this.add(lblHeading);  
  
 empIDChoice = new Choice();  
 empIDChoice.setBounds(220,100,150,20);  
 this.add(empIDChoice);  
 empIDChoice.add("Select...");  
  
 // 1.3 Connection to get a list of choice of emp ids  
 try{  
 Conn c = new Conn();  
 String query = "select distinct emp\_id from INSTRUCTOR;";  
 ResultSet rs = c.stmt.executeQuery(query);  
 while(rs.next()){  
 empIDChoice.add(rs.getString("emp\_id"));  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
  
 /\* 2. Form Details \*/  
  
 // 2.1 First Name  
 lblFirstName = new JLabel("First Name: ");  
 lblFirstName.setBounds(50, 150, 150, 30);  
 lblFirstName.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblFirstName);  
  
 lblFirstName2 = new JLabel();  
 lblFirstName2.setBounds(200, 150, 150, 30);  
 lblFirstName2.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblFirstName2);  
  
 // 2.2 Last Name  
 lblLastName = new JLabel("Last Name: ");  
 lblLastName.setBounds(400, 150, 150, 30);  
 lblLastName.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblLastName);  
  
 lblLastName2 = new JLabel();  
 lblLastName2.setBounds(600, 150, 150, 30);  
 lblLastName2.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblLastName2);  
  
 // 2.3 Email  
 lblEmail = new JLabel("Email: ");  
 lblEmail.setBounds(50, 200, 150, 30);  
 lblEmail.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblEmail);  
  
 lblEmail2 = new JLabel();  
 lblEmail2.setBounds(200, 200, 500, 30);  
 lblEmail2.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblEmail2);  
  
 // 2.6 Contact  
 lblContact = new JLabel("Contact: ");  
 lblContact.setBounds(50, 250, 150, 30);  
 lblContact.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblContact);  
  
 tfContact = new TextField();  
 tfContact.setBounds(200, 250, 150, 30);  
 this.add(tfContact);  
  
 // 2.7 Date of Birth  
 lblDOB = new JLabel("Date of Birth: ");  
 lblDOB.setBounds(400, 250, 150, 30);  
 lblDOB.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblDOB);  
  
 lblDOB2 = new JLabel();  
 lblDOB2.setBounds(600, 250, 150, 30);  
 lblDOB2.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblDOB2);  
  
 // 2.8 HFT Cabin  
 lblHFTCabin = new JLabel("HFT Cabin: ");  
 lblHFTCabin.setBounds(400, 100, 150, 30);  
 lblHFTCabin.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblHFTCabin);  
  
 lblHFTCabin2 = new JLabel();  
 lblHFTCabin2.setBounds(600, 100, 150, 30);  
 lblHFTCabin2.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblHFTCabin2);  
  
 // 2.5 Address  
 lblAddress = new JLabel("Home Address: ");  
 lblAddress.setBounds(50, 300, 150, 30);  
 lblAddress.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblAddress);  
  
 tfAddress = new TextField();  
 tfAddress.setBounds(200, 300, 550, 30);  
 this.add(tfAddress);  
  
 // 2.8 Department  
 lblDepartment = new JLabel("Department: ");  
 lblDepartment.setBounds(50,350,200,30);  
 lblDepartment.setFont(new Font("serif",Font.BOLD,20));  
 this.add(lblDepartment);  
  
 lblDepartment2 = new JLabel();  
 lblDepartment2.setBounds(200,350,200,30);  
 lblDepartment2.setFont(new Font("serif",Font.BOLD,20));  
 this.add(lblDepartment2);  
  
 // 2.9 Course  
 lblCourse = new JLabel("Course: ");  
 lblCourse.setBounds(400,350,150,30);  
 lblCourse.setFont(new Font("serif",Font.BOLD,20));  
 this.add(lblCourse);  
  
 lblCourse2 = new JLabel();  
 lblCourse2.setBounds(500,350,250,30);  
 lblCourse2.setFont(new Font("serif",Font.BOLD,20));  
 this.add(lblCourse2);  
  
 empIDChoice.addItemListener(new ItemListener() {  
 @Override  
 public void itemStateChanged(ItemEvent ie) {  
 try {  
 Conn c = new Conn();  
 String choiceItem = empIDChoice.getSelectedItem();  
 String query = """  
 SELECT DISTINCT  
 I.EMP\_ID, I.FIRST\_NAME, I.LAST\_NAME, I.CONTACT, I.EMAIL, I.DOB, I.HOME\_ADDRESS, I.HFT\_CABIN,  
 C.COURSE\_NAME, D.DEPT\_NAME  
 FROM  
 INSTRUCTOR I  
 LEFT JOIN COURSE C  
 ON I.EMP\_ID = C.INSTRUCTOR\_ID  
 LEFT JOIN DEPARTMENT D  
 ON C.DEPT\_ID = D.DEPT\_ID  
 WHERE I.EMP\_ID =  
 """ + choiceItem + ";";  
  
 ResultSet resultSet = c.stmt.executeQuery(query);  
  
 while (resultSet.next()) {  
 lblFirstName2.setText(resultSet.getString("first\_name"));  
 lblLastName2.setText(resultSet.getString("last\_name"));  
 lblEmail2.setText(resultSet.getString("email"));  
 lblDOB2.setText(resultSet.getString("dob"));  
 lblHFTCabin2.setText(resultSet.getString("hft\_cabin"));  
 lblCourse2.setText(resultSet.getString("course\_name"));  
 lblDepartment2.setText(resultSet.getString("dept\_name"));  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }}  
 });  
  
 /\* 3. Buttons \*/  
  
 // 3.1 Update  
 btnUpdate = new JButton("UPDATE");  
 btnUpdate.setBounds(250,550,120,30);  
 btnUpdate.setBackground(Color.BLACK);  
 btnUpdate.setForeground(Color.WHITE);  
 btnUpdate.addActionListener(this);  
 this.add(btnUpdate);  
  
 // 3.2 Remove  
 btnRemove = new JButton("REMOVE");  
 btnRemove.setBounds(450,550,120,30);  
 btnRemove.setBackground(Color.BLACK);  
 btnRemove.setForeground(Color.WHITE);  
 btnRemove.addActionListener(this);  
 this.add(btnRemove);  
  
 // 3.2 Cancel  
 btnCancel = new JButton("CANCEL");  
 btnCancel.setBounds(650,550,120,30);  
 btnCancel.setBackground(Color.BLACK);  
 btnCancel.setForeground(Color.WHITE);  
 btnCancel.addActionListener(this);  
 this.add(btnCancel);  
  
 /\* 4. JFrame Configurations \*/  
 // this.getContentPane().setBackground(new Color(166,164,252));  
 this.setSize(900, 700);  
 this.setLocation(350, 50);  
 this.setLayout(null);  
 this.setVisible(true);  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent ae){  
 if(ae.getSource() == btnUpdate){  
 String choiceItem = empIDChoice.getSelectedItem();  
 if(!choiceItem.equals("Select...")){  
 int dialogButton = JOptionPane.YES\_NO\_OPTION;  
 int dialogResult = JOptionPane.showConfirmDialog(null, "WARNING: You are about to update " + choiceItem + "!","WARNING",dialogButton);  
  
 Conn c = new Conn();  
  
 try{  
 if(dialogResult == 0) {  
 // YES  
 String contact = tfContact.getText();  
 String address = tfAddress.getText();  
  
 c.conn.setAutoCommit(false);  
  
 Instructor existingInstructor = new Instructor();  
  
 if(existingInstructor.updateContactDB(c, choiceItem, contact, address)==1){  
 // COMMIT  
 c.conn.commit();  
  
 JOptionPane.showMessageDialog(null, "UPDATE: Instructor Record updated successfully!");  
 setVisible(false);  
 }  
 else{  
 // ROLLBACK  
 // Contact mistake  
 c.conn.rollback();  
 JOptionPane.showMessageDialog(null, "ERROR: Contact must follow german phone standards!");  
 };  
 }  
 else{  
 // NO ACTION ITEM  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
 else{  
 JOptionPane.showMessageDialog(null, "ERROR: Select an instructor employee number to update!");  
 }  
 }  
 else if(ae.getSource() == btnRemove){  
 String choiceItem = empIDChoice.getSelectedItem();  
 if(!choiceItem.equals("Select...")){  
 int dialogButton = JOptionPane.YES\_NO\_OPTION;  
 int dialogResult = JOptionPane.showConfirmDialog(null, "WARNING: You are about to remove " + choiceItem + "!","WARNING",dialogButton);  
  
 if(dialogResult == 0) {  
 // YES  
  
 Conn c = new Conn();  
  
 try {  
 c.conn.setAutoCommit(false);  
  
 String query = """  
 DELETE FROM INSTRUCTOR  
 WHERE EMP\_ID =  
 """  
 + choiceItem + ";";  
  
 c.stmt.executeUpdate(query);  
 c.conn.commit();  
  
 JOptionPane.showMessageDialog(null, "UPDATE: Instructor Record deleted successfully!");  
 setVisible(false);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 else{  
 // NO ACTION ITEM  
 }  
 }  
 else{  
 JOptionPane.showMessageDialog(null, "ERROR: Select an instructor employee number to delete!");  
 }  
 }  
 else{  
 this.setVisible(false);  
 }  
 }  
  
 public static void main(String[] args) {  
 new UpdateInstructor();  
 }  
}

### 7.1.11 Java Swings Class to view an existing instructor record – ViewInstructor.java

package instructor;  
  
*/\*\*  
 @author Sanjay Prabhu Kunjibettu  
 @author Tanay Khilare  
 \*/*// General Libraries  
import java.awt.\*;  
import javax.swing.\*;  
import javax.swing.table.TableColumn;  
import javax.swing.table.TableColumnModel;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.sql.ResultSet;  
import java.sql.ResultSetMetaData;  
  
// Import Plugins  
import net.proteanit.sql.DbUtils; // rs2xml.jar  
  
// Importing Custom Classes  
import connections.Conn;  
  
public class ViewInstructor extends JFrame implements ActionListener {  
 Choice empIDChoice;  
 JLabel lblHeading;  
 JTable tableAll;  
 JButton btnSearch, btnPrint, btnCancel;  
 JScrollPane scrollPane;  
  
 public ViewInstructor(){  
  
 // 0. Set Border Layout  
 // Border Layout works best for tables, scrolling and printing  
 this.setLayout(new BorderLayout());  
  
 /\* 1. Select the Instructor ID to view their details \*/  
  
 // 1.1 Label  
 lblHeading = new JLabel("Choose an Employee ID: ");  
  
 // 1.2 Choice  
 empIDChoice = new Choice();  
 empIDChoice.add("All");  
  
 // 1.3 Connection to get a list of choice of emp ids  
 try{  
 Conn c = new Conn();  
 String query = "select distinct emp\_id from INSTRUCTOR;";  
 ResultSet rs = c.stmt.executeQuery(query);  
 while(rs.next()){  
 empIDChoice.add(rs.getString("emp\_id"));  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
  
 // 1.4 Connection to get instructor info with course and dept info  
 tableAll = new JTable();  
 try{  
 Conn c = new Conn();  
 String query = """  
 SELECT DISTINCT  
 I.EMP\_ID, I.FIRST\_NAME, I.LAST\_NAME, I.CONTACT, I.EMAIL, I.DOB, I.HOME\_ADDRESS, I.HFT\_CABIN,  
 C.COURSE\_NAME, D.DEPT\_NAME  
 FROM  
 INSTRUCTOR I  
 LEFT JOIN COURSE C  
 ON I.EMP\_ID = C.INSTRUCTOR\_ID  
 LEFT JOIN DEPARTMENT D  
 ON C.DEPT\_ID = D.DEPT\_ID;  
 """;  
 ResultSet rs = c.stmt.executeQuery(query);  
 ResultSetMetaData metaData = rs.getMetaData();  
 int totalColumns = metaData.getColumnCount();  
  
 tableAll.setModel(DbUtils.*resultSetToTableModel*(rs));  
 tableAll.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*);  
  
 // Set column size  
 TableColumnModel colModel = tableAll.getColumnModel();  
 for(int i=0; i<totalColumns; i++){  
 TableColumn tc = colModel.getColumn(i);  
 tc.setPreferredWidth(130);  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
  
 // 1.5 Scroll Pane  
 scrollPane = new JScrollPane(tableAll, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_ALWAYS*);  
  
 // 1.6 Border Layout for Scroll Pane  
 JPanel centerPanel = new JPanel();  
 centerPanel.add(scrollPane);  
 centerPanel.setPreferredSize(new Dimension(900,(int)centerPanel.getPreferredSize().getHeight()));  
 centerPanel.setLayout(new GridLayout(1, 3));  
 this.add(centerPanel, BorderLayout.*CENTER*);  
  
 /\* 2. Buttons \*/  
  
 // 2.1 Search  
 btnSearch = new JButton("SEARCH");  
 btnSearch.addActionListener(this);  
  
 // 2.2 Print  
 btnPrint = new JButton("PRINT");  
 btnPrint.addActionListener(this);  
  
 // 2.3 Cancel the operation  
 btnCancel = new JButton("CANCEL");  
 btnCancel.addActionListener(this);  
  
 // 2.4 North Border Layout  
 JPanel northPanel = new JPanel();  
 northPanel.add(lblHeading);  
 northPanel.add(empIDChoice);  
 northPanel.add(btnPrint);  
 northPanel.add(btnSearch);  
 northPanel.add(btnCancel);  
 northPanel.setPreferredSize(new Dimension(900,(int)northPanel.getPreferredSize().getHeight()));  
 this.add(northPanel, BorderLayout.*NORTH*);  
  
 /\* 3. JFrame Configurations \*/  
  
 this.setTitle("Instructor Details");  
 this.pack();  
 this.setSize(900, 600);  
 this.setLocation(350, 50);  
 this.setVisible(true);  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent ae){  
 if (ae.getSource() == btnSearch){  
 try {  
 Conn c = new Conn();  
 String choiceItem = empIDChoice.getSelectedItem();  
 if(!choiceItem.equals("All")){  
 String query = """  
 SELECT DISTINCT  
 I.EMP\_ID, I.FIRST\_NAME, I.LAST\_NAME, I.CONTACT, I.EMAIL, I.DOB, I.HOME\_ADDRESS, I.HFT\_CABIN,  
 C.COURSE\_NAME, D.DEPT\_NAME  
 FROM  
 INSTRUCTOR I  
 LEFT JOIN COURSE C  
 ON I.EMP\_ID = C.INSTRUCTOR\_ID  
 LEFT JOIN DEPARTMENT D  
 ON C.DEPT\_ID = D.DEPT\_ID  
 WHERE I.EMP\_ID =  
 """ + choiceItem + ";";  
  
 ResultSet resultSet = c.stmt.executeQuery(query);  
  
 ResultSetMetaData metaData = resultSet.getMetaData();  
 int totalColumns = metaData.getColumnCount();  
  
 tableAll.setModel(DbUtils.*resultSetToTableModel*(resultSet));  
 tableAll.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*);  
  
 // Set column size  
 TableColumnModel colModel = tableAll.getColumnModel();  
 for(int i=0; i<totalColumns; i++){  
 TableColumn tc = colModel.getColumn(i);  
 tc.setPreferredWidth(130);  
 }  
 }  
 else{  
 String query = """  
 SELECT DISTINCT  
 I.EMP\_ID, I.FIRST\_NAME, I.LAST\_NAME, I.CONTACT, I.EMAIL, I.DOB, I.HOME\_ADDRESS, I.HFT\_CABIN,  
 C.COURSE\_NAME, D.DEPT\_NAME  
 FROM  
 INSTRUCTOR I  
 LEFT JOIN COURSE C  
 ON I.EMP\_ID = C.INSTRUCTOR\_ID  
 LEFT JOIN DEPARTMENT D  
 ON C.DEPT\_ID = D.DEPT\_ID  
 """;  
  
 ResultSet resultSet = c.stmt.executeQuery(query);  
 ResultSetMetaData metaData = resultSet.getMetaData();  
 int totalColumns = metaData.getColumnCount();  
  
 tableAll.setModel(DbUtils.*resultSetToTableModel*(resultSet));  
 tableAll.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*);  
  
 // Set column size  
 TableColumnModel colModel = tableAll.getColumnModel();  
 for(int i=0; i<totalColumns; i++){  
 TableColumn tc = colModel.getColumn(i);  
 tc.setPreferredWidth(130);  
 }  
 }  
  
 }catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
 else if (ae.getSource() == btnPrint){  
 try {  
 tableAll.print();  
 }catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
 else {  
 // Cancel Transaction  
 this.setVisible(false);  
 }  
 }  
  
 public static void main(String[] args) {  
 new ViewInstructor();  
 }  
}

### 7.1.12 Java Swings Class to insert a new student record – AddStudent.java

package student;  
  
*/\*\*  
 @author Sanjay Prabhu Kunjibettu  
 @author Tanay Khilare  
 \*/*// General Libraries  
import java.awt.\*;  
import javax.swing.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.util.HashMap;  
import java.util.List;  
import java.sql.\*;  
import java.time.LocalDate;  
  
// Plugins  
import com.toedter.calendar.JDateChooser;  
  
// Importing Custom Classes  
import connections.Conn;  
import connections.DeptToCourseMapping;  
import oop.Student;  
  
public class AddStudent extends JFrame implements ActionListener{  
 JLabel lblHeading, lblDepartment, lblCourse,  
 lblFirstName, lblLastName, lblContact,  
 lblDOB, lblAddressStreet, lblAddressBuilding, lblAddressPostCode, lblAddressCity, lblXII, lblInternational;  
 TextField tfFirstName, tfLastName, tfContact,  
 tfAddressBuilding, tfAddressStreet, tfAddressPostCode, tfAddressCity, tfXII;  
 JDateChooser dateChooserDOB;  
 JComboBox cbDepartment, cbInternational;  
 JComboBox<String> cbCourse;  
 JButton btnSubmit, btnCancel;  
  
 DeptToCourseMapping dcm;  
  
 public AddStudent(){  
  
 /\* 1. Form Heading \*/  
  
 // 1. Heading  
 lblHeading = new JLabel("Form: Add a New Student");  
 lblHeading.setBounds(270, 30, 500, 50);  
 lblHeading.setFont(new Font("serif",Font.*BOLD*,30));  
 this.add(lblHeading);  
  
 /\* 2. Form Details \*/  
  
 // 2.1 First Name  
 lblFirstName = new JLabel("First Name: ");  
 lblFirstName.setBounds(50, 150, 150, 30);  
 lblFirstName.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblFirstName);  
  
 tfFirstName = new TextField();  
 tfFirstName.setBounds(200, 150, 150, 30);  
 this.add(tfFirstName);  
  
 // 2.2 Last Name  
 lblLastName = new JLabel("Last Name: ");  
 lblLastName.setBounds(400, 150, 150, 30);  
 lblLastName.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblLastName);  
  
 tfLastName = new TextField();  
 tfLastName.setBounds(600, 150, 150, 30);  
 this.add(tfLastName);  
  
 // 2.3 Contact  
 lblContact = new JLabel("Contact: ");  
 lblContact.setBounds(50, 200, 150, 30);  
 lblContact.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblContact);  
  
 tfContact = new TextField();  
 tfContact.setBounds(200, 200, 150, 30);  
 this.add(tfContact);  
  
 // 2.4 Date of Birth  
 lblDOB = new JLabel("Date of Birth");  
 lblDOB.setBounds(400, 200, 150, 30);  
 lblDOB.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblDOB);  
  
 dateChooserDOB = new JDateChooser();  
 dateChooserDOB.setBounds(600, 200, 150, 30);  
 this.add(dateChooserDOB);  
  
 // 2.5 Address  
  
 // // 2.5.1 Street  
 lblAddressStreet = new JLabel("Street: ");  
 lblAddressStreet.setBounds(50, 250, 150, 30);  
 lblAddressStreet.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblAddressStreet);  
  
 tfAddressStreet = new TextField();  
 tfAddressStreet.setBounds(200, 250, 150, 30);  
 this.add(tfAddressStreet);  
  
 // // 2.5.2 Building Number  
 lblAddressBuilding = new JLabel("Building: ");  
 lblAddressBuilding.setBounds(400, 250, 150, 30);  
 lblAddressBuilding.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblAddressBuilding);  
  
 tfAddressBuilding = new TextField();  
 tfAddressBuilding.setBounds(600, 250, 150, 30);  
 this.add(tfAddressBuilding);  
  
 // // 2.5.3 Post Code  
 lblAddressPostCode = new JLabel("Post Code: ");  
 lblAddressPostCode.setBounds(50, 300, 150, 30);  
 lblAddressPostCode.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblAddressPostCode);  
  
 tfAddressPostCode = new TextField();  
 tfAddressPostCode.setBounds(200, 300, 150, 30);  
 this.add(tfAddressPostCode);  
  
 // // 2.5.4 City  
 lblAddressCity = new JLabel("City: ");  
 lblAddressCity.setBounds(400, 300, 150, 30);  
 lblAddressCity.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblAddressCity);  
  
 tfAddressCity = new TextField();  
 tfAddressCity.setBounds(600, 300, 150, 30);  
 this.add(tfAddressCity);  
  
 // 2.6 Grade XII Marks  
 lblXII = new JLabel("Grade XII: ");  
 lblXII.setBounds(50, 350, 150, 30);  
 lblXII.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblXII);  
  
 tfXII = new TextField();  
 tfXII.setBounds(200, 350, 150, 30);  
 this.add(tfXII);  
  
 /\* 3. Hochschule Details \*/  
  
 // 3.1 International Student Status  
 lblInternational = new JLabel("International ?: ");  
 lblInternational.setBounds(400, 350, 150, 30);  
 lblInternational.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblInternational);  
  
 String[] internationalStatus = {"Yes", "No"};  
 cbInternational = new JComboBox(internationalStatus);  
 cbInternational.setBounds(600, 350, 150, 30);  
 cbInternational.setBackground(Color.*WHITE*);  
 this.add(cbInternational);  
  
 // 3.2 Department  
 lblDepartment = new JLabel("Department");  
 lblDepartment.setBounds(50,400,150,30);  
 lblDepartment.setFont(new Font("serif",Font.*BOLD*,20));  
 this.add(lblDepartment);  
  
 dcm = new DeptToCourseMapping(); // Dynamic Department to Course Mapping  
 String[] dept = dcm.getDeptList();  
 cbDepartment = new JComboBox(dept);  
 cbDepartment.setBounds(200,400,150,30);  
 cbDepartment.setBackground(Color.*WHITE*);  
 this.add(cbDepartment);  
  
 // 3.3 Course  
 lblCourse = new JLabel("Course: ");  
 lblCourse.setBounds(400,400,150,30);  
 lblCourse.setFont(new Font("serif",Font.*BOLD*,20));  
 this.add(lblCourse);  
  
 cbCourse = new JComboBox<>();  
 cbCourse.setBounds(500,400,250,30);  
 cbCourse.setBackground(Color.*WHITE*);  
 this.add(cbCourse);  
  
 // Dynamic Mapping of Departments to Courses  
 HashMap<String, List<String>> deptToCourse = dcm.getDeptToCourseMapping();  
  
 cbDepartment.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 String deptStr = (String) cbDepartment.getSelectedItem();  
 cbCourse.removeAllItems();  
 for (String course : deptToCourse.get(deptStr)) {  
 cbCourse.addItem(course);  
 }  
 }  
 });  
  
 cbDepartment.setSelectedIndex(0); // Default Selection to 1st Option  
  
 /\* 4. Buttons \*/  
  
 // 4.1 Submit  
 btnSubmit = new JButton("Submit");  
 btnSubmit.setBounds(250,550,120,30);  
 btnSubmit.setBackground(Color.*BLACK*);  
 btnSubmit.setForeground(Color.*WHITE*);  
 btnSubmit.addActionListener(this);  
 this.add(btnSubmit);  
  
 // 4.2 Cancel  
 btnCancel = new JButton("Cancel");  
 btnCancel.setBounds(450,550,120,30);  
 btnCancel.setBackground(Color.*BLACK*);  
 btnCancel.setForeground(Color.*WHITE*);  
 btnCancel.addActionListener(this);  
 this.add(btnCancel);  
  
 /\* 5. JFrame Configurations \*/  
 this.setSize(900, 700);  
 this.setLocation(350, 50);  
 this.setLayout(null);  
 this.setVisible(true);  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent ae){  
 if(ae.getSource() == btnSubmit){  
  
 // 1. Get the form information  
 String firstName = tfFirstName.getText();  
 String lastName = tfLastName.getText();  
 String contact = tfContact.getText();  
 String dob = ((JTextField) dateChooserDOB.getDateEditor().getUiComponent()).getText();  
  
 // // Address  
 String addressStreet = tfAddressStreet.getText();  
 String addressBuilding = tfAddressBuilding.getText();  
 String addressPostCode = tfAddressPostCode.getText();  
 String addressCity = tfAddressCity.getText();  
 String address = addressBuilding + ", " + addressStreet + ", " + addressPostCode + ", " + addressCity;  
  
 String gradeXII = tfXII.getText();  
 String international = (String) cbInternational.getSelectedItem();  
  
 String dept = (String) cbDepartment.getSelectedItem();  
 String course = (String) cbCourse.getSelectedItem();  
  
 // 2. Create connection from the student template  
 Student newStudent = new Student(firstName, lastName, contact, dob, address, gradeXII, international, course, dept);  
 Conn c = new Conn();  
  
 // 3. Transactions - COMMIT and ROLLBACK  
  
 try{  
 c.conn.setAutoCommit(false);  
  
 // Check if the new student is more than 16 years old  
 int result = newStudent.insertIntoDB(c, 16);  
 if(result == 1){  
 // COMMIT  
 c.conn.commit();  
  
 String deptID="", courseID="";  
 String matID = newStudent.getMatID();  
  
 // Get Course ID  
 String query = "select dept\_id, course\_id from COURSE where upper(course\_name) = upper('" + course + "');";  
 ResultSet rs = c.stmt.executeQuery(query);  
 while(rs.next()){  
 deptID = rs.getString("dept\_id");  
 courseID = rs.getString("course\_id");  
 }  
  
 System.*out*.println(dept + " " + course + " " + matID);  
  
 // Todays date  
 String today = String.*valueOf*(LocalDate.*now*());  
  
 query = "INSERT INTO ENROLLMENT (IMMA\_ID, DEPT\_ID, COURSE\_ID, IMMA\_DATE, IMMA\_STATUS) VALUES ('"  
 + matID + "','"  
 + deptID + "','"  
 + courseID + "','"  
 + today + "','ADMITTED');";  
 c.stmt.executeUpdate(query);  
 c.conn.commit();  
  
 JOptionPane.*showMessageDialog*(null, "New Student Successfully added!");  
 this.setVisible(false);  
 }  
 else if (result == 0) {  
 JOptionPane.*showMessageDialog*(null, "ERROR: All values must be entered!");  
 }  
 else{  
 // ROLLBACK  
 // Student is less than or equal to 16 years and is NOT eligible for enrollment  
 c.conn.rollback();  
 JOptionPane.*showMessageDialog*(null, "ERROR: Age>=16 and/or Contact must follow german phone standards!");  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
 else{  
 // Cancel Button  
 // Do nothing  
 this.setVisible(false);  
 }  
 }  
  
 public static void main(String[] args) {  
 new AddStudent();  
 }  
}

### 7.1.13 Java Swings Class to update/deregister an existing student record – UpdateStudent.java

package student;  
  
*/\*\*  
 @author Sanjay Prabhu Kunjibettu  
 @author Tanay Khilare  
 \*/*// General Libraries  
import java.awt.\*;  
import javax.swing.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.ItemEvent;  
import java.awt.event.ItemListener;  
import java.util.Random;  
import java.sql.\*;  
import java.time.LocalDate;  
  
// Plugins  
import net.proteanit.sql.DbUtils;  
  
// Importing Custom Classes  
import connections.Conn;  
import oop.Student;  
  
public class UpdateStudent extends JFrame implements ActionListener{  
 Choice studIDChoice;  
 JLabel lblMainHeading, lblHeading,  
 lblDepartment, lblCourse, lblDepartment2, lblCourse2,  
 lblFirstName, lblLastName, lblFirstName2, lblLastName2,  
 lblInternational, lblInternational2,  
 lblEmail, lblEmail2, lblIMMAStatus, lblIMMAStatus2, lblFeeStatus, lblFeeStatus2,  
 lblDOB, lblDOB2, lblContact, lblAddress;  
 TextField tfContact, tfAddress;  
 JButton btnUpdate, btnRemove, btnCancel;  
  
 public UpdateStudent(){  
  
 /\* 1. Form Heading \*/  
  
 // 1.1 Heading  
 lblMainHeading = new JLabel("Form: Update an existing Student Record");  
 lblMainHeading.setBounds(100, 10, 700, 50);  
 lblMainHeading.setFont(new Font("serif",Font.*BOLD*,30));  
 this.add(lblMainHeading);  
  
 // 1.2 Choice  
 lblHeading = new JLabel("Choose a Matriculation ID: ");  
 lblHeading.setBounds(50,100,170,20);  
 this.add(lblHeading);  
  
 studIDChoice = new Choice();  
 studIDChoice.setBounds(220,100,150,20);  
 this.add(studIDChoice);  
 studIDChoice.add("Select...");  
  
 // 1.3 Connection to get a list of choice of emp ids  
 try{  
 Conn c = new Conn();  
 String query = "select distinct imma\_id from STUDENT;";  
 ResultSet rs = c.stmt.executeQuery(query);  
 while(rs.next()){  
 studIDChoice.add(rs.getString("imma\_id"));  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
  
 /\* 2. Form Details \*/  
  
 // 2.1 First Name  
 lblFirstName = new JLabel("First Name: ");  
 lblFirstName.setBounds(50, 150, 150, 30);  
 lblFirstName.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblFirstName);  
  
 lblFirstName2 = new JLabel();  
 lblFirstName2.setBounds(200, 150, 150, 30);  
 lblFirstName2.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblFirstName2);  
  
 // 2.2 Last Name  
 lblLastName = new JLabel("Last Name: ");  
 lblLastName.setBounds(400, 150, 150, 30);  
 lblLastName.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblLastName);  
  
 lblLastName2 = new JLabel();  
 lblLastName2.setBounds(600, 150, 150, 30);  
 lblLastName2.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblLastName2);  
  
 // 2.3 Email  
 lblEmail = new JLabel("Email: ");  
 lblEmail.setBounds(50, 200, 150, 30);  
 lblEmail.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblEmail);  
  
 lblEmail2 = new JLabel();  
 lblEmail2.setBounds(200, 200, 500, 30);  
 lblEmail2.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblEmail2);  
  
 // 2.4 Matriculation Status  
 lblIMMAStatus = new JLabel("IMMA Status: ");  
 lblIMMAStatus.setBounds(50, 250, 150, 30);  
 lblIMMAStatus.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblIMMAStatus);  
  
 lblIMMAStatus2 = new JLabel();  
 lblIMMAStatus2.setBounds(200, 250, 150, 30);  
 lblIMMAStatus2.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblIMMAStatus2);  
  
 // 2.5 Fee Status  
 lblFeeStatus = new JLabel("Fees Status: ");  
 lblFeeStatus.setBounds(400, 250, 150, 30);  
 lblFeeStatus.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblFeeStatus);  
  
 lblFeeStatus2 = new JLabel();  
 lblFeeStatus2.setBounds(600, 250, 150, 30);  
 lblFeeStatus2.setFont(new Font("serif", Font.BOLD, 15));  
 this.add(lblFeeStatus2);  
  
 // 2.6 Contact  
 lblContact = new JLabel("Contact: ");  
 lblContact.setBounds(50, 300, 150, 30);  
 lblContact.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblContact);  
  
 tfContact = new TextField();  
 tfContact.setBounds(200, 300, 150, 30);  
 this.add(tfContact);  
  
 // 2.7 Date of Birth  
 lblDOB = new JLabel("Date of Birth");  
 lblDOB.setBounds(400, 300, 150, 30);  
 lblDOB.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblDOB);  
  
 lblDOB2 = new JLabel();  
 lblDOB2.setBounds(600, 300, 150, 30);  
 lblDOB2.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblDOB2);  
  
 // 2.8 International Student Status  
 lblInternational = new JLabel("International ?: ");  
 lblInternational.setBounds(400, 100, 150, 30);  
 lblInternational.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblInternational);  
  
 lblInternational2 = new JLabel();  
 lblInternational2.setBounds(600, 100, 150, 30);  
 lblInternational2.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblInternational2);  
  
 // 2.5 Address  
 lblAddress = new JLabel("Home Address: ");  
 lblAddress.setBounds(50, 350, 150, 30);  
 lblAddress.setFont(new Font("serif", Font.BOLD, 20));  
 this.add(lblAddress);  
  
 tfAddress = new TextField();  
 tfAddress.setBounds(200, 350, 550, 30);  
 this.add(tfAddress);  
  
 // 2.8 Department  
 lblDepartment = new JLabel("Department");  
 lblDepartment.setBounds(50,400,200,30);  
 lblDepartment.setFont(new Font("serif",Font.BOLD,20));  
 this.add(lblDepartment);  
  
 lblDepartment2 = new JLabel();  
 lblDepartment2.setBounds(200,400,200,30);  
 lblDepartment2.setFont(new Font("serif",Font.BOLD,20));  
 this.add(lblDepartment2);  
  
 // 2.9 Course  
 lblCourse = new JLabel("Course: ");  
 lblCourse.setBounds(400,400,150,30);  
 lblCourse.setFont(new Font("serif",Font.BOLD,20));  
 this.add(lblCourse);  
  
 lblCourse2 = new JLabel();  
 lblCourse2.setBounds(500,400,250,30);  
 lblCourse2.setFont(new Font("serif",Font.BOLD,20));  
 this.add(lblCourse2);  
  
 studIDChoice.addItemListener(new ItemListener() {  
 @Override  
 public void itemStateChanged(ItemEvent ie) {  
 try {  
 Conn c = new Conn();  
 String choiceItem = studIDChoice.getSelectedItem();  
 String query = """  
 SELECT DISTINCT  
 S.IMMA\_ID, S.FIRST\_NAME, S.LAST\_NAME, S.CONTACT, S.EMAIL, S.DOB, S.HOME\_ADDRESS, S.INTERNATIONAL\_STUDENT,  
 C.COURSE\_NAME, D.DEPT\_NAME,  
 E.IMMA\_DATE, E.IMMA\_STATUS, F.FEES\_PAID  
 FROM  
 STUDENT S  
 LEFT JOIN ENROLLMENT E  
 ON S.IMMA\_ID = E.IMMA\_ID  
 LEFT JOIN FEES F  
 ON S.IMMA\_ID = F.STUDENT\_ID  
 LEFT JOIN COURSE C  
 ON E.COURSE\_ID = C.COURSE\_ID  
 LEFT JOIN DEPARTMENT D  
 ON C.DEPT\_ID = D.DEPT\_ID  
 WHERE S.IMMA\_ID =  
 """ + choiceItem + ";";  
  
 ResultSet resultSet = c.stmt.executeQuery(query);  
  
 while (resultSet.next()) {  
 lblFirstName2.setText(resultSet.getString("first\_name"));  
 lblLastName2.setText(resultSet.getString("last\_name"));  
 lblEmail2.setText(resultSet.getString("email"));  
 lblDOB2.setText(resultSet.getString("dob"));  
 lblInternational2.setText(resultSet.getString("international\_student"));  
 lblCourse2.setText(resultSet.getString("course\_name"));  
 lblDepartment2.setText(resultSet.getString("dept\_name"));  
 lblIMMAStatus2.setText(resultSet.getString("imma\_status"));  
 lblFeeStatus2.setText(resultSet.getString("fees\_paid"));  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }}  
 });  
  
 /\* 3. Buttons \*/  
  
 // 3.1 Update  
 btnUpdate = new JButton("UPDATE");  
 btnUpdate.setBounds(250,550,120,30);  
 btnUpdate.setBackground(Color.BLACK);  
 btnUpdate.setForeground(Color.WHITE);  
 btnUpdate.addActionListener(this);  
 this.add(btnUpdate);  
  
 // 3.2 Remove  
 btnRemove = new JButton("DEREGISTER");  
 btnRemove.setBounds(450,550,120,30);  
 btnRemove.setBackground(Color.BLACK);  
 btnRemove.setForeground(Color.WHITE);  
 btnRemove.addActionListener(this);  
 this.add(btnRemove);  
  
 // 3.2 Cancel  
 btnCancel = new JButton("CANCEL");  
 btnCancel.setBounds(650,550,120,30);  
 btnCancel.setBackground(Color.BLACK);  
 btnCancel.setForeground(Color.WHITE);  
 btnCancel.addActionListener(this);  
 this.add(btnCancel);  
  
 /\* 4. JFrame Configurations \*/  
 this.setSize(900, 700);  
 this.setLocation(350, 50);  
 this.setLayout(null);  
 this.setVisible(true);  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent ae){  
 if(ae.getSource() == btnUpdate){  
 String choiceItem = studIDChoice.getSelectedItem();  
 if(!choiceItem.equals("Select...")){  
 int dialogButton = JOptionPane.*YES\_NO\_OPTION*;  
 int dialogResult = JOptionPane.*showConfirmDialog*(null, "WARNING: You are about to update " + choiceItem + "!","WARNING",dialogButton);  
  
 Conn c = new Conn();  
  
 try {  
 if(dialogResult == 0) {  
 // YES  
 String contact = tfContact.getText();  
 String address = tfAddress.getText();  
  
 c.conn.setAutoCommit(false);  
  
 Student existingStudent = new Student();  
  
 if(existingStudent.updateContactDB(c, choiceItem, contact, address)==1){  
 // COMMIT  
 c.conn.commit();  
  
 JOptionPane.*showMessageDialog*(null, "UPDATE: Student Record updated successfully!");  
 setVisible(false);  
 }  
 else{  
 // ROLLBACK  
 // Contact mistake  
 c.conn.rollback();  
 JOptionPane.showMessageDialog(null, "ERROR: Contact must follow german phone standards!");  
 };  
 }  
 else {  
 // NO ACTION ITEM  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
 else{  
 JOptionPane.showMessageDialog(null, "Select a student matriculation number to update!");  
 }  
 }  
 else if(ae.getSource() == btnRemove){  
 String choiceItem = studIDChoice.getSelectedItem();  
 if(!choiceItem.equals("Select...")){  
 int dialogButton = JOptionPane.YES\_NO\_OPTION;  
 int dialogResult = JOptionPane.showConfirmDialog(null, "WARNING: You are about to remove " + choiceItem + "!","WARNING",dialogButton);  
  
 if(dialogResult == 0) {  
 // YES  
 Conn c = new Conn();  
  
 try {  
  
 c.conn.setAutoCommit(false);  
  
 String query = """  
 DELETE FROM STUDENT  
 WHERE IMMA\_ID =  
 """  
 + choiceItem + ";";  
  
 c.stmt.executeUpdate(query);  
 c.conn.commit();  
  
 JOptionPane.showMessageDialog(null, "UPDATE: Student Record deleted successfully!");  
 setVisible(false);  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
 else {  
 // NO ACTION ITEM  
 }  
 }  
 else{  
 JOptionPane.showMessageDialog(null, "ERROR: Select a student matriculation number to delete!");  
 }  
 }  
 else{  
 this.setVisible(false);  
 }  
 }  
  
 public static void main(String[] args) {  
 new UpdateStudent();  
 }  
}

### 7.1.14 Java Swings Class to view an existing student record – ViewStudent.java

package student;  
  
*/\*\*  
 @author Sanjay Prabhu Kunjibettu  
 @author Tanay Khilare  
 \*/*// General Libraries  
import java.awt.\*;  
import javax.swing.\*;  
import javax.swing.table.TableColumn;  
import javax.swing.table.TableColumnModel;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.sql.ResultSet;  
import java.sql.ResultSetMetaData;  
  
// Plugins  
import net.proteanit.sql.DbUtils; // rs2xml.jar  
  
// Importing Custom Classes  
import connections.Conn;  
  
public class ViewStudent extends JFrame implements ActionListener {  
 Choice studIDChoice;  
 JLabel lblHeading;  
 JTable tableAll;  
 JButton btnSearch, btnPrint, btnCancel;  
 JScrollPane scrollPane;  
  
 public ViewStudent(){  
  
 // 0. Set Border Layout  
 // Border Layout works best for tables, scrolling and printing  
 this.setLayout(new BorderLayout());  
  
 /\* 1. Select the Student Matriculation ID to view their details \*/  
  
 // 1.1 Label  
 lblHeading = new JLabel("Choose a Matriculation ID: ");  
  
 // 1.2 Choice  
 studIDChoice = new Choice();  
 studIDChoice.add("All");  
  
 // 1.3 Connection to get a list of choice of emp ids  
 try{  
 Conn c = new Conn();  
 String query = "select distinct imma\_id from STUDENT;";  
 ResultSet rs = c.stmt.executeQuery(query);  
 while(rs.next()){  
 studIDChoice.add(rs.getString("imma\_id"));  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
  
 // 1.4 Connection to get student info with course and dept info  
 tableAll = new JTable();  
 try{  
 Conn c = new Conn();  
 String query = """  
 SELECT DISTINCT  
 S.IMMA\_ID, S.FIRST\_NAME, S.LAST\_NAME, S.CONTACT, S.EMAIL, S.DOB, S.HOME\_ADDRESS, S.INTERNATIONAL\_STUDENT,  
 C.COURSE\_NAME, D.DEPT\_NAME,  
 E.IMMA\_DATE, E.IMMA\_STATUS, F.FEES\_PAID  
 FROM  
 STUDENT S  
 LEFT JOIN ENROLLMENT E  
 ON S.IMMA\_ID = E.IMMA\_ID  
 LEFT JOIN FEES F  
 ON S.IMMA\_ID = F.STUDENT\_ID  
 LEFT JOIN COURSE C  
 ON E.COURSE\_ID = C.COURSE\_ID  
 LEFT JOIN DEPARTMENT D  
 ON C.DEPT\_ID = D.DEPT\_ID;  
 """;  
 ResultSet rs = c.stmt.executeQuery(query);  
 ResultSetMetaData metaData = rs.getMetaData();  
 int totalColumns = metaData.getColumnCount();  
  
 tableAll.setModel(DbUtils.*resultSetToTableModel*(rs));  
 tableAll.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*);  
  
 // Set column size  
 TableColumnModel colModel = tableAll.getColumnModel();  
 for(int i=0; i<totalColumns; i++){  
 TableColumn tc = colModel.getColumn(i);  
 tc.setPreferredWidth(130);  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
  
 // 1.5 Scroll Pane  
 scrollPane = new JScrollPane(tableAll, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_ALWAYS*);  
  
 // 1.6 Border Layout for Scroll Pane  
 JPanel centerPanel = new JPanel();  
 centerPanel.add(scrollPane);  
 centerPanel.setPreferredSize(new Dimension(900,(int)centerPanel.getPreferredSize().getHeight()));  
 centerPanel.setLayout(new GridLayout(1, 3));  
 this.add(centerPanel, BorderLayout.*CENTER*);  
  
 /\* 2. Buttons \*/  
  
 // 2.1 Search  
 btnSearch = new JButton("SEARCH");  
 btnSearch.addActionListener(this);  
  
 // 2.2 Print  
 btnPrint = new JButton("PRINT");  
 btnPrint.addActionListener(this);  
  
 // 2.3 Cancel the operation  
 btnCancel = new JButton("CANCEL");  
 btnCancel.addActionListener(this);  
  
 // 2.4 North Border Layout  
 JPanel northPanel = new JPanel();  
 northPanel.add(lblHeading);  
 northPanel.add(studIDChoice);  
 northPanel.add(btnPrint);  
 northPanel.add(btnSearch);  
 northPanel.add(btnCancel);  
 northPanel.setPreferredSize(new Dimension(900,(int)northPanel.getPreferredSize().getHeight()));  
 this.add(northPanel, BorderLayout.*NORTH*);  
  
 /\* 3. JFrame Configurations \*/  
  
 this.setTitle("Student Details");  
 this.pack();  
 this.setSize(900, 600);  
 this.setLocation(350, 50);  
 this.setVisible(true);  
  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent ae){  
 if (ae.getSource() == btnSearch){  
 try {  
 Conn c = new Conn();  
 String choiceItem = studIDChoice.getSelectedItem();  
 if(!choiceItem.equals("All")){  
 String query = """  
 SELECT DISTINCT  
 S.IMMA\_ID, S.FIRST\_NAME, S.LAST\_NAME, S.CONTACT, S.EMAIL, S.DOB, S.HOME\_ADDRESS, S.INTERNATIONAL\_STUDENT,  
 C.COURSE\_NAME, D.DEPT\_NAME,  
 E.IMMA\_DATE, E.IMMA\_STATUS, F.FEES\_PAID  
 FROM  
 STUDENT S  
 LEFT JOIN ENROLLMENT E  
 ON S.IMMA\_ID = E.IMMA\_ID  
 LEFT JOIN FEES F  
 ON S.IMMA\_ID = F.STUDENT\_ID  
 LEFT JOIN COURSE C  
 ON E.COURSE\_ID = C.COURSE\_ID  
 LEFT JOIN DEPARTMENT D  
 ON C.DEPT\_ID = D.DEPT\_ID  
 WHERE S.IMMA\_ID =  
 """ + choiceItem + ";";  
  
 ResultSet resultSet = c.stmt.executeQuery(query);  
  
 ResultSetMetaData metaData = resultSet.getMetaData();  
 int totalColumns = metaData.getColumnCount();  
  
 tableAll.setModel(DbUtils.*resultSetToTableModel*(resultSet));  
 tableAll.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*);  
  
 // Set column size  
 TableColumnModel colModel = tableAll.getColumnModel();  
 for(int i=0; i<totalColumns; i++){  
 TableColumn tc = colModel.getColumn(i);  
 tc.setPreferredWidth(130);  
 }  
  
 }  
 else{  
 String query = """  
 SELECT DISTINCT  
 S.IMMA\_ID, S.FIRST\_NAME, S.LAST\_NAME, S.CONTACT, S.EMAIL, S.DOB, S.HOME\_ADDRESS, S.INTERNATIONAL\_STUDENT,  
 C.COURSE\_NAME, D.DEPT\_NAME,  
 E.IMMA\_DATE, E.IMMA\_STATUS, F.FEES\_PAID  
 FROM  
 STUDENT S  
 LEFT JOIN ENROLLMENT E  
 ON S.IMMA\_ID = E.IMMA\_ID  
 ON S.IMMA\_ID = F.STUDENT\_ID  
 LEFT JOIN COURSE C  
 ON E.COURSE\_ID = C.COURSE\_ID  
 LEFT JOIN DEPARTMENT D  
 ON C.DEPT\_ID = D.DEPT\_ID;  
 """;  
  
 ResultSet resultSet = c.stmt.executeQuery(query);  
  
 ResultSetMetaData metaData = resultSet.getMetaData();  
 int totalColumns = metaData.getColumnCount();  
  
 tableAll.setModel(DbUtils.*resultSetToTableModel*(resultSet));  
 tableAll.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*);  
  
 // Set column size  
 TableColumnModel colModel = tableAll.getColumnModel();  
 for(int i=0; i<totalColumns; i++){  
 TableColumn tc = colModel.getColumn(i);  
 tc.setPreferredWidth(130);  
 }  
  
 }  
 }catch (Exception E){  
 E.printStackTrace();  
 }  
 }  
 else if (ae.getSource() == btnPrint){  
 try {  
 tableAll.print();  
 }catch (Exception e) {  
 e.printStackTrace();  
 }  
 }else {  
 // Cancel Transaction  
 setVisible(false);  
 }  
 }  
  
 public static void main(String[] args) {  
 new ViewStudent();  
 }  
}

### 7.1.15 Java Swings Class to view the fee structures – FeeStructure.java

package student;  
  
// Only illustrates the structure of Fees  
// Includes the necessary details for the students  
  
*/\*\*  
 @author Sanjay Prabhu Kunjibettu  
 @author Tanay Khilare  
 \*/*// General Libraries  
import java.awt.\*;  
import javax.swing.\*;  
  
public class FeeStructure extends JFrame{  
 JLabel lblMainHeading, lblHeading,  
 lblTuitionFees, lblTuitionFees2,  
 lblSemesterFees, lblSemesterFees2,  
 lblDTicket, lblDTicket2, lblVariable, lblVariable2;  
  
 public FeeStructure(){  
 /\* 1. Form Heading \*/  
  
 // 1.1 Heading  
 lblMainHeading = new JLabel("Fee Structure for Students");  
 lblMainHeading.setBounds(230, 10, 500, 50);  
 lblMainHeading.setFont(new Font("serif",Font.*BOLD*,30));  
 this.add(lblMainHeading);  
  
 /\* 2. Form Details \*/  
  
 // 2.1 Tuition Fees  
 lblTuitionFees = new JLabel("Tuition Fees: ");  
 lblTuitionFees.setBounds(50, 100, 150, 30);  
 lblTuitionFees.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblTuitionFees);  
  
 lblTuitionFees2 = new JLabel("Only for International Students - €1500,00 / Semester");  
 lblTuitionFees2.setBounds(250, 100, 500, 30);  
 lblTuitionFees2.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblTuitionFees2);  
  
 // 2.2 Semester Fees  
 lblSemesterFees = new JLabel("Semester Fees: ");  
 lblSemesterFees.setBounds(50, 150, 150, 30);  
 lblSemesterFees.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblSemesterFees);  
  
 lblSemesterFees2 = new JLabel("For All - € 200,00 / Semester");  
 lblSemesterFees2.setBounds(250, 150, 500, 30);  
 lblSemesterFees2.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblSemesterFees2);  
  
 // 2.3 D-Ticket (Optional)  
 lblDTicket = new JLabel("D-Ticket: ");  
 lblDTicket.setBounds(50, 200, 150, 30);  
 lblDTicket.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblDTicket);  
  
 lblDTicket2 = new JLabel("Optional - € 180,00 / Semester");  
 lblDTicket2.setBounds(250, 200, 500, 30);  
 lblDTicket2.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblDTicket2);  
  
 // 2.4 Variable Fees  
 lblVariable = new JLabel("Variables: ");  
 lblVariable.setBounds(50, 250, 150, 30);  
 lblVariable.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblVariable);  
  
 lblVariable2 = new JLabel("Includes any variable fees / fines");  
 lblVariable2.setBounds(250, 250, 500, 30);  
 lblVariable2.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblVariable2);  
  
 /\* 3. JFrame Configurations \*/  
 this.setSize(900, 700);  
 this.setLocation(350, 50);  
 this.setLayout(null);  
 this.setVisible(true);  
 }  
  
 public static void main(String[] args) {  
 new FeeStructure();  
 }  
}

### 7.1.16 Java Swings Class to pay the fees – FeesForm.java

package student;  
  
*/\*\*  
 @author Sanjay Prabhu Kunjibettu  
 @author Tanay Khilare  
 \*/*// General Libraries  
import java.awt.\*;  
import javax.swing.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.ItemEvent;  
import java.awt.event.ItemListener;  
import java.util.Random;  
import java.sql.\*;  
import java.time.LocalDate;  
  
// Plugins  
import net.proteanit.sql.DbUtils;  
  
  
// Importing Custom Classes  
import connections.Conn;  
import oop.Student;  
  
public class FeesForm extends JFrame implements ActionListener{  
 Choice studIDChoice;  
 JLabel lblMainHeading, lblHeading,  
 lblTuitionFees, lblTuitionFees2,  
 lblSemesterFees, lblSemesterFees2,  
 lblDTicket, lblVariable,  
 lblFeesPaid, lblTotalFees;  
 JTextField tfVariable;  
 JComboBox cbDTicket;  
 JButton btnPay, btnCancel, btnReset, btnCalculate;  
  
 public FeesForm(){  
 /\* 1. Form Heading \*/  
  
 // 1.1 Heading  
 lblMainHeading = new JLabel("Form: Student Fees");  
 lblMainHeading.setBounds(230, 10, 500, 50);  
 lblMainHeading.setFont(new Font("serif",Font.*BOLD*,30));  
 this.add(lblMainHeading);  
  
 // 1.2 Choice  
 lblHeading = new JLabel("Choose a Matriculation ID: ");  
 lblHeading.setBounds(20,60,170,20);  
 this.add(lblHeading);  
  
 studIDChoice = new Choice();  
 studIDChoice.setBounds(190,60,150,20);  
 this.add(studIDChoice);  
 studIDChoice.add("Select...");  
  
 // 1.3 Connection to get a list of choice of emp ids  
 try{  
 Conn c = new Conn();  
 String query = "select distinct imma\_id from STUDENT;";  
 ResultSet rs = c.stmt.executeQuery(query);  
 while(rs.next()){  
 studIDChoice.add(rs.getString("imma\_id"));  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
  
 /\* 2. Form Details \*/  
  
 // 2.1 Fees Status and Total Fees  
 lblFeesPaid = new JLabel();  
 lblFeesPaid.setBounds(50, 100, 150, 30);  
 lblFeesPaid.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblFeesPaid);  
  
 lblTotalFees = new JLabel("NOT CALCULATED");  
 lblTotalFees.setBounds(500, 100, 200, 30);  
 lblTotalFees.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblTotalFees);  
  
  
 // 2.2 Tuition Fees  
 lblTuitionFees = new JLabel("Tuition Fees: ");  
 lblTuitionFees.setBounds(50, 150, 150, 30);  
 lblTuitionFees.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblTuitionFees);  
  
 lblTuitionFees2 = new JLabel();  
 lblTuitionFees2.setBounds(200, 150, 150, 30);  
 lblTuitionFees2.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblTuitionFees2);  
  
 // 2.3 Semester Fees  
 lblSemesterFees = new JLabel("Semester Fees: ");  
 lblSemesterFees.setBounds(400, 150, 150, 30);  
 lblSemesterFees.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblSemesterFees);  
  
 lblSemesterFees2 = new JLabel();  
 lblSemesterFees2.setBounds(600, 150, 150, 30);  
 lblSemesterFees2.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblSemesterFees2);  
  
 // 2.4 D-Ticket (Optional)  
 lblDTicket = new JLabel("D-Ticket: ");  
 lblDTicket.setBounds(50, 200, 150, 30);  
 lblDTicket.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblDTicket);  
  
 cbDTicket = new JComboBox(new String[]{"YES", "NO"});  
 cbDTicket.setBounds(200, 200, 150, 30);  
 cbDTicket.setFont(new Font("serif", Font.*BOLD*, 15));  
 this.add(cbDTicket);  
  
 // 2.5 Variable Fees  
 lblVariable = new JLabel("Variables: ");  
 lblVariable.setBounds(400, 200, 150, 30);  
 lblVariable.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblVariable);  
  
 tfVariable = new JTextField();  
 tfVariable.setBounds(600, 200, 150, 30);  
 tfVariable.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(tfVariable);  
  
 studIDChoice.addItemListener(new ItemListener() {  
 @Override  
 public void itemStateChanged(ItemEvent ie) {  
 try {  
 Conn c = new Conn();  
 String choiceItem = studIDChoice.getSelectedItem();  
 String query = """  
 SELECT \*  
 FROM  
 FEES  
 WHERE STUDENT\_ID =  
 """ + choiceItem + ";";  
  
 ResultSet resultSet = c.stmt.executeQuery(query);  
  
 while (resultSet.next()) {  
 lblFeesPaid.setText("Fees Paid?: " + resultSet.getString("fees\_paid"));  
 lblTuitionFees2.setText(resultSet.getString("tuition\_fees"));  
 lblSemesterFees2.setText(resultSet.getString("semester\_fees"));  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }}  
 });  
  
 /\* 3. Buttons \*/  
  
 // 3.1 Calculate  
 btnCalculate = new JButton("CALCULATE");  
 btnCalculate.setBounds(100,250,120,30);  
 btnCalculate.setBackground(Color.*BLACK*);  
 btnCalculate.setForeground(Color.*WHITE*);  
 btnCalculate.addActionListener(this);  
 this.add(btnCalculate);  
  
 // 3.2 Pay  
 btnPay = new JButton("PAY");  
 btnPay.setBounds(250,250,120,30);  
 btnPay.setBackground(Color.*BLACK*);  
 btnPay.setForeground(Color.*WHITE*);  
 btnPay.addActionListener(this);  
 this.add(btnPay);  
  
 // 3.3 Cancel  
 btnCancel = new JButton("CANCEL");  
 btnCancel.setBounds(400,250,120,30);  
 btnCancel.setBackground(Color.*BLACK*);  
 btnCancel.setForeground(Color.*WHITE*);  
 btnCancel.addActionListener(this);  
 this.add(btnCancel);  
  
 // 3.4 Reset  
 btnReset = new JButton("RESET");  
 btnReset.setBounds(550,250,120,30);  
 btnReset.setBackground(Color.*BLACK*);  
 btnReset.setForeground(Color.*WHITE*);  
 btnReset.addActionListener(this);  
 this.add(btnReset);  
  
 /\* 4. JFrame Configurations \*/  
 this.setSize(900, 450);  
 this.setLocation(350, 50);  
 this.setLayout(null);  
 this.setVisible(true);  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent ae){  
 if(!studIDChoice.getSelectedItem().equals("Select...")){  
 if(ae.getSource() == btnCalculate){  
 double tuitionFees = Double.*parseDouble*(lblTuitionFees2.getText());  
 double semesterFees = Double.*parseDouble*(lblSemesterFees2.getText());  
  
 // D Ticket  
 String dTicket = String.*valueOf*(cbDTicket.getSelectedItem());  
 double d\_ticket = (dTicket.equals("YES") ? 180 : 0);  
  
 double variables = Double.*parseDouble*(  
 (tfVariable.getText().isEmpty() ? "0" : tfVariable.getText())  
 );  
  
 double totalFees = tuitionFees + semesterFees + d\_ticket + variables;  
  
 lblTotalFees.setText("Total: €" + String.*valueOf*(totalFees));  
 }  
 else if(ae.getSource() == btnPay){  
 if(lblTotalFees.getText().equals("NOT CALCULATED")){  
 JOptionPane.*showMessageDialog*(null, "Calculate the amount first!");  
 }  
 else{  
 // D Ticket  
 String dTicket = String.*valueOf*(cbDTicket.getSelectedItem());  
 double d\_ticket = (dTicket.equals("YES") ? 360 : 0);  
  
 double variables = Double.*parseDouble*(  
 (tfVariable.getText().isEmpty() ? "0" : tfVariable.getText())  
 );  
  
 Conn c = new Conn();  
 Student s = new Student();  
 String immaID = studIDChoice.getSelectedItem();  
 System.*out*.println(immaID);  
  
 if(s.updateFeesDB(c, immaID, String.*valueOf*(d\_ticket), String.*valueOf*(variables))==1){  
 try {  
 c.conn.commit();  
  
 String query = "UPDATE ENROLLMENT SET IMMA\_STATUS = 'ENROLLED' WHERE IMMA\_ID = '" + immaID + "';";  
 c.stmt.executeUpdate(query);  
  
 c.conn.commit();  
  
 JOptionPane.*showMessageDialog*(null, "Fees paid!");  
 this.setVisible(false);  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
 else {  
 JOptionPane.*showMessageDialog*(null, "The student has already paid the fees!");  
 }  
 }  
 }  
 else if(ae.getSource() == btnReset){  
 Conn c = new Conn();  
  
 try{  
 c.conn.setAutoCommit(false);  
  
 String immaID = studIDChoice.getSelectedItem();  
  
 // TRANSACTIONS - COMMIT ROLLBACK  
 try{  
 String query = "UPDATE FEES SET D\_TICKET = '0', RESEARCH\_VARIABLE = '0', FEES\_PAID='NO';";  
 c.stmt.executeUpdate(query);  
 c.conn.commit();  
  
 query = "UPDATE ENROLLMENT SET IMMA\_STATUS = 'ADMITTED' WHERE IMMA\_ID = '" + immaID + "';";  
 c.stmt.executeUpdate(query);  
 c.conn.commit();  
 }  
 catch (Exception ee){  
 c.conn.rollback();  
 }  
  
 JOptionPane.*showMessageDialog*(null, "Warning: Fees Status changed to NOT PAID! ");  
 this.setVisible(false);  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
 else{  
 this.setVisible(false);  
 }  
 }  
 else{  
 JOptionPane.*showMessageDialog*(null, "Select a Student Matriculation Number!");  
  
 }  
  
 }  
  
 public static void main(String[] args) {  
 new FeesForm();  
 }  
}

### 7.1.17 Java Swings Class to insert grades for subjects – InsertGrades.java

package grades;  
  
*/\*\*  
 @author Sanjay Prabhu Kunjibettu  
 @author Tanay Khilare  
 \*/*// General Libraries  
import java.awt.\*;  
import javax.swing.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.ItemEvent;  
import java.awt.event.ItemListener;  
import java.sql.ResultSet;  
import java.util.HashMap;  
import java.util.Random;  
  
// Plugins  
import com.toedter.calendar.JDateChooser;  
import connections.Conn;  
  
// Importing Custom Classes  
  
public class InsertGrades extends JFrame implements ActionListener{  
 Choice studIDChoice;  
 JComboBox cbSemester;  
 JLabel lblMainHeading, lblHeading, lblSemester,  
 lblSubject, lblGrades, lblCourse;  
 String courseID;  
 JTextField tfSubject1, tfSubject2, tfSubject3, tfSubject4, tfSubject5, tfElective1,  
 tfGrade1, tfGrade2, tfGrade3, tfGrade4, tfGrade5, tfGrade6;  
 JButton btnSubmit, btnCancel;  
  
 public InsertGrades(){  
 /\* 1. Form Heading \*/  
  
 // 1.1 Heading  
 lblMainHeading = new JLabel("Form: Insert Grades for a Student");  
 lblMainHeading.setBounds(230, 10, 500, 50);  
 lblMainHeading.setFont(new Font("serif",Font.*BOLD*,30));  
 this.add(lblMainHeading);  
  
 // 1.2 Choice  
 lblHeading = new JLabel("Choose a Matriculation ID: ");  
 lblHeading.setBounds(50,60,200,20);  
 this.add(lblHeading);  
  
 studIDChoice = new Choice();  
 studIDChoice.setBounds(270,60,150,20);  
 this.add(studIDChoice);  
 studIDChoice.add("Select...");  
  
 // 1.3 Connection to get a list of choice of emp ids  
 try{  
 Conn c = new Conn();  
 String query = "select distinct imma\_id from STUDENT;";  
 ResultSet rs = c.stmt.executeQuery(query);  
 while(rs.next()){  
 studIDChoice.add(rs.getString("imma\_id"));  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
  
 // Course Label  
 lblCourse = new JLabel();  
 lblCourse.setBounds(20, 100, 150, 20);  
 this.add(lblCourse);  
  
 // Get the course id  
 studIDChoice.addItemListener(new ItemListener() {  
 @Override  
 public void itemStateChanged(ItemEvent ie) {  
 try {  
 Conn c = new Conn();  
 String choiceItem = studIDChoice.getSelectedItem();  
 String query = """  
 SELECT DISTINCT  
 C.COURSE\_ID, C.COURSE\_NAME  
 FROM  
 STUDENT S  
 LEFT JOIN ENROLLMENT E  
 ON S.IMMA\_ID = E.IMMA\_ID  
 LEFT JOIN COURSE C  
 ON E.COURSE\_ID = C.COURSE\_ID  
 WHERE S.IMMA\_ID =  
 """ + choiceItem + ";";  
  
 ResultSet resultSet = c.stmt.executeQuery(query);  
  
 while (resultSet.next()) {  
 lblCourse.setText(resultSet.getString("course\_name"));  
 courseID = resultSet.getString("course\_id");  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }}  
 });  
  
 /\* 2. Form Details \*/  
  
 // 2.1 Select Semester  
 lblSemester = new JLabel("Select Semester: ");  
 lblSemester.setBounds(50,110,150,20);  
 this.add(lblSemester);  
  
 String[] semester = {"1","2","3","4","5","6","7","8"};  
 cbSemester = new JComboBox(semester);  
 cbSemester.setBounds(270,110,150,20);  
 cbSemester.setBackground(Color.*WHITE*);  
 this.add(cbSemester);  
  
 // 2.2 Headings for Subjects and Marks  
 lblSubject = new JLabel("Subject: ");  
 lblSubject.setBounds(100,150,200,40);  
 lblSubject.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblSubject);  
  
 lblGrades = new JLabel("Grades: ");  
 lblGrades.setBounds(400,150,200,40);  
 lblGrades.setFont(new Font("serif", Font.*BOLD*, 20));  
 this.add(lblGrades);  
  
 // 2.3 Subjects  
 tfSubject1 = new JTextField();  
 tfSubject1.setBounds(100,200,200,20);  
 this.add(tfSubject1);  
  
 tfSubject2 = new JTextField();  
 tfSubject2.setBounds(100,230,200,20);  
 this.add(tfSubject2);  
  
 tfSubject3 = new JTextField();  
 tfSubject3.setBounds(100,260,200,20);  
 this.add(tfSubject3);  
  
 tfSubject4 = new JTextField();  
 tfSubject4.setBounds(100,290,200,20);  
 this.add(tfSubject4);  
  
 tfSubject5 = new JTextField();  
 tfSubject5.setBounds(100,320,200,20);  
 this.add(tfSubject5);  
  
 tfElective1 = new JTextField();  
 tfElective1.setBounds(100,350,200,20);  
 this.add(tfElective1);  
  
 // 2.4 Grades  
 tfGrade1 = new JTextField();  
 tfGrade1.setBounds(350,200,200,20);  
 this.add(tfGrade1);  
  
 tfGrade2 = new JTextField();  
 tfGrade2.setBounds(350,230,200,20);  
 this.add(tfGrade2);  
  
 tfGrade3= new JTextField();  
 tfGrade3.setBounds(350,260,200,20);  
 this.add(tfGrade3);  
  
 tfGrade4 = new JTextField();  
 tfGrade4.setBounds(350,290,200,20);  
 this.add(tfGrade4);  
  
 tfGrade5 = new JTextField();  
 tfGrade5.setBounds(350,320,200,20);  
 this.add(tfGrade5);  
  
 tfGrade6 = new JTextField();  
 tfGrade6.setBounds(350,350,200,20);  
 this.add(tfGrade6);  
  
 /\* 3. Buttons \*/  
  
 // 3.1 Submit  
 // 3.1 Insert  
 btnSubmit = new JButton("INSERT");  
 btnSubmit.setBounds(150,400,120,30);  
 btnSubmit.setBackground(Color.*BLACK*);  
 btnSubmit.setForeground(Color.*WHITE*);  
 btnSubmit.addActionListener(this);  
 this.add(btnSubmit);  
  
 // 3.2 Cancel  
 btnCancel = new JButton("CANCEL");  
 btnCancel.setBounds(370,400,120,30);  
 btnCancel.setBackground(Color.*BLACK*);  
 btnCancel.setForeground(Color.*WHITE*);  
 btnCancel.addActionListener(this);  
 this.add(btnCancel);  
  
 /\* 4. JFrame Configurations \*/  
 this.setSize(900, 700);  
 this.setLocation(350, 50);  
 this.setLayout(null);  
 this.setVisible(true);  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent ae){  
 if(ae.getSource() == btnSubmit){  
 String choiceItem = studIDChoice.getSelectedItem();  
 if(!choiceItem.equals("Select...")){  
  
 Conn c = new Conn();  
  
 try {  
  
 c.conn.setAutoCommit(false);  
  
 // Check if there are no duplicates  
 String query = """  
 select count(\*) student\_id from SUBJECT  
 where student\_id = '"""  
 + choiceItem + "' and semester = '" + cbSemester.getSelectedItem() + "';";  
  
 ResultSet rs = c.stmt.executeQuery(query);  
 String rowCount="";  
  
 while(rs.next()){  
 rowCount = rs.getString("student\_id");  
 }  
  
 if(rowCount.equals("0")){  
 query = "INSERT INTO SUBJECT VALUES('" + choiceItem + "', '" + courseID + "', '" + cbSemester.getSelectedItem() + "', '" +  
 tfSubject1.getText() + "', '" + tfSubject2.getText() + "', '" + tfSubject3.getText() + "', '" + tfSubject4.getText() + "', '" + tfSubject5.getText() + "', '" + tfElective1.getText() + "');";  
  
 c.stmt.executeUpdate(query);  
 c.conn.commit();  
  
 query = "INSERT INTO GRADES VALUES('" + choiceItem + "', '" + courseID + "', '" + cbSemester.getSelectedItem() + "', '" +  
 tfGrade1.getText() + "', '" + tfGrade2.getText() + "', '" + tfGrade3.getText() + "', '" + tfGrade4.getText() + "', '" + tfGrade5.getText() + "', '" + tfGrade6.getText() + "');";  
 c.stmt.executeUpdate(query);  
 c.conn.commit();  
  
 JOptionPane.*showMessageDialog*(null, "Grades inserted successfully!");  
 setVisible(false);  
 }  
 else{  
 JOptionPane.*showMessageDialog*(null, "Value exists for the selected student and semester! Please try again!");  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
 else{  
 JOptionPane.*showMessageDialog*(null, "Select a student matriculation number to insert!");  
 }  
 }  
 else{  
 this.setVisible(false);  
 }  
 }  
  
 public static void main(String[] args) {  
 new InsertGrades();  
 }  
}

### 7.1.18 Java Swings Class to view results – ViewGrades.java

package grades;  
  
*/\*\*  
 @author Sanjay Prabhu Kunjibettu  
 @author Tanay Khilare  
 \*/*// General Libraries  
import java.awt.\*;  
import javax.swing.\*;  
import javax.swing.table.TableColumn;  
import javax.swing.table.TableColumnModel;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.sql.ResultSet;  
import java.sql.ResultSetMetaData;  
  
// Import Plugins  
import net.proteanit.sql.DbUtils; // rs2xml.jar  
  
// Importing Custom Classes  
import connections.Conn;  
  
public class ViewGrades extends JFrame implements ActionListener{  
 Choice studIDChoice;  
 JLabel lblHeading;  
 JTable tableAll;  
 JButton btnSearch, btnPrint, btnCancel;  
 JScrollPane scrollPane;  
  
 public ViewGrades(){  
  
 // 0. Set Border Layout  
 // Border Layout works best for tables, scrolling and printing  
 this.setLayout(new BorderLayout());  
  
 /\* 1. Select the Student Matriculation ID to view their details \*/  
  
 // 1.1 Label  
 lblHeading = new JLabel("Choose a Matriculation ID: ");  
  
 // 1.2 Choice  
 studIDChoice = new Choice();  
 studIDChoice.add("All");  
  
 // 1.3 Connection to get a list of choice of emp ids  
 try{  
 Conn c = new Conn();  
 String query = "select distinct imma\_id from STUDENT;";  
 ResultSet rs = c.stmt.executeQuery(query);  
 while(rs.next()){  
 studIDChoice.add(rs.getString("imma\_id"));  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
  
 // 1.4 Connection to get student info with course and dept info  
 tableAll = new JTable();  
 try{  
 Conn c = new Conn();  
 String query = """  
 SELECT DISTINCT S.FIRST\_NAME, S.LAST\_NAME, C.COURSE\_NAME, SUB.\*, G.\*  
 FROM  
 SUBJECT SUB  
 LEFT JOIN GRADES G  
 ON SUB.STUDENT\_ID = G.STUDENT\_ID and SUB.COURSE\_ID = G.COURSE\_ID and SUB.SEMESTER = G.SEMESTER  
 LEFT JOIN STUDENT S  
 ON S.IMMA\_ID = SUB.STUDENT\_ID  
 LEFT JOIN ENROLLMENT E  
 ON S.IMMA\_ID = E.IMMA\_ID  
 LEFT JOIN COURSE C  
 ON E.COURSE\_ID = C.COURSE\_ID  
 """;  
 ResultSet rs = c.stmt.executeQuery(query);  
 ResultSetMetaData metaData = rs.getMetaData();  
 int totalColumns = metaData.getColumnCount();  
  
 tableAll.setModel(DbUtils.*resultSetToTableModel*(rs));  
 tableAll.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*);  
  
 // Set column size  
 TableColumnModel colModel = tableAll.getColumnModel();  
 for(int i=0; i<totalColumns; i++){  
 TableColumn tc = colModel.getColumn(i);  
 tc.setPreferredWidth(130);  
 }  
 }  
 catch (Exception e){  
 e.printStackTrace();  
 }  
  
 // 1.5 Scroll Pane  
 scrollPane = new JScrollPane(tableAll, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_ALWAYS*);  
  
 // 1.6 Border Layout for Scroll Pane  
 JPanel centerPanel = new JPanel();  
 centerPanel.add(scrollPane);  
 centerPanel.setPreferredSize(new Dimension(900,(int)centerPanel.getPreferredSize().getHeight()));  
 centerPanel.setLayout(new GridLayout(1, 3));  
 this.add(centerPanel, BorderLayout.*CENTER*);  
  
 /\* 2. Buttons \*/  
  
 // 2.1 Search  
 btnSearch = new JButton("SEARCH");  
 btnSearch.addActionListener(this);  
  
 // 2.2 Print  
 btnPrint = new JButton("PRINT");  
 btnPrint.addActionListener(this);  
  
 // 2.3 Cancel the operation  
 btnCancel = new JButton("CANCEL");  
 btnCancel.addActionListener(this);  
  
 // 2.4 North Border Layout  
 JPanel northPanel = new JPanel();  
 northPanel.add(lblHeading);  
 northPanel.add(studIDChoice);  
 northPanel.add(btnPrint);  
 northPanel.add(btnSearch);  
 northPanel.add(btnCancel);  
 northPanel.setPreferredSize(new Dimension(900,(int)northPanel.getPreferredSize().getHeight()));  
 this.add(northPanel, BorderLayout.*NORTH*);  
  
 /\* 3. JFrame Configurations \*/  
  
 this.setTitle("Student Results Details");  
 this.pack();  
 this.setSize(900, 600);  
 this.setLocation(350, 50);  
 this.setVisible(true);  
  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent ae){  
 if (ae.getSource() == btnSearch){  
 try {  
 Conn c = new Conn();  
 String choiceItem = studIDChoice.getSelectedItem();  
 if(!choiceItem.equals("All")){  
 String query = """  
 SELECT DISTINCT S.FIRST\_NAME, S.LAST\_NAME, C.COURSE\_NAME, SUB.\*, G.\*  
 FROM  
 SUBJECT SUB  
 LEFT JOIN GRADES G  
 ON SUB.STUDENT\_ID = G.STUDENT\_ID and SUB.COURSE\_ID = G.COURSE\_ID and SUB.SEMESTER = G.SEMESTER  
 LEFT JOIN STUDENT S  
 ON S.IMMA\_ID = SUB.STUDENT\_ID  
 LEFT JOIN ENROLLMENT E  
 ON S.IMMA\_ID = E.IMMA\_ID  
 LEFT JOIN COURSE C  
 ON E.COURSE\_ID = C.COURSE\_ID  
 WHERE SUB.STUDENT\_ID =  
 """ + choiceItem + ";";  
  
 ResultSet resultSet = c.stmt.executeQuery(query);  
  
 ResultSetMetaData metaData = resultSet.getMetaData();  
 int totalColumns = metaData.getColumnCount();  
  
 tableAll.setModel(DbUtils.*resultSetToTableModel*(resultSet));  
 tableAll.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*);  
  
 // Set column size  
 TableColumnModel colModel = tableAll.getColumnModel();  
 for(int i=0; i<totalColumns; i++){  
 TableColumn tc = colModel.getColumn(i);  
 tc.setPreferredWidth(130);  
 }  
 }  
 else{  
 String query = """  
 SELECT DISTINCT S.FIRST\_NAME, S.LAST\_NAME, C.COURSE\_NAME, SUB.\*, G.\*  
 FROM  
 SUBJECT SUB  
 LEFT JOIN GRADES G  
 ON SUB.STUDENT\_ID = G.STUDENT\_ID and SUB.COURSE\_ID = G.COURSE\_ID and SUB.SEMESTER = G.SEMESTER  
 LEFT JOIN STUDENT S  
 ON S.IMMA\_ID = SUB.STUDENT\_ID  
 LEFT JOIN ENROLLMENT E  
 ON S.IMMA\_ID = E.IMMA\_ID  
 LEFT JOIN COURSE C  
 ON E.COURSE\_ID = C.COURSE\_ID  
 """;  
  
 ResultSet resultSet = c.stmt.executeQuery(query);  
  
 ResultSetMetaData metaData = resultSet.getMetaData();  
 int totalColumns = metaData.getColumnCount();  
  
 tableAll.setModel(DbUtils.*resultSetToTableModel*(resultSet));  
 tableAll.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*);  
  
 // Set column size  
 TableColumnModel colModel = tableAll.getColumnModel();  
 for(int i=0; i<totalColumns; i++){  
 TableColumn tc = colModel.getColumn(i);  
 tc.setPreferredWidth(130);  
 }  
 }  
 }catch (Exception E){  
 E.printStackTrace();  
 }  
 }  
 else if (ae.getSource() == btnPrint){  
 try {  
 tableAll.print();  
 }catch (Exception e) {  
 e.printStackTrace();  
 }  
 }else {  
 // Cancel Transaction  
 setVisible(false);  
 }  
 }  
  
 public static void main(String[] args) {  
 new ViewGrades();  
 }  
}

### 7.1.19 Stored Procedures with DDL and DML Statements

/\* 1. STORED PROCEDURE \*/  
  
/\* DDL STATEMENTS \*/  
  
DROP PROCEDURE IF EXISTS UMS\_Schema\_Creation;  
  
DELIMITER $$  
  
CREATE PROCEDURE UMS\_Schema\_Creation()  
BEGIN  
  
/\* DDL STATEMENTS \*/  
  
/\* 1. DROP TABLES \*/  
  
/\*1.1\*/  
DROP TABLE IF EXISTS GRADES;  
DROP TABLE IF EXISTS SUBJECT;  
  
/\*1.2\*/  
DROP TABLE IF EXISTS FEES;  
DROP TABLE IF EXISTS ENROLLMENT;  
  
/\*1.3\*/  
DROP TABLE IF EXISTS COURSE;  
DROP table IF EXISTS DEPARTMENT;  
  
/\*1.4\*/  
DROP TABLE IF EXISTS INSTRUCTOR;  
DROP TABLE IF EXISTS STUDENT;  
  
/\*1.5\*/  
DROP TABLE IF EXISTS ADMINISTRATOR;  
  
/\* 2. SCHEMA CREATION \*/  
  
/\*2.1 ADMINISTRATOR TABLE \*/  
CREATE TABLE ADMINISTRATOR(  
 USER\_ID *VARCHAR*(16) PRIMARY KEY,  
 PSWD *VARCHAR*(16),  
 FIRST\_NAME *VARCHAR*(255),  
 LAST\_NAME *VARCHAR*(255),  
 CONTACT *VARCHAR*(17),  
 EMAIL *VARCHAR*(255) /\* Trigger to generate regex email -> firstname.lastname@hft-europa.com\*/  
);  
  
/\*2.2 INSTRUCTOR TABLE \*/  
CREATE TABLE INSTRUCTOR(  
 USER\_ID *VARCHAR*(16),  
 PSWD *VARCHAR*(16),  
 FIRST\_NAME *VARCHAR*(255),  
 LAST\_NAME *VARCHAR*(255),  
 CONTACT *VARCHAR*(15),  
 EMAIL *VARCHAR*(50), /\* Trigger to generate regex email -> firstname.lastname@hft-europa.com\*/  
 DOB *VARCHAR*(255),  
 HOME\_ADDRESS *VARCHAR*(255),  
 HFT\_CABIN *VARCHAR*(255),  
 EMP\_ID *VARCHAR*(16) PRIMARY KEY  
);  
  
/\*2.3 STUDENT TABLE \*/  
CREATE TABLE STUDENT(  
 USER\_ID *VARCHAR*(16),  
 PSWD *VARCHAR*(16),  
 FIRST\_NAME *VARCHAR*(255),  
 LAST\_NAME *VARCHAR*(255),  
 CONTACT *VARCHAR*(15),  
 EMAIL *VARCHAR*(50),  
 DOB *VARCHAR*(255),  
 HOME\_ADDRESS *VARCHAR*(255),  
 GRADE\_XII *VARCHAR*(16),  
 INTERNATIONAL\_STUDENT *VARCHAR*(16),  
 IMMA\_ID *VARCHAR*(16) PRIMARY KEY  
);  
  
/\*2.4 DEPARTMENT TABLE \*/  
CREATE TABLE DEPARTMENT(  
 DEPT\_ID *INT* PRIMARY KEY,  
 DEAN\_ID *VARCHAR*(16),  
 DEPT\_NAME *VARCHAR*(255),  
 FOREIGN KEY (DEAN\_ID) REFERENCES INSTRUCTOR(EMP\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);  
  
/\*2.5 COURSE TABLE \*/  
CREATE TABLE COURSE(  
 COURSE\_ID *INT* PRIMARY KEY,  
 DEPT\_ID *INT*,  
 INSTRUCTOR\_ID *VARCHAR*(16),  
 COURSE\_NAME *VARCHAR*(255),  
 COURSE\_CODE *VARCHAR*(10), /\* Master Software Technology -> mst \*/  
 CREDITS *INT*,  
 BUILDING\_NO *INT*,  
 ROOM\_NO *INT*,  
 FOREIGN KEY (DEPT\_ID) REFERENCES DEPARTMENT(DEPT\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE,  
 FOREIGN KEY (INSTRUCTOR\_ID) REFERENCES INSTRUCTOR(EMP\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);  
  
/\*2.6 FEES \*/  
CREATE TABLE FEES(  
 STUDENT\_ID *VARCHAR*(16),  
 TUITION\_FEES *VARCHAR*(16), /\* 1500 Euros for International Students \*/  
 SEMESTER\_FEES *VARCHAR*(16), /\* 200 Euros for all \*/  
 D\_TICKET *VARCHAR*(16), /\* Optional: 360 Euros for all \*/  
 RESEARCH\_VARIABLE *VARCHAR*(16), /\* Optional: Variable \*/  
 FEES\_PAID *VARCHAR*(3),  
 FOREIGN KEY (STUDENT\_ID) REFERENCES STUDENT(IMMA\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);  
  
/\*2.7 ENROLLMENT \*/  
CREATE TABLE ENROLLMENT(  
 IMMA\_ID *VARCHAR*(16),  
 DEPT\_ID *INT*,  
 COURSE\_ID *INT*,  
 IMMA\_DATE *VARCHAR*(255),  
 IMMA\_STATUS *VARCHAR*(100),  
 FOREIGN KEY (IMMA\_ID) REFERENCES STUDENT(IMMA\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE,  
 FOREIGN KEY (DEPT\_ID) REFERENCES DEPARTMENT(DEPT\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE,  
 FOREIGN KEY (COURSE\_ID) REFERENCES COURSE(COURSE\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);  
  
/\*2.8 SUBJECT TABLE \*/  
CREATE TABLE SUBJECT(  
 STUDENT\_ID *VARCHAR*(16),  
 COURSE\_ID *INT*,  
 SEMESTER *INT*,  
 SUBJECT1 *VARCHAR*(255),  
 SUBJECT2 *VARCHAR*(255),  
 SUBJECT3 *VARCHAR*(255),  
 SUBJECT4 *VARCHAR*(255),  
 SUBJECT5 *VARCHAR*(255),  
 ELECTIVE1 *VARCHAR*(255),  
 FOREIGN KEY (STUDENT\_ID) REFERENCES STUDENT(IMMA\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE,  
 FOREIGN KEY (COURSE\_ID) REFERENCES COURSE(COURSE\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);  
  
/\*2.9 GRADES TABLE \*/  
CREATE TABLE GRADES(  
 STUDENT\_ID *VARCHAR*(16),  
 COURSE\_ID *INT*,  
 SEMESTER *INT*,  
 GRADE1 *VARCHAR*(8),  
 GRADE2 *VARCHAR*(8),  
 GRADE3 *VARCHAR*(8),  
 GRADE4 *VARCHAR*(8),  
 GRADE5 *VARCHAR*(8),  
 GRADE6 *VARCHAR*(8),  
 FOREIGN KEY (STUDENT\_ID) REFERENCES STUDENT(IMMA\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE,  
 FOREIGN KEY (COURSE\_ID) REFERENCES COURSE(COURSE\_ID)  
 ON DELETE SET NULL ON UPDATE CASCADE  
);  
  
END;  
  
$$  
  
DELIMITER ;

/\* 3. STORED PROCEDURE \*/  
  
/\* DML STATEMENTS \*/  
  
DROP PROCEDURE IF EXISTS UML\_DML\_Statements;  
  
DELIMITER $$  
  
CREATE PROCEDURE UML\_DML\_Statements()  
BEGIN  
  
/\* DML STATEMENTS \*/  
  
/\*3.1 ADMINISTRATOR \*/  
  
INSERT INTO ADMINISTRATOR (FIRST\_NAME, LAST\_NAME, CONTACT) VALUES ('Ryan', 'Gosling', '+49 15511234567');  
INSERT INTO ADMINISTRATOR (FIRST\_NAME, LAST\_NAME, CONTACT) VALUES ('Patrick', 'Bateman', '+49 15512345671');  
INSERT INTO ADMINISTRATOR (FIRST\_NAME, LAST\_NAME, CONTACT) VALUES ('Lou', 'Bloom', '+49 15513456712');  
  
/\* 3.2 DEPARTMENT \*/  
  
INSERT INTO DEPARTMENT (DEPT\_ID, DEPT\_NAME, DEAN\_ID) VALUES (1,'Computer Science', NULL);  
INSERT INTO DEPARTMENT (DEPT\_ID, DEPT\_NAME, DEAN\_ID) VALUES (2,'Mathematics', NULL);  
INSERT INTO DEPARTMENT (DEPT\_ID, DEPT\_NAME, DEAN\_ID) VALUES (3,'Business', NULL);  
INSERT INTO DEPARTMENT (DEPT\_ID, DEPT\_NAME, DEAN\_ID) VALUES (4,'Building Physics', NULL);  
  
/\* 3.3 COURSE \*/  
  
/\* Computer Science \*/  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (11, 1, NULL, 'Bachelor of Computer Science', 'bcs', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (12, 1, NULL, 'Bachelor of Augmented Reality', 'bar', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (13, 1, NULL, "Bachelor's degree in digitalization and information management", 'bdi', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (14, 1, NULL, 'Bachelor Surveying and Geoinformatics', 'bsg', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (15, 1, NULL, 'Master Photogrammetry and Geoinformatics', 'mpg', 90, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (16, 1, NULL, 'Master Software Technology', 'mst', 90, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (17, 1, NULL, 'Master Digital Processes and Technologies', 'mdp', 90, NULL, NULL);  
  
/\* Mathematics \*/  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (21, 2, NULL, 'Bachelor Applied Mathematics and AI', 'bam', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (22, 2, NULL, 'Master Mathematics', 'mam', 90, NULL, NULL);  
  
/\* Business \*/  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (31, 3, NULL, 'Bachelor of Business Administration', 'bba', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (32, 3, NULL, 'Business Administration International Business course', 'bai', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (33, 3, NULL, 'Bachelor of Infrastructure Management', 'bim', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (34, 3, NULL, 'Bachelor of Business Informatics', 'bbi', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (35, 3, NULL, 'Master General Management', 'mgm', 90, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (36, 3, NULL, 'Master Environmentally Oriented Logistics', 'meo', 90, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (37, 3, NULL, "Master of Business Psychology", 'mdb', 90, NULL, NULL);  
  
/\* Building Physics \*/  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (41, 4, NULL, 'Bachelor of Building Physics', 'bbp', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (42, 4, NULL, 'Bachelor of Climate Engineering', 'bce', 210, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (43, 4, NULL, 'Master Building Physics', 'mbp', 90, NULL, NULL);  
INSERT INTO COURSE (COURSE\_ID, DEPT\_ID, INSTRUCTOR\_ID, COURSE\_NAME, COURSE\_CODE, CREDITS, BUILDING\_NO, ROOM\_NO) VALUES (44, 4, NULL, 'Master in Sustainable Energy Competence', 'mse', 90, NULL, NULL);  
  
END;  
  
$$  
  
DELIMITER ;

### 7.1.20 Triggers

/\* 1. ADMINISTRATOR \*/  
DROP TRIGGER IF EXISTS trigger\_admin;  
  
DELIMITER $$  
  
CREATE TRIGGER trigger\_admin  
BEFORE INSERT ON ADMINISTRATOR  
FOR EACH ROW  
BEGIN  
SET NEW.PSWD = CONCAT(LEFT(UUID(), 8),LEFT(UUID(), 8));  
SET NEW.EMAIL = CONCAT(NEW.FIRST\_NAME, '.', NEW.LAST\_NAME, '@hft-europa.de');  
SET NEW.USER\_ID = CONCAT(RIGHT(YEAR(CURRENT\_DATE()), 1),IF(MONTH(CURRENT\_DATE()) <= 6, '1', '2'),LEFT(NEW.LAST\_NAME, 2),LEFT(NEW.FIRST\_NAME, 2));  
END;  
$$  
  
DELIMITER ;

/\* 2. INSTRUCTOR \*/  
DROP TRIGGER IF EXISTS trigger\_instructor;  
  
DELIMITER $$  
  
CREATE TRIGGER trigger\_instructor  
BEFORE INSERT ON INSTRUCTOR  
FOR EACH ROW  
BEGIN  
SET NEW.EMAIL = CONCAT(NEW.FIRST\_NAME, '.', NEW.LAST\_NAME, '@hft-europa.de');  
SET NEW.USER\_ID = CONCAT(RIGHT(YEAR(CURRENT\_DATE()), 1),IF(MONTH(CURRENT\_DATE()) <= 6, '1', '2'),LEFT(NEW.LAST\_NAME, 2),LEFT(NEW.FIRST\_NAME, 2));  
SET NEW.PSWD = CONCAT(LEFT(UUID(), 8),LEFT(UUID(), 8));  
END;  
$$  
  
DELIMITER ;

/\* 3. STUDENT \*/  
DROP TRIGGER IF EXISTS trigger\_student;  
  
DELIMITER $$  
  
CREATE TRIGGER trigger\_student  
BEFORE INSERT ON STUDENT  
FOR EACH ROW  
BEGIN  
SET NEW.EMAIL = CONCAT(NEW.FIRST\_NAME, '.', NEW.LAST\_NAME, '@hft-europa.de');  
SET NEW.USER\_ID = CONCAT(RIGHT(YEAR(CURRENT\_DATE()), 1),IF(MONTH(CURRENT\_DATE()) <= 6, '1', '2'),LEFT(NEW.LAST\_NAME, 2),LEFT(NEW.FIRST\_NAME, 2));  
SET NEW.PSWD = CONCAT(LEFT(UUID(), 8),LEFT(UUID(), 8));  
END;  
$$  
  
DELIMITER ;

/\* 4. FEES \*/  
DROP TRIGGER IF EXISTS trigger\_fees\_insert;  
  
DELIMITER $$  
  
CREATE TRIGGER trigger\_fees\_insert  
AFTER INSERT ON STUDENT  
FOR EACH ROW  
BEGIN  
INSERT INTO FEES (STUDENT\_ID, TUITION\_FEES, SEMESTER\_FEES, D\_TICKET, RESEARCH\_VARIABLE, FEES\_PAID)  
VALUES (NEW.IMMA\_ID, CASE WHEN UPPER(NEW.INTERNATIONAL\_STUDENT) = 'YES' THEN 1500 ELSE 0 END, 200, 0, 0, 'NO');  
END;  
$$  
  
DELIMITER ;

## **7.2 Other sources of inspiration for the project**

Our primary inspiration was derived from the official HFT Stuttgart website, which we found to be exceptionally simple, concise, and one of the best Hochschule websites we have encountered. Additionally, we were impressed by the quicklinks portal, which facilitated easy access to other platforms such as LSF and Moodle.

We have identified several areas for future development, including Row Level Security and the capability to assign deans automatically on a round-robin basis, among other enhancements.

Our initial objective was to fulfill all the requirements specified in the Moodle document. Moving forward, our next goal is to begin work on these future proposals. Prior to moving on this next phase, we would greatly appreciate feedback from the Professors. Such feedback would not only delight us but also significantly aid in refining our approach.