

DATABASE MANAGEMENT SYSTEM FOR UNIVERSITY (UDBMS)

Prepared by
Ahmed Hatem



Database Management System for a University

Contents

Overview	4
I. Database Design	5
1. Requirements	5
Student Table	5
Department Table	5
Course Table	5
Student-Course Table	5
Grade Table	5
Evaluation Table	6
2. ER-Design	6
3. Mapping & Normalization	6
II. SQL Implementation	7
1. SQL Script for Building Database Schema	7
2. Database Population with Sample Data	7
3. Testing and Validation of Database Correctness:	9
III. PLSQL Implementation	10
1. PL/SQL Procedure: Update Evaluation and CGPA	10
2. PL/SQL Function: Calculate Course Average GPA	10
3. PL/SQL Trigger: Delete Course Trigger	10
4. PL/SQL Trigger: Delete Department Trigger	11
5. PL/SQL Trigger: Delete Student Data Trigger	11



6.	PL/SQL Trigger: Enroll Into Evaluation Trigger	12
7.	PL/SQL Trigger: Unenroll Student Trigger	12
8.	PL/SQL Trigger: Enroll Student Name Trigger	12
9.	PL/SQL Trigger: Insert Course Name Trigger	13
10.	PL/SQL Trigger: Generate Evaluation Date Trigger	13
IV.	Bash Scripting	14
1.	Backup Script Documentation	14
2.	Disk Space Monitoring	16
V.	Java Application Development	18
	Application Architecture	18
1.	Database Connection using Oracle 11g:	19
2.	Data Transfer Objects (DTOs):	19
3.	The University Package (The main package):	21
3.1.	JavaFX Application: University	21
3.2.	HomeController Class:	22
3.3.	StudentsController Class:	23
3.4.	CoursesController Class:	24
3.5.	EvaluationController Class:	25
3.6.	ReportsController Class:	26
	Application Scenes	27
1.	Home Scene:	27
2.	Students Scene:	28
3.	Course Scene:	29
4.	Evaluation Scene:	30



5. Reports Scene: _____	31
Validation and Anomalies Checks _____	32
1. Handling Empty Fields: _____	32
2. Handling E-mail Format: _____	32
3. Handling Primary Keys Repetition: _____	32
4. Handling Invalid Department ID: _____	32
5. Handling Deleting Department with Enrolled Students: _____	33
6. Handling Deleting Course with Enrolled Students: _____	33
7. Handling Adding Existing Course ID: _____	33
8. Handling Adding Course to Non-Existing Department: _____	33
9. Handling Adding Department with Same ID: _____	34
10. Handling Enrolling Student into a Course not in his Department: _____	34
11. Handling Invalid Grades: _____	34



Overview

The **University Database Management System (UDMS)** is a user-friendly solution designed to streamline administrative tasks in educational institutions. With its intuitive GUI, administrators can efficiently manage student, course, and departmental data, facilitating tasks such as adding, deleting, and updating records. UDMS also generates comprehensive reports for informed decision-making and incorporates bash scripts for automated database backups and disk space monitoring, enhancing data security and system reliability. Overall, UDMS enhances administrative efficiency, improves decision-making, and ensures data integrity within university settings.



I. Database Design

1. Requirements

Student Table

- It stores information about students, like their name, GPA (grade average), hometown, email, gender, and birthdate.
- Each student has a unique ID number.
- We also keep track of which department the student belongs to.

Department Table

- This table holds details about different departments within the university.
- It includes the department name, the department head, and contact information.
- We also count how many instructors and students are in each department.

Course Table

- Here, we list all the courses offered by the university.
- Each course has a name, and a unique ID number, a credit hour and a maximum number of students who can enroll.
- We link each course to its corresponding department.

Student-Course Table

- This table keeps track of which students are taking which courses.
- It notes the semester when the student took the course.

Grade Table

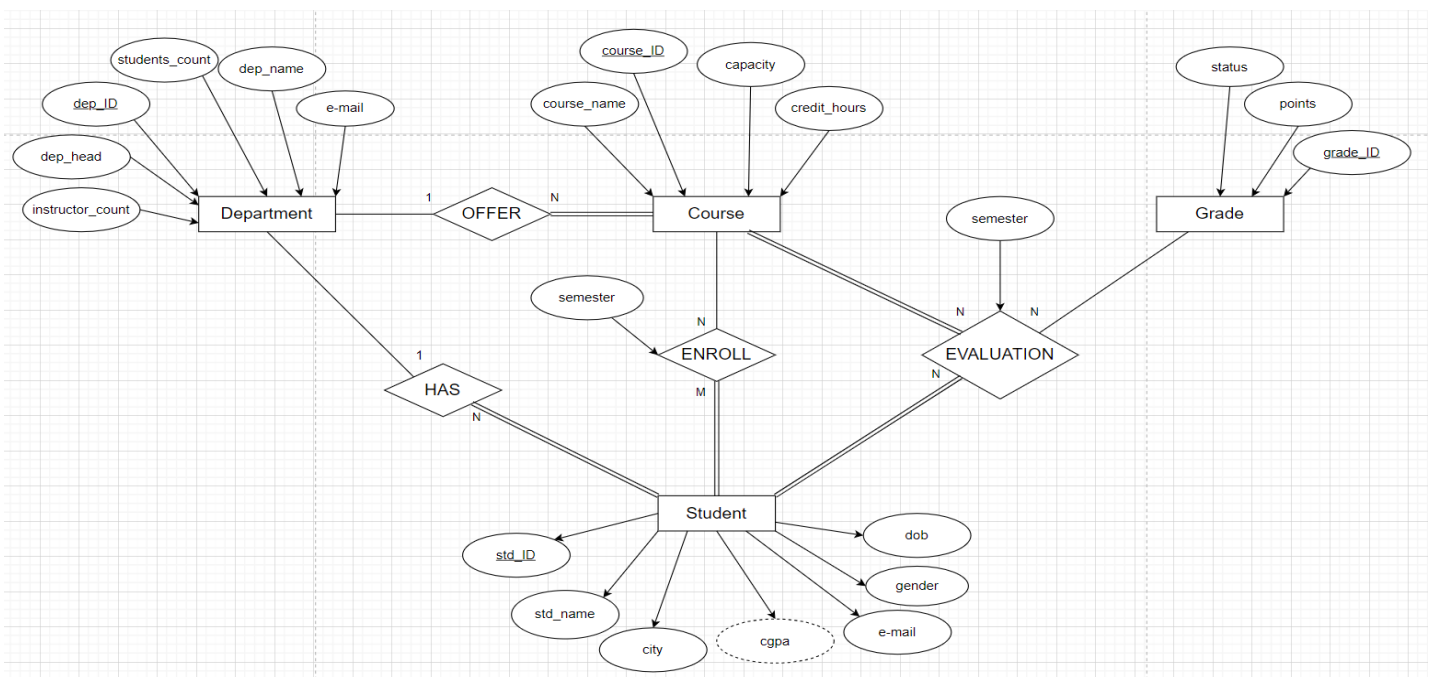
- We use this table to record the standard grades points.
- Each grade has a unique identifier, like an ID number.
- We also store whether the grade indicates a passing or failing status.



Evaluation Table

- Here, we store evaluations of student performance in courses.
- We record details like the student's ID and name, the course ID and name, the grade received, the date of evaluation, and the semester.

2. ER-Design



3. Mapping & Normalization

The tables below are already in the **3NF**

Student (std_ID, std_name, cgpa, **dep_id(f.k)**, city, e-mail, gender, dob)

Department (dep_ID, dep_name, dep_head, e-mail, instructor_count, std_count)

Course (course_id, course_name, credit_hours, capacity, **dep_ID(f.k)**)

Std_Course (**std_ID(f.k)**, **course_ID(f.k)**, semester, std_name, course_name)

Grade (grade_ID, points, status)

Evaluation (**std_ID(f.k)**, **course_ID(f.k)**, semester, std_name, course_name, **grade_ID(f.k)**, credit_hours, evaluation_date)



II. SQL Implementation

1. SQL Script for Building Database Schema

- I have created SQL scripts to define and build the database schema for the University Database Management System (UDMS).
- These scripts include the necessary commands to create tables such as Student, Department, Course, Std_Course, Grade, and Evaluation, along with their respective attributes and relationships.

2. Database Population with Sample Data

- Additionally, I have populated the database with sample data from excel sheets to simulate real-world scenarios.
- This includes inserting mock student records, course information, department details, grades, and evaluations into the respective tables.

- Students

std_ID	std_name	dep_id	cgpa	city	email	gender	dob
1	John Doe	1		New York	john.doe@example.com	M	1998-05-15
2	Jane Smith	2		Los Angeles	jane.smith@example.com	F	1997-08-22
3	Bob Johnson	1		Chicago	bob.johnson@example.com	M	1999-01-10
4	Alice Williams	2		San Francisco	alice.williams@example.com	F	1996-11-30
5	Charlie Brown	1		Houston	charlie.brown@example.com	M	1998-03-05
6	Eva Martinez	2		Miami	eva.martinez@example.com	F	1997-06-18
7	David Lee	1		Seattle	david.lee@example.com	M	1999-09-25
8	Sophia Chen	2		Boston	sophia.chen@example.com	F	1996-12-12
9	Michael Nguyen	1		Dallas	michael.nguyen@example.com	M	1998-02-03
10	Olivia Garcia	2		Atlanta	olivia.garcia@example.com	F	1997-04-14
11	William Wang	1		Phoenix	william.wang@example.com	M	1999-07-07
12	Emma Davis	2		Denver	emma.davis@example.com	F	1996-10-20



- Courses

course_id	course_name	credit_hours	capacity	dep_id
11	SQL	3	10	1
12	Python	6	15	1
13	Data Warehouse	3	10	1
14	Big Data Tools	6	15	1
21	Cyber Security	6	15	2
22	Networking	6	15	2
23	Operating Systems	3	10	2
24	Ethical Hacking	3	10	2

- Departments

Dep_ID	Dep_name	Dep_head	e-mail	Instructor_count	Students_count
1	Data Engineering	Ahmed Hatem	dataengineering@gmail.com	5	40
2	System Administration	Ahmed Hatem	cybersecuirty@gmail.com	3	30

- Student Course

Std_ID	Std_name	Course_ID	Course_name	Dep_ID
1	John Doe	11	SQL	1
1	John Doe	12	Python	1
1	John Doe	13	Data Warehouse	2
1	John Doe	14	Big Data Tools	2
2	Jane Smith	21	Cyber Security	1
2	Jane Smith	22	Networking	1
2	Jane Smith	23	Operating Systems	2
2	Jane Smith	24	Ethical Hacking	2

- Grades

Grade_ID	Points	Grade_status
A	4	Excellent
A-	3.666	Excellent
B+	3.333	Very Good
B	3	Very Good
B-	2.666	Good
C+	2.333	Good
C	2	Fair
C-	1.666	Fair
D+	1.333	Weak
D	1	Weak
F	0	Fail



- Evaluations

std_ID	std_name	course_ID	course_name	grade_ID	credit_hours	evaluation_date	semester
1	John Doe	11	SQL	B	3	2023-01-20	1
1	John Doe	12	Python	C-	6	2023-01-25	1
1	John Doe	13	Data Warehouse	B+	3	2023-06-15	2
1	John Doe	14	Big Data Tools	A	6	2023-06-30	2
2	Jane Smith	21	Cyber Security	A-	6	2023-01-05	1
2	Jane Smith	22	Networking	C	6	2023-01-20	1
2	Jane Smith	23	Operating Systems	B	3	2023-06-20	2
2	Jane Smith	24	Ethical Hacking	B+	3	2023-06-05	2

- By doing so, the database now contains representative data that can be used for testing and demonstration purposes.

3. Testing and Validation of Database Correctness:

- To ensure the accuracy and functionality of the database, thorough testing and validation have been conducted.
- This involved running queries and performing operations to verify data integrity, enforce referential constraints, and validate the overall behavior of the database system.
- Through systematic testing procedures, the correctness of the database has been validated, meeting the requirements and expectations outlined in the project specifications.



III. PLSQL Implementation

1. PL/SQL Procedure: Update Evaluation and CGPA

This procedure updates a student's grade for a specific course and semester, recalculating their Cumulative Grade Point Average (CGPA) accordingly.

- **Update Grade:** It updates the grade for a student in a given course and semester.
- **Calculate Total Credits and Grade Points:** It calculates the total credits and grade points earned by the student across all courses.
- **Calculate New CGPA:** It computes the new CGPA for the student based on the updated information.
- **Update CGPA:** Finally, it updates the student's CGPA in the database.

This procedure ensures that the student's academic records remain accurate and up to date with their latest grades and CGPA.

2. PL/SQL Function: Calculate Course Average GPA

This function computes the average GPA for a specified course in the university database system.

- **Calculate Average GPA:** It calculates the average GPA for the given course by querying the **evaluation** table to obtain the grades and then averaging those grades using the associated grade points.
- **Return Average GPA:** The function returns the calculated average GPA for the specified course.

This function facilitates the quick retrieval of average GPA information for a particular course, aiding in academic assessment and analysis.

3. PL/SQL Trigger: Delete Course Trigger

This trigger is designed to handle the deletion of a course from the university database system.

- **Delete Related Data:** Before the deletion of a course (**BEFORE DELETE ON course**), this trigger removes related data from two tables:
 - **Std_Course:** Deletes entries in the **Std_Course** table where the course ID matches the ID of the course being deleted.



- **Evaluation:** Deletes entries in the **Evaluation** table where the course ID matches the ID of the course being deleted.
- **Automatic Cascade Deletion:** Foreign key constraints are set up to automatically delete related data from other tables. Therefore, this trigger ensures that any associated data is properly cleaned up when a course is deleted from the database.

4. PL/SQL Trigger: Delete Department Trigger

This trigger facilitates the deletion process of a department from the university database.

- **Delete Related Records:** Executed before the deletion of a department (**BEFORE DELETE ON department**), this trigger removes related records from two tables:
 - **std_course:** Deletes entries where the department ID matches the ID of the department being deleted.
 - **evaluation:** Deletes entries where the department ID matches the ID of the department being deleted.

By handling the deletion of associated records, this trigger ensures that data integrity is maintained when a department is removed from the system.

5. PL/SQL Trigger: Delete Student Data Trigger

This trigger handles the deletion of student data from the university database.

- **Delete Related Data:** Triggered before the deletion of a student (**BEFORE DELETE ON Student**), this trigger removes related data from two tables:
 - **Std_Course:** Deletes entries where the student ID matches the ID of the student being deleted.
 - **Evaluation:** Deletes entries where the student ID matches the ID of the student being deleted.
- **Automatic Cascade Deletion:** Foreign key constraints are in place to automatically delete related data from other tables, ensuring that associated records are properly handled when a student is removed from the system.

This trigger contributes to maintaining data consistency and integrity by handling the cascading deletion of associated records when a student is deleted from the database.



6. PL/SQL Trigger: Enroll Into Evaluation Trigger

This trigger automates the process of enrolling a student into the evaluation system when they are enrolled in a course.

- **Enroll Into Evaluation:** Triggered after a new enrollment entry is inserted into the **STD_COURSE** table (**AFTER INSERT ON STD_COURSE**), this trigger performs the following actions:
 - Retrieves the credit hours associated with the course being enrolled.
 - Inserts a new record into the **evaluation** table, including student ID, student name, course ID, course name, null grade ID (initially), credit hours, and semester information.
- **Exception Handling:** It includes exception handling to capture and output any errors that may occur during the process. If an error occurs, the transaction is rolled back to maintain data consistency.

This trigger streamlines the enrollment process by automatically adding students to the evaluation system when they are enrolled in a course.

7. PL/SQL Trigger: Unenroll Student Trigger

This trigger manages the unenrollment of a student from a course in the university database.

- **Unenroll Student:** Triggered before the deletion of an evaluation entry (**BEFORE DELETE ON evaluation**), this trigger removes the corresponding enrollment entry from the **std_course** table.
 - It deletes the record where the student ID, course ID, and semester match the values of the evaluation entry being deleted.

This trigger ensures that when a student is unenrolled from a course, their enrollment record is properly removed from the system to maintain accurate and up-to-date data.

8. PL/SQL Trigger: Enroll Student Name Trigger

This trigger automatically assigns the student's name to the **STD_NAME** column when a new enrollment entry is inserted into the **STD_COURSE** table.

- **Retrieve Student Name:** Before insertion (**BEFORE INSERT ON CASESTUDY.STD_COURSE**), it retrieves the student's name from the **student** table based on their student ID (**STD_ID**).



- **Assign Student Name:** It assigns the retrieved student's name to the **STD_NAME** column in the **std_course** table for the new enrollment entry.

This trigger ensures that the student's name is accurately associated with their enrollment record in the database.

9. PL/SQL Trigger: Insert Course Name Trigger

This trigger populates the **COURSE_NAME** column in the **std_course** table with the corresponding course name before insertion of a new enrollment entry.

- **Retrieve Course Name:** Before insertion (**BEFORE INSERT ON std_course**), it retrieves the course name from the **course** table based on the course ID (**COURSE_ID**).
- **Assign Course Name:** It assigns the retrieved course name to the **COURSE_NAME** column for the new enrollment entry.

This trigger helps maintain consistency by ensuring that the course name is included in the enrollment record.

10. PL/SQL Trigger: Generate Evaluation Date Trigger

This trigger generates a random evaluation date based on the semester before inserting a new entry into the **evaluation** table.

- **Generate Evaluation Date:** Before insertion (**BEFORE INSERT ON evaluation**), it generates a random evaluation date based on the semester:
 - For Semester 1, it generates a random date in January.
 - For Semester 2, it generates a random date in June.
- **Assign Evaluation Date:** It assigns the generated evaluation date to the **EVALUATION_DATE** column for the new entry.

This trigger introduces randomness to the evaluation dates while ensuring that they are within the appropriate timeframe for each semester.



IV. Bash Scripting

1. Backup Script Documentation

This script is designed to perform automated backups of an Oracle database using Oracle Data Pump. It exports data from the specified database and saves it to a backup file in a designated directory. Additionally, it logs the outcome of the backup operation for tracking and monitoring purposes.

Script Overview:

1. Set Database Connection Information:

- The script defines variables to store the username (**db_owner**), password (**pass**), and database instance (**db**) required for database connection.

2. Define Backup Directory:

- A variable named **direct** is created to store the directory path where backup files will be saved.

3. Generate Backup File Name:

- The **date** command is utilized to generate a timestamp (**d**) in the format **YYYYMMDD_HHMMSS**.
- A backup file name (**file_name**) is constructed by combining the prefix "**backup_**", the timestamp (**d**), and the file extension **.dmp**.

4. Perform Database Export:

- The script executes the **expdp** command to export data from the database.
- Database connection details (**\${db_owner}/\${pass}@\${db}**), backup file name (**DUMPFILE=\${file_name}**), and backup directory (**DIRECTORY=BACKUP_CASESTUDY**) are provided as parameters.



5. Check Export Status:

- The exit status of the **expdp** command is checked using **\$?**.
- If the exit status is **0**, indicating success, a success message is appended to the backup log file.
- If the exit status is non-zero, indicating failure, an error message is appended to the backup log file.

6. Log Backup Operation:

- A message indicating whether the backup operation succeeded or failed is appended to the backup log file **\$(direct)/backupLogs.log**.

```
export.log - Notepad
File Edit Format View Help
;;;
Export: Release 11.2.0.2.0 - Production on Sat Feb 10 20:09:34 2024

Copyright (c) 1982, 2009, Oracle and/or its affiliates. All rights reserved.
;;;
Connected to: Oracle Database 11g Express Edition Release 11.2.0.2.0 - Production
Starting "CASESTUDY"."SYS_EXPORT_SCHEMA_01": CASESTUDY/*****@XE DUMPFILE=backup_20240210_200933.dmp DIRECTORY=BACKUP_CASESTUDY
Estimate in progress using BLOCKS method...
Processing object type SCHEMA_EXPORT/TABLE/TABLE_DATA
Total estimation using BLOCKS method: 384 KB
Processing object type SCHEMA_EXPORT/USER
Processing object type SCHEMA_EXPORT/SYSTEM_GRANT
Processing object type SCHEMA_EXPORT/ROLE_GRANT
Processing object type SCHEMA_EXPORT/DEFAULT_ROLE
Processing object type SCHEMA_EXPORT/PRE_SCHEMA/PROCACT_SCHEMA
Processing object type SCHEMA_EXPORT/TABLE/TABLE
Processing object type SCHEMA_EXPORT/TABLE/INDEX/INDEX
Processing object type SCHEMA_EXPORT/TABLE/CONSTRAINT/CONSTRAINT
Processing object type SCHEMA_EXPORT/TABLE/INDEX/STATISTICS/INDEX_STATISTICS
Processing object type SCHEMA_EXPORT/TABLE/COMMENT
Processing object type SCHEMA_EXPORT/FUNCTION/FUNCTION
Processing object type SCHEMA_EXPORT/PROCEDURE/PROCEDURE
Processing object type SCHEMA_EXPORT/FUNCTION/ALTER_FUNCTION
Processing object type SCHEMA_EXPORT/PROCEDURE/ALTER_PROCEDURE
Processing object type SCHEMA_EXPORT/TABLE/CONSTRAINT/REF_CONSTRAINT
Processing object type SCHEMA_EXPORT/TABLE/TRIGGER
Processing object type SCHEMA_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
. . exported "CASESTUDY"."COURSE" 6.835 KB 8 rows
. . exported "CASESTUDY"."DEPARTMENT" 7.156 KB 2 rows
. . exported "CASESTUDY"."EVALUATION" 10.00 KB 45 rows
. . exported "CASESTUDY"."GRADE" 6.046 KB 11 rows
. . exported "CASESTUDY"."STD_COURSE" 8.257 KB 45 rows
. . exported "CASESTUDY"."STUDENT" 8.679 KB 13 rows
Master table "CASESTUDY"."SYS_EXPORT_SCHEMA_01" successfully loaded/unloaded
*****
Dump file set for CASESTUDY.SYS_EXPORT_SCHEMA_01 is:
C:\ORACLE\APP\ORACLE\BACKUP_CASESTUDY\BACKUP_20240210_200933.DMP
Job "CASESTUDY"."SYS_EXPORT_SCHEMA_01" successfully completed at 20:09:38
```

```
backupLogs.log - Notepad
File Edit Format View Help
| Make Backup Successfully
```




2. Disk Space Monitoring

This Bash script monitors disk space usage on a Windows system. It leverages the Windows Management Instrumentation Command-line (WMIC) to gather disk space information from the C: drive. If the disk space usage exceeds a specified threshold, the script writes an alert message to a log file.

1. Script Components:

1.1. Threshold Setting:

- The **THRESHOLD** variable allows users to define the threshold value for disk space usage, expressed as a percentage. For instance, setting **THRESHOLD=10** indicates that an alert should be triggered if disk space usage exceeds 10%.

1.2. Log File Path Definition:

- The **LOG_FILE** variable specifies the path to the log file where alert messages will be written. Users can customize this path to their preferred location.

1.3. Disk Space Checking:

- The script uses WMIC to execute commands that retrieve disk space information. Specifically, it obtains the amount of free space and total size of the C: drive. This information is then used to calculate the percentage of disk space used.

1.4. Alert Triggering:

- If the calculated disk space usage exceeds the threshold value, the script writes an alert message to the log file. This message contains the current date and time, along with details about the disk space usage percentage.



2. Script Usage:

2.1. Download:

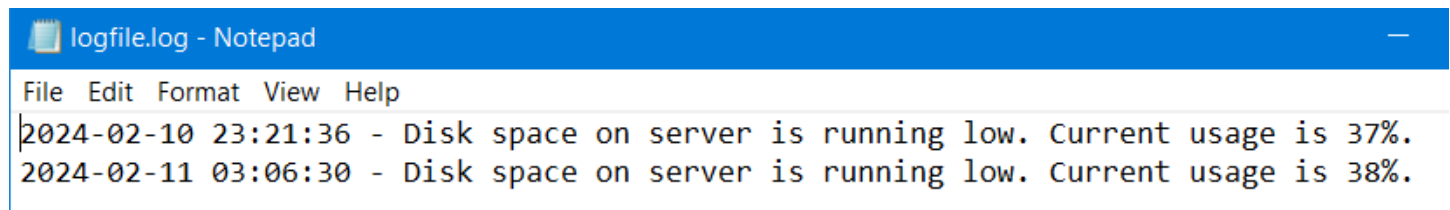
- Users can download the script file to their Windows system.

2.2. Execute:

- The script can be executed by running it from the command line or by double-clicking on it in a file explorer window.

2.3. Check Log File:

- Upon execution, if the script detects low disk space, it writes an alert message to the log file specified by **LOG_FILE**. Users can review this log file to track disk space alerts over time.

A screenshot of a Windows Notepad application window titled 'logfile.log - Notepad'. The window has a blue title bar and a menu bar with 'File', 'Edit', 'Format', 'View', and 'Help'. The text area contains two lines of log entries in a monospaced font: '2024-02-10 23:21:36 - Disk space on server is running low. Current usage is 37%.' and '2024-02-11 03:06:30 - Disk space on server is running low. Current usage is 38%.'

```
logfile.log - Notepad
File Edit Format View Help
2024-02-10 23:21:36 - Disk space on server is running low. Current usage is 37%.
2024-02-11 03:06:30 - Disk space on server is running low. Current usage is 38%.
```

3. Requirements:

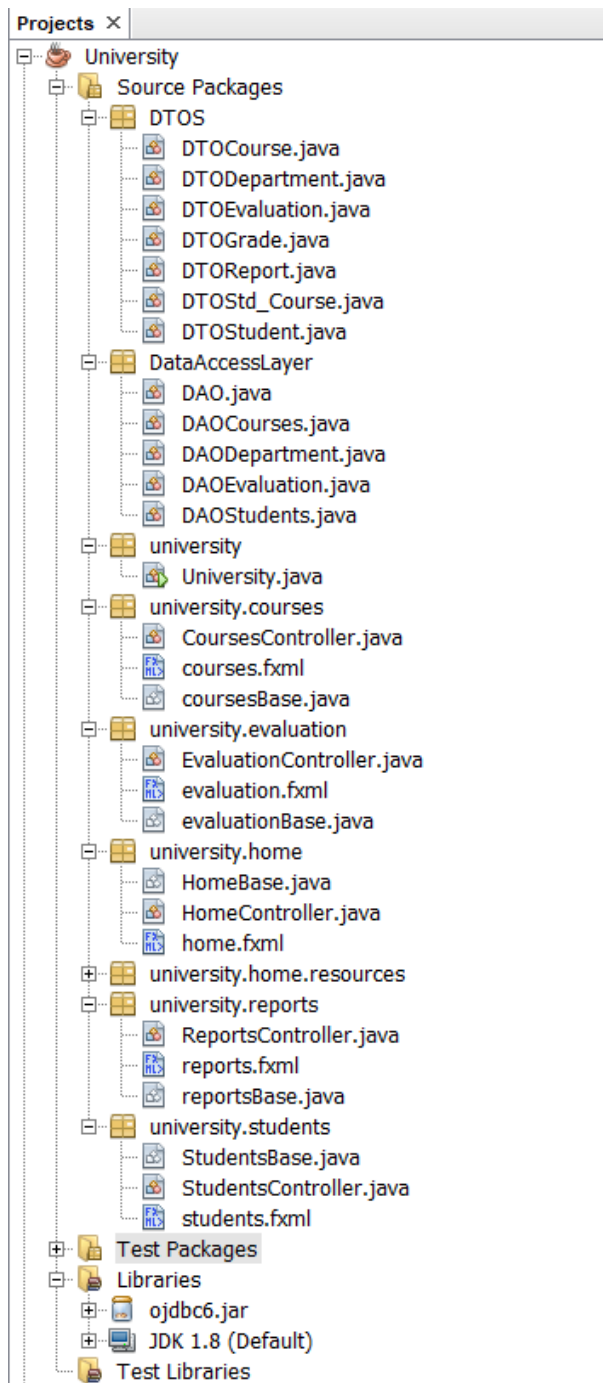
- This script is designed for use on a Windows system.
- It relies on the presence of WMIC for retrieving disk space information.
- Users must have appropriate permissions to execute the script and access disk space data.



V. Java Application Development

In this phase of the project, a Java application was developed to interact with the university database system.

Application Architecture



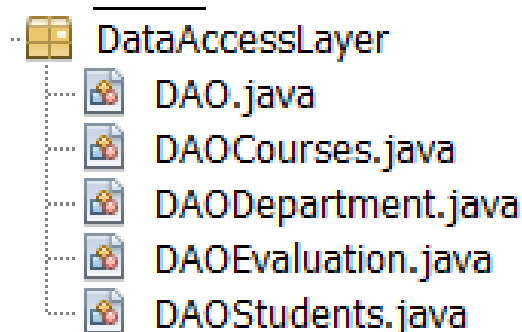


1. Database Connection using Oracle 11g:

DAO Class (Data Access Object):

Within the DataAccessLayer package, we created a class called DAO.

- **Database Connection Establishment:** The DAO class includes a method to establish a connection to our Oracle database. We utilized the Oracle JDBC driver version 6 for this purpose.
- **Error Handling:** Any errors encountered during the connection process are logged for troubleshooting purposes.



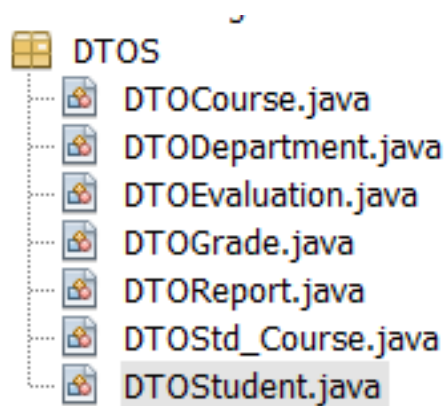
2. Data Transfer Objects (DTOs):

The DTOs classes serves as a structured representation of our tables within our university database system. Let's take the **DTOStudent** as an example Here's a breakdown of its architecture:

- **Attributes:** It encapsulates various attributes to store essential information about a student, including:
 - Student ID (**std_id**)
 - Student Name (**std_name**)
 - Department ID (**dep_id**)
 - Cumulative Grade Point Average (CGPA) (**cgpa**)
 - City (**city**)
 - Email (**email**)
 - Gender (**gender**)
 - Date of Birth (**dob**)

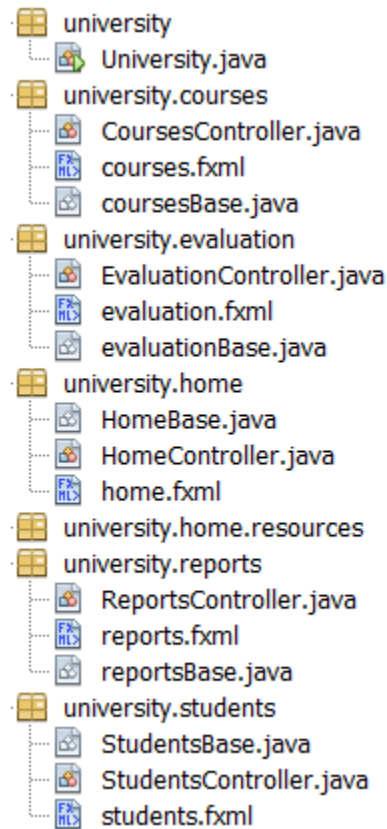


- **Constructors:** Two constructors are provided:
 - One constructor initializes all attributes, including CGPA and city.
 - Another constructor is for the initial entry of students, requiring only essential information such as student ID, name, department ID, email, gender, and date of birth.
- **Getter and Setter Methods:** These methods facilitate access to the private attributes of the DTOStudent object, allowing retrieval and modification of student information.
- **Date Formatting Method:** The **getFormattedDob()** method formats the date of birth into a string with the pattern "dd/MM/yyyy" for easy display.





3. The University Package (The main package):



3.1. JavaFX Application: University

This class serves as the entry point for our JavaFX application, facilitating the initialization of the graphical user interface (GUI) and loading the home screen.

- **Application Extension:** The **University** class extends the **Application** class provided by JavaFX, enabling us to create a JavaFX application.
- **start Method:** This method is overridden from the **Application** class and is automatically called when the application is launched. It sets up the primary stage (window) by loading the FXML file for the home screen (**home.fxml**) using **FXMLLoader**, creating a scene with the loaded root node, and displaying the stage.
- **main Method:** The **main** method is the entry point for the Java application. It launches the JavaFX application by calling the **launch** method and passing any command-line arguments.



3.2. HomeController Class:

The HomeController class is a controller for the home screen of our university application. It manages the user interface elements and handles user interactions.

- **FXML Controller:** This class is annotated with **@FXML**, indicating that it is an FXML controller responsible for managing the graphical user interface defined in the corresponding FXML file.
- **FXML Elements:** The class contains various FXML elements representing buttons, image views, and anchor panes. These elements are injected into the controller using the **@FXML** annotation and are used to interact with the user interface.
- **Initialization Method:** The **initialize** method, overridden from the **Initializable** interface, is called automatically when the controller is initialized. Currently, it does not contain any specific initialization logic.
- **Action Event Methods:** The class includes several methods annotated with **@FXML** corresponding to different user actions, such as clicking on buttons.
 - These methods handle user actions by loading different FXML files corresponding to different screens (e.g., students, evaluation, reports, courses) and setting them as the center content of the parent anchor pane.
- **Exception Handling:** Any exceptions that occur during the loading of FXML files are caught and printed to the console for debugging purposes.



3.3. StudentsController Class:

The StudentsController class manages the functionality of the student management screen in our university application. It handles user interactions, data validation, and database operations related to students.

- **FXML Controller:** This class is annotated with **@FXML**, indicating that it is an FXML controller responsible for managing the graphical user interface defined in the corresponding FXML file.
- **FXML Elements:** The class contains various FXML elements representing text fields, buttons, table views, and other user interface components. These elements are injected into the controller using the **@FXML** annotation and are used to interact with the user interface.
- **Initialization Method:** The **initialize** method, overridden from the **Initializable** interface, is called automatically when the controller is initialized. It populates the table view with student data retrieved from the database.
- **Action Event Methods:** The class includes methods annotated with **@FXML** corresponding to different user actions, such as adding, deleting, or clearing student data.
 - These methods handle user actions by performing data validation, executing database operations (e.g., adding or deleting students), and updating the user interface accordingly.
- **Exception Handling:** Exception handling is implemented to catch and handle various types of exceptions that may occur during database operations or user interactions. Error messages are displayed to the user via alert dialogs for feedback and troubleshooting.
- **Data Access Object (DAO):** The class interacts with a **DAOStudents** object to perform database operations related to students, such as adding, deleting, or retrieving student data from the database.
- **Validation Logic:** The class includes validation logic to ensure that user input is valid before performing database operations. For example, it checks for empty fields, valid email format, and existence of department IDs in the system.



3.4. CoursesController Class:

The CoursesController class manages the functionality of the courses management screen in our university application. It handles user interactions, data validation, and database operations related to courses and departments.

- **FXML Controller:** This class is annotated with **@FXML**, indicating that it is an FXML controller responsible for managing the graphical user interface defined in the corresponding FXML file.
- **FXML Elements:** The class contains various FXML elements representing text fields, buttons, table views, and other user interface components. These elements are injected into the controller using the **@FXML** annotation and are used to interact with the user interface.
- **Initialization Method:** The **initialize** method, overridden from the **Initializable** interface, is called automatically when the controller is initialized. It populates the table views with course and department data retrieved from the database.
- **Action Event Methods:** The class includes methods annotated with **@FXML** corresponding to different user actions, such as adding, deleting, or clearing course and department data.
 - These methods handle user actions by performing data validation, executing database operations (e.g., adding or deleting courses/departments), and updating the user interface accordingly.
- **Exception Handling:** Exception handling is implemented to catch and handle various types of exceptions that may occur during database operations or user interactions. Error messages are displayed to the user via alert dialogs for feedback and troubleshooting.
- **Data Access Objects (DAOs):** The class interacts with **DAOCourses** and **DAODepartment** objects to perform database operations related to courses and departments, such as adding, deleting, or retrieving data from the database.
- **Validation Logic:** The class includes validation logic to ensure that user input is valid before performing database operations. For example, it checks for empty fields, valid department IDs, and existing course IDs in the system.



3.5. EvaluationController Class:

The EvaluationController class manages the functionality of the evaluation screen in our university application. It handles user interactions, data validation, and database operations related to student evaluations and course enrollments.

- **FXML Controller:** This class is annotated with **@FXML**, indicating that it is an FXML controller responsible for managing the graphical user interface defined in the corresponding FXML file.
- **FXML Elements:** The class contains various FXML elements representing text fields, buttons, table views, and other user interface components. These elements are injected into the controller using the **@FXML** annotation and are used to interact with the user interface.
- **Initialization Method:** The **initialize** method, overridden from the **Initializable** interface, is called automatically when the controller is initialized. It populates the table view with evaluation data retrieved from the database.
- **Action Event Methods:** The class includes methods annotated with **@FXML** corresponding to different user actions, such as updating grades, enrolling students in courses, unenrolling students from courses, and clearing text fields.
 - These methods handle user actions by performing data validation, executing database operations, and updating the user interface accordingly.
- **Exception Handling:** Exception handling is implemented to catch and handle various types of exceptions that may occur during database operations or user interactions. Error messages are displayed to the user via alert dialogs for feedback and troubleshooting.
- **Data Access Object (DAO):** The class interacts with a **DAOEvaluation** object to perform database operations related to evaluations, such as updating grades, enrolling students, and unenrolling students.
- **Validation Logic:** The class includes validation logic to ensure that user input is valid before performing database operations. For example, it checks for valid grade values, numeric input for student ID, course ID, and semester, and verifies if the course belongs to the student's department before enrolling.



3.6. ReportsController Class:

The ReportsController class is responsible for managing the functionality related to generating reports in the university application. It handles user interactions and data retrieval from the database to populate the report table.

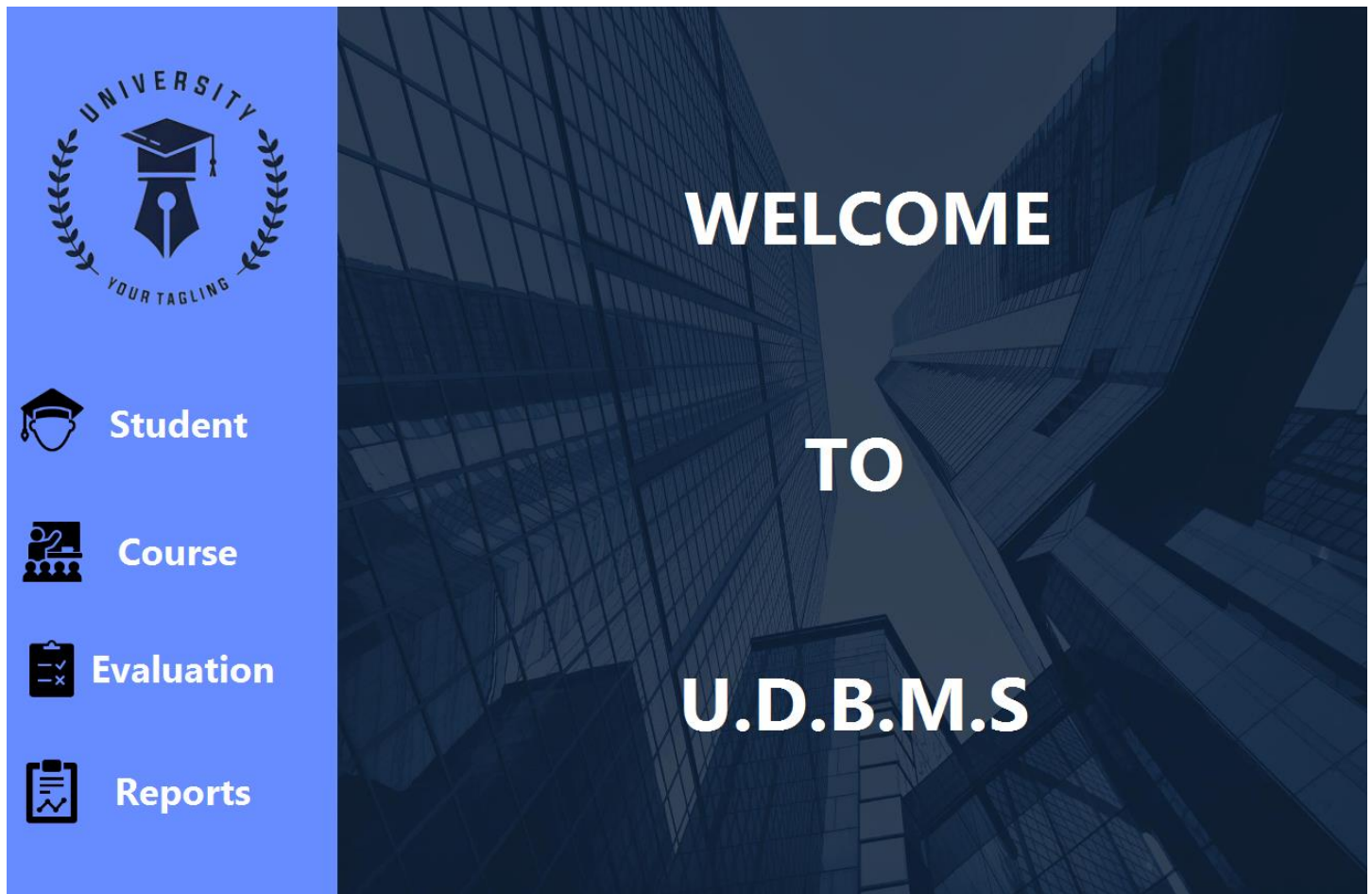
- **FXML Controller:** This class is annotated with **@FXML**, indicating that it is an FXML controller responsible for managing the graphical user interface defined in the corresponding FXML file.
- **FXML Elements:** The class contains various FXML elements representing a table view and its columns, as well as a button to trigger report generation. These elements are injected into the controller using the **@FXML** annotation and are used to interact with the user interface.
- **Initialization Method:** The **initialize** method, overridden from the **Initializable** interface, is currently empty. This method would typically be used to initialize the controller and set up initial state or data.
- **Generate Report Method:** The **generateReport** method is an event handler for the button that triggers report generation. When the button is clicked, this method retrieves report data from the **DAOEvaluation** object and populates the report table view with the fetched data.
 - It calls the **generateReport** method of the **DAOEvaluation** object to fetch report data.
 - The fetched report data is then used to populate the table view by setting cell value factories for each column.
- **Data Access Object (DAO):** The class interacts with a **DAOEvaluation** object to retrieve report data from the database. It calls the **generateReport** method of the **DAOEvaluation** object to fetch the required data.
- **Table View Configuration:** The table view columns are configured to display specific properties of the **DTOResult** objects. Each column is associated with a corresponding property of the **DTOResult** class using the **PropertyValueFactory** class.



Application Scenes

1. Home Scene:


- The Home scene serves as the landing page or main menu of the university application.
- It likely provides navigation options or buttons to access different functionalities or scenes within the application, such as Students, Courses, Evaluation, and Reports.
- The Home scene may also display important announcements, notifications, or general information about the university.






2. Students Scene:


- The Students scene is where administrators or users can manage student-related operations.
- It typically includes functionalities such as adding new students, updating student information, deleting students, and viewing a list of enrolled students.
- Users may interact with forms or input fields to enter student details, and there might be buttons or actions to perform CRUD (Create, Read, Update, Delete) operations on student records.
- Additionally, the Students scene may display a table or list of existing students with relevant information such as student ID, name, course enrollment, etc.




UNIVERSITY
YOUR TAGLINE




Student



Course



Evaluation



Reports

Student ID	Student Name	Department ID	CGPA	City	E-mail	Gender	Date of birth
1	John Doe	1	3.0	New York	john.doe@example.com	M	1998-05-15
2	Jane Smith	2	2.67	Los Ange...	jane.smith@example.com	F	1997-08-22
3	Bob Johnson	1	4.0	Chicago	bob.johnson@example.com	M	1999-01-10
4	Alice Williams	2	0.0	San Franc...	alice.williams@example.com	F	1996-11-30
5	Charlie Brown	1	1.0	Houston	charlie.brown@example.c...	M	1998-03-05
6	Eva Martinez	2	0.0	Miami	eva.martinez@example.com	F	1997-06-18
7	David Lee	1	0.0	Seattle	david.lee@example.com	M	1999-09-25
8	Sophia Chen	2	0.0	Boston	sophia.chen@example.com	F	1996-12-12
9	Michael Nguyen	1	0.0	Dallas	michael.nguyen@example...	M	1998-02-03
10	Olivia Garcia	2	0.0	Atlanta	olivia.garcia@example.com	F	1997-04-14
11	William Wang	1	1.67	Phoenix	william.wang@example.com	M	1999-07-07
12	Emma Davis	2	0.0	Denver	emma.davis@example.com	F	1996-10-20
13	Ahmed Hatem	1	4.0		ahmedhatem@gmail.com	M	2001-06-24

Remove Student

Add Student

Student_ID :

Enter student ID....

Student Name:

Enter student name...

Department_ID :

Enter student department ID...

Email :

Enter student Email...

Date of Birth :

Enter student birthdate...

Gender :

Male


Female

Add Student




3. Course Scene:


- The Courses scene is dedicated to managing course-related activities within the university application.
- Users can perform tasks like adding new courses, updating course details, deleting courses, and viewing a list of available courses.
- Similar to the Students scene, the Courses scene may feature input fields for entering course information and buttons for CRUD operations on course records.
- It typically displays a table or list of courses with details such as course ID, name, credit hours, department, etc.




UNIVERSITY
YOUR TAGLING




Student



Course



Evaluation



Reports

Course ID	Course Name	Credit Hours	Department_ID
11	SQL	3	1
12	Python	6	1
13	Data Warehouse	3	1
14	Big Data Tools	6	1
21	Cyber Security	6	2
22	Networking	6	2
23	Operating Systems	3	2
24	Ethical Hacking	3	2

Department ID	Department Name
1	Data Engineering
2	System Administration

Delete Course

Add Course

Course ID:

Course Name :

Credit Hours :

Department ID :

Add Courses

Delete Department

Add Department

Department_ID :


Department Name :

Add Department




4. Evaluation Scene:


- The Evaluation scene is where administrators or instructors handle student evaluations and grades.
- Users can update student grades for various courses, view evaluation records, and perform actions related to grading.
- It may include input fields for entering evaluation details like student ID, course ID, semester, grade, etc., along with buttons for submitting or updating grades.
- Additionally, the Evaluation scene may display a table of evaluation records showing student grades, course details, evaluation dates, etc.




UNIVERSITY
YOUR TAGLING




Student



Course



Evaluation



Reports

Student ID	Student Name	Course ID	Course Name	Grade	Credit Hours	Semester	Evaluation Date
1	John Doe	11	SQL	A	3	1	2024-01-08
1	John Doe	12	Python	B+	6	1	2023-01-25
1	John Doe	13	Data Warehouse	D+	3	2	2023-06-15
2	Jane Smith	22	Networking	B-	6	1	2023-01-20
2	Jane Smith	23	Operating Systems		3	2	2023-06-20
2	Jane Smith	24	Ethical Hacking		3	2	2023-06-05
3	Bob Johnson	11	SQL		3	1	2023-01-08
3	Bob Johnson	12	Python	A	6	1	2023-01-25
3	Bob Johnson	13	Data Warehouse		3	2	2023-06-15
3	Bob Johnson	14	Big Data Tools		6	2	2023-06-10
4	Alice Williams	22	Networking		6	1	2023-01-01
4	Alice Williams	23	Operating Systems		3	2	2023-06-01

UnEnroll Student

Course Evaluation

Student ID :

Enter student ID.....

Course ID :

Enter student department ID

Semester :

Enter Semester..

Grade :

Enter Student Grade..

Update Grade

Course Enrollment

Student ID :

Enter student ID.....

Course ID :

Enter student department ID

Course Name :

Enter Course Name..

Semester :

Enter Semester..

Enroll Student

5. Reports Scene:

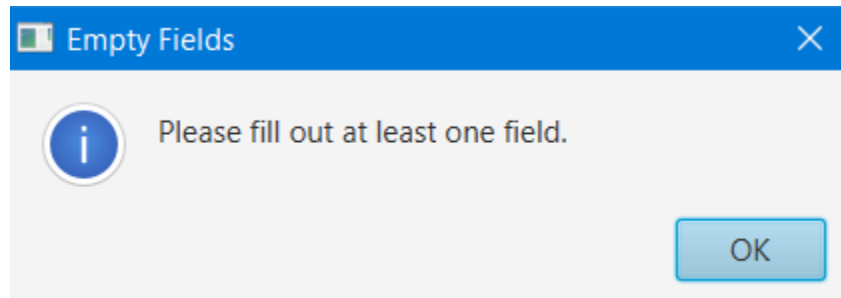
- The Reports scene provides insights and analysis based on data collected from student evaluations and course enrollments.
- Users can generate various types of reports, such as course enrollment statistics, GPA averages, grade distributions, etc.
- It typically includes a button or action to generate a report, which triggers data retrieval from the database and displays the report in a tabular format.
- The Reports scene may contain a table view with columns representing different report metrics or attributes, allowing users to analyze and interpret the data.

[illegible]

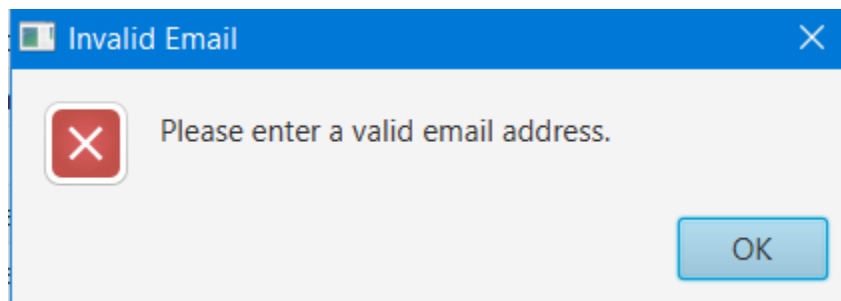


Validation and Anomalies Checks

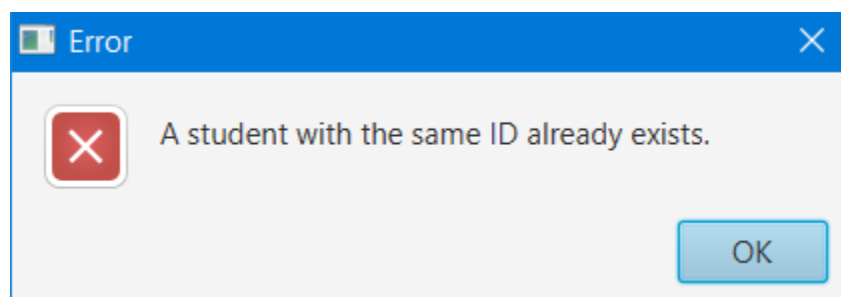
1. Handling Empty Fields:



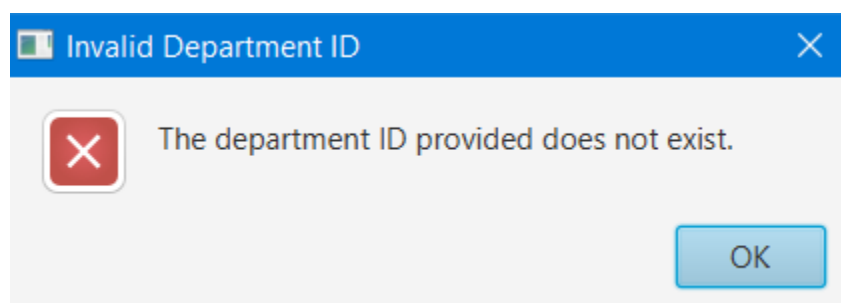
2. Handling E-mail Format:



3. Handling Primary Keys Repetition:

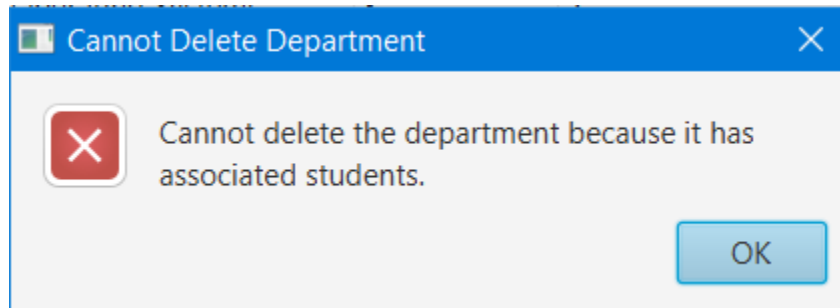


4. Handling Invalid Department ID:

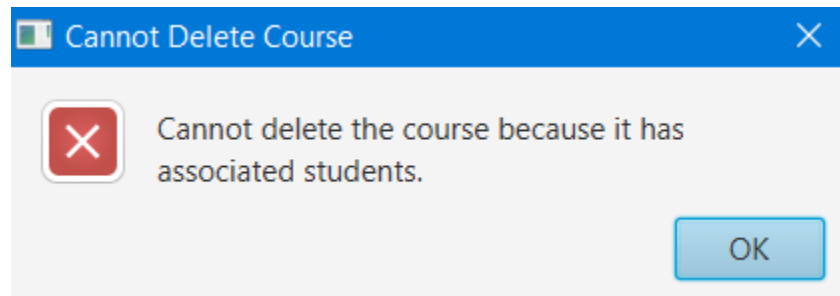




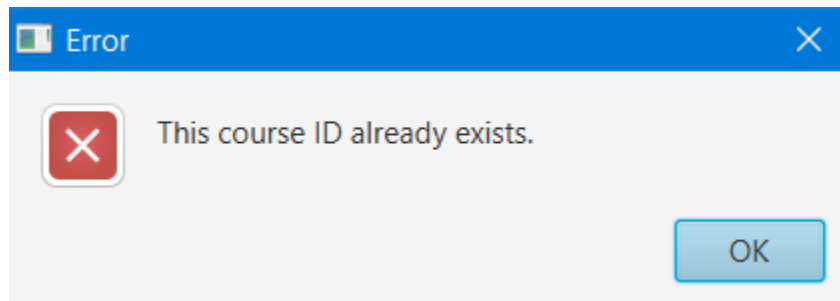
5. Handling Deleting Department with Enrolled Students:



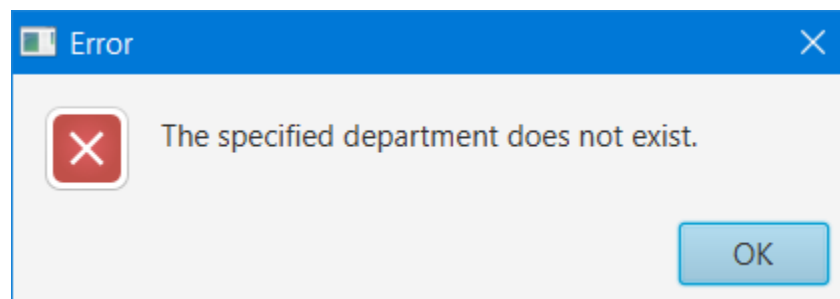
6. Handling Deleting Course with Enrolled Students:



7. Handling Adding Existing Course ID:

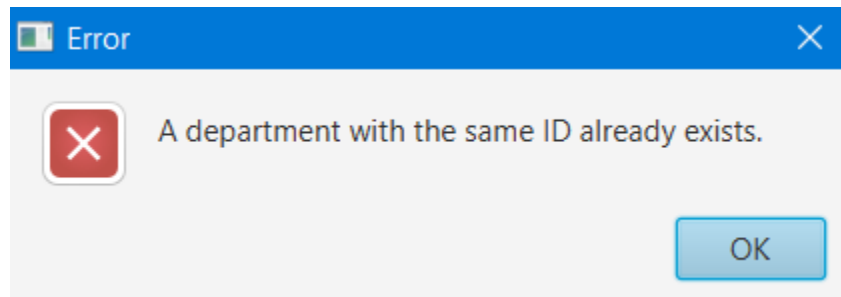


8. Handling Adding Course to Non-Existing Department:

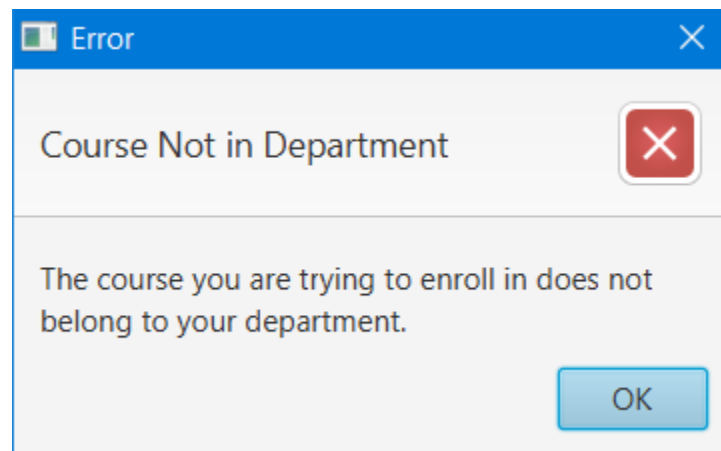




9. Handling Adding Department with Same ID:



10. Handling Enrolling Student into a Course not in his Department:



11. Handling Invalid Grades:

