



Atlas University Data Management System

Case Study

By: Abdullah Ahmed Salah Mohamed

Index:

1. Introduction

- Background
- Objective
- Overview of Components

2. Database Design

- Relational Database Schema Definition
- Table Design for Students, Courses, Departments, and Grades
- Normalization for Data Integrity
- Database Design Documentation

3. SQL Implementation

- SQL Scripts for Database Schema Creation
- Database Population with Sample Data
- Testing and Validation

4. PLSQL Implementation

- PLSQL Procedure for Updating Student Information
- PLSQL Function for Calculating GPA
- PLSQL Procedure for Updating Department Information
- PLSQL Procedure for Updating Course Information
- PLSQL Procedure for Deleting Department Information
- PLSQL Procedure for Deleting Course Information
- Testing PLSQL Procedures with Sample Data

5. Automation Scripts

- Bash Script for Database Backup
- Bash Script for Disk Space Monitoring and Alerts

6. Java Application Development

- CRUD Operations Implementation in Java
- Integration of Java Application with SQL Database
- Testing Java Application with Scenarios

7. Conclusion

- Summary of Project Accomplishments
- Challenges Faced and Overcome
- Future Recommendations

Introduction

1.1 Background

The Atlas University Data Management System Case Study is conceived to address the evolving needs of modern educational institutions. With the growing complexity of university operations, efficient data management becomes paramount. This project aims to harness the power of SQL, PLSQL, Bash scripting, Java SE, and OOP principles to design and implement a comprehensive data management system for Atlas University.

1.2 Objective

The primary objective of this project is to create a robust data management system that encompasses various facets of university operations, including student information, course management, departmental data, and grading. The integration of SQL, PLSQL, Bash scripting, and Java applications will facilitate seamless data handling, automation of critical tasks, and insightful reporting for better decision-making.

1.3 Overview of Components

The project involves several key components, each contributing to the overall success of the data management system:

- **Database Design:** Establishing a well-structured relational database schema for students, courses, departments, and grades, with a focus on normalization to ensure data integrity.
- **SQL Implementation:** Creating SQL scripts for building the database schema, populating it with sample data, and rigorously testing its correctness.
- **PLSQL Implementation:** Developing PLSQL procedures for updating student, department, and course information. As well as a Function that calculates GPA. Testing these procedures and functions with sample data.
- **Automation Scripts:** Implementing Bash scripts for essential tasks such as database backup and disk space monitoring.

- **Java Application Development:** Implementing CRUD operations, and integrating the application with the SQL database.
- **Integration and Reporting:** Developing a feature within the Java application to generate comprehensive reports, showcasing a list of courses, enrolled students, and average GPA for each course.
- **Project Presentation:** Summarizing the project's key features and achievements for effective communication and understanding.

The individual assignment nature of this project provides an opportunity for each participant to showcase their understanding and mastery of the skills acquired during the courses. The subsequent sections of this documentation will delve into each milestone, detailing the steps taken to accomplish the objectives set for each day of the project.

Database Design

2.1 Relational Database Schema Definition

Students Table:

- student_id (PK): Unique identifier for students.
- name: Student's name.
- dob: Date of birth.
- city: City where the student resides.
- street: Street address.
- year: Academic year.
- join_date: Date of joining.
- department_id (FK): Foreign key referencing the department.

Departments Table:

- department_id (PK): Unique identifier for departments.
- name: Department name.

Courses Table:

- course_id (PK): Unique identifier for courses.
- name: Course name.
- hours: Duration of the course in hours.
- max_grade: Maximum achievable grade.

Enrollments Table:

- student_id (FK): Foreign key referencing the student.
- course_id (FK): Foreign key referencing the course.
- grade: Student's grade.
- semester: Academic semester.
- year: Course year.

Department_Courses Table:

- department_id (FK): Foreign key referencing the department.
- course_id (FK): Foreign key referencing the course.

2.2 Entity-Relationship Diagram (ERD)

The ERD visually represents the relationships between different entities in the Atlas University Data Management System, offering a clear depiction of data flow and connectivity.

1. Students Table:

- Primary Key: student_id (PK)

2. Departments Table:

- Primary Key: department_id (PK)

3. Courses Table:

- Primary Key: course_id (PK)

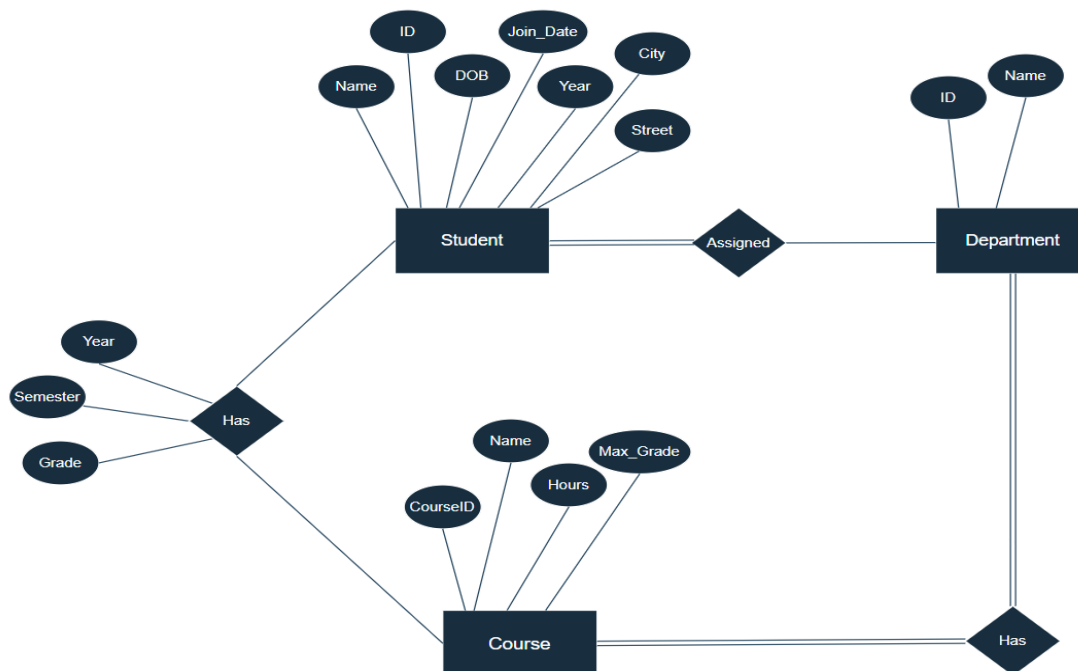
4. Enrollments Table:

- Foreign Keys: student_id (FK), course_id (FK)
- Associated Attributes: grade, semester, year

5. Department_Courses Table:

- Foreign Keys: department_id (FK), course_id (FK)

Entity-Relationship Diagram (ERD) Visualization:



Key Relationships:

The **Students** table is connected to the **Departments** table through the foreign key relationship (**department_id**).

The **Courses** table is connected to the **Enrollments** table through the foreign key relationship (**course_id**).

The **Students** and **Courses** tables are connected to the **Enrollments** table, representing the enrollment of students in specific courses.

The **Departments** and **Courses** tables are connected through the **Department_Courses** table, illustrating the relationship between departments and the courses they offer.

2.3 Normalization Importance

Elimination of Data Redundancy:

Normalization plays a vital role in eliminating data redundancy by organizing data into related tables. In the Atlas University Data Management System, it ensures that information about students, courses, and departments is stored efficiently without unnecessary duplication. Redundancy reduction not only optimizes storage space but also minimizes the risk of inconsistencies that may arise from duplicate data.

Data Integrity:

Normalization contributes to maintaining data integrity by adhering to the principles of Atomicity, Consistency, Isolation, and Durability (ACID). The structured organization of tables and relationships helps prevent anomalies such as update, insertion, and deletion errors. In the **Enrollments** table, for instance, normalization ensures that each grade is associated with a specific student, course, semester, and year, maintaining consistency and accuracy.

Consistent Database Structure:

A normalized database structure leads to consistency in the way data is stored, making it easier to manage and maintain. In the Atlas University system, this consistency ensures that each department is uniquely identified, each course has a defined duration, and each student's information is linked to their respective department. Consistency in data representation facilitates easier updates, reduces errors, and enhances overall system stability.

SQL Implementation

3.1.1 Create Students Table

```
CREATE TABLE Students (  
    student_id INT PRIMARY KEY,  
    name VARCHAR(255),  
    dob DATE,  
    city VARCHAR(255), street VARCHAR(255),  
    year INT,  
    join_date DATE, department_id INT,  
    FOREIGN KEY (department_id) REFERENCES Departments(department_id) ON DELETE SET NULL  
);
```

Purpose:

The **Students** table stores information about students, including their unique identifier (**student_id**), personal details, academic information, and the department they belong to.

Key Features:

Primary Key: **student_id**

Foreign Key: **department_id** referencing **Departments(department_id)** with ON DELETE SET NULL, allowing students to exist without a specified department.

3.1.2 Create Departments Table

```
CREATE TABLE Departments (  
    department_id INT PRIMARY KEY,  
    name VARCHAR(255)  
);
```

Purpose:

The **Departments** table holds information about university departments, such as their unique identifier (**department_id**) and department name (**name**).

Key Features:

Primary Key: department_id

3.1.3 Create Courses Table

```
CREATE TABLE Courses (  
    course_id INT PRIMARY KEY,  
    name VARCHAR(255),  
    hours INT,  
    max_grade INT  
);
```

Purpose:

The **Courses** table stores details about university courses, including their unique identifier (**course_id**), name, duration in hours (**hours**), and the maximum achievable grade (**max_grade**).

Key Features:

Primary Key: course_id

3.1.4 Create Enrollments Table

```
CREATE TABLE Enrollments (  
    student_id INT,  
    course_id INT,  
    grade INT ,  
    semester VARCHAR(255),  
    year INT,  
    PRIMARY KEY (student_id, course_id),  
    FOREIGN KEY (student_id) REFERENCES Students(student_id) ON DELETE SET NULL,  
    FOREIGN KEY (course_id) REFERENCES Courses(course_id) ON DELETE SET NULL  
);
```

Purpose:

The **Enrollments** table represents the enrollment of students in courses, tracking details such as student ID, course ID, grade, semester, and academic year.

Key Features:

Composite Primary Key: (student_id, course_id)

Foreign Keys: **student_id** referencing **Students(student_id)** and **course_id** referencing **Courses(course_id)** with ON DELETE SET NULL, allowing flexibility in managing enrollments.

3.1.5 Create Department_Courses Table

```
CREATE TABLE Department_Courses (  
    department_id INT,  
    course_id INT,  
    PRIMARY KEY (department_id, course_id),  
    FOREIGN KEY (department_id) REFERENCES Departments(department_id) ON DELETE SET NULL,  
    FOREIGN KEY (course_id) REFERENCES Courses(course_id) ON DELETE SET NULL  
);
```

Purpose:

The **Department_Courses** table establishes the relationship between departments and the courses they offer, linking department IDs with course IDs.

Key Features:

Composite Primary Key: (department_id, course_id)

Foreign Keys: **department_id** referencing **Departments(department_id)** and **course_id** referencing **Courses(course_id)** with ON DELETE SET NULL, ensuring referential integrity.

These SQL scripts create the foundational tables for the Atlas University Data Management System, establishing relationships and constraints to maintain data integrity. In the next step, we can move on to populating the database with sample data. Let me know if you have any specific details to add or if you'd like to proceed.

3.2 Database Population with Sample Data

After creating the tables, the next step is to populate them with sample data to ensure that the database is functioning as intended.

STUDENT_ID	NAME	DOB	CITY	STREET	YEAR	JOIN_DATE	DEPARTMENT_ID
1	John Donner	1/1/2000	Cityville	123 Main Street	3	9/1/2021	1
2	Jane Smith	2/28/1998	Los Angeles	456 Oak St	2	8/15/2020	2
3	Bob Johnson	11/10/1997	Chicago	789 Pine St	4	1/5/2022	1
4	Alice White	8/20/1996	San Francisco	234 Elm St	3	10/1/2021	2
5	Charlie Brown	4/5/1999	Seattle	567 Birch St	2	2/15/2022	1
6	Eva Green	9/30/1997	Miami	890 Cedar St	4	9/10/2020	3
7	David Lee	6/18/1998	Dallas	345 Maple St	3	11/20/2021	1
8	Grace Taylor	1/25/1996	Houston	678 Pine St	4	12/5/2020	2
9	Frank Miller	12/12/1999	Phoenix	901 Oak St	2	3/1/2022	3
10	Helen Wilson	3/8/1997	Denver	123 Spruce St	4	8/10/2021	1
11	Ivan Rodriguez	10/22/1998	Atlanta	456 Cedar St	4	7/15/2020	2
12	Jill Turner	7/14/1996	Boston	789 Birch St	2	4/1/2022	3
13	Kevin Harris	5/2/1999	Philadelphia	234 Maple St	3	10/15/2021	1
14	Laura Davis	2/19/1997	San Diego	567 Elm St	4	11/30/2020	2
15	Mark Robinson	9/5/1996	Austin	890 Spruce St	4	1/15/2022	3
16	Nina Martinez	4/12/1998	Chicago	345 Oak St	3	7/20/2021	1
17	Oliver Moore	11/28/1999	Seattle	678 Cedar St	4	6/5/2020	2
18	Pamela King	8/3/1996	Los Angeles	901 Elm St	2	2/20/2022	3
19	Quincy Johnson	4/18/1997	Miami	123 Birch St	3	9/25/2021	1
20	Rachel Turner	1/11/1998	Denver	456 Spruce St	4	8/30/2020	2

DEPARTMENT_ID	COURSE_ID
1	101
1	102
1	107
1	110
2	101
2	102
2	103
2	104
2	108
3	102
3	105
3	106
3	109

STUDENT_ID	COURSE_ID	GRADE	SEMESTER	YEAR
1	101	95	Fall	2021
1	102	87	Spring	2022
2	102	90	Fall	2021
3	103	82	Spring	2022
4	104	94	Fall	2021
5	105	88	Spring	2023
6	106	89	Fall	2023
7	107	92	Spring	2022
8	108	96	Fall	2021
9	109	85	Spring	2023
10	110	98	Fall	2023
11	101	89	Spring	2022
12	102	82	Fall	2021
13	103	94	Spring	2023
14	104	89	Fall	2023
15	105	91	Spring	2022
16	106	96	Fall	2023
17	107	88	Spring	2023
18	108	90	Fall	2021
19	109	93	Spring	2022
20	110	87	Fall	2021
1	103	80	Spring	2022

COURSE_ID	NAME	HOURS	MAX_GRADE
101	Programming	3	100
102	Calculus I	4	100
103	World History	3	100
104	Database Mang	3	100
105	Linear Algebra	3	100
106	American Rev	3	100
107	Web Dev	4	100
108	Statistics	3	100
109	Ancient Civil	3	100
110	Software Eng	4	100
111	Calculus II	4	100
100	new	2	100

DEPARTMENT_ID	NAME
1	Computer Science
2	Mathematics
3	History

These sample data entries represent a basic scenario with students enrolled in courses, each associated with a specific department.

3.3 Testing and Validation

Testing and validation are crucial to ensure the correctness and reliability of the database. Key aspects to consider include:

- **Data Accuracy:** Verify that the sample data accurately reflects the relationships between students, departments, courses, and enrollments.
- **Referential Integrity:** Confirm that foreign key constraints are properly enforced, ensuring that references between tables are valid.
- **Primary Key Uniqueness:** Ensure that primary key values are unique for each record in their respective tables.
- **Data Consistency:** Validate that data types and constraints are applied correctly, preventing inconsistencies in data representation.
- **Query Execution:** Execute sample queries to retrieve information from the database and confirm that the results align with expectations.
- **Error Handling:** Test error scenarios, such as attempting to insert duplicate primary key values, to validate that appropriate error messages or actions are triggered.

By rigorously testing and validating the database, you can identify and address any issues early in the development process, ensuring a robust and reliable data management system for Atlas University.

PLSQL Implementation

4.1.1 PLSQL Procedure for Updating Student Information

```
CREATE OR REPLACE PROCEDURE ADMIN.update_stud_info (  
    v_student_id INTEGER,  
    v_name VARCHAR2,  
    v_street VARCHAR2,  
    v_dob DATE,  
    v_city VARCHAR2,  
    v_year INTEGER,  
    v_join_date DATE,  
    v_department_id INTEGER  
) AS  
BEGIN  
    UPDATE students  
    SET  
        NAME = v_name,  
        DOB = v_dob,  
        CITY = v_city,  
        STREET = v_street,  
        YEAR = v_year,  
        JOIN_DATE = v_join_date,  
        DEPARTMENT_ID = v_department_id  
    WHERE STUDENT_ID = v_student_id;  
  
    IF SQL%ROWCOUNT = 0 THEN  
        DBMS_OUTPUT.PUT_LINE('No student found with ID ' || v_student_id);  
    ELSE  
        DBMS_OUTPUT.PUT_LINE('Update successful for student ID ' || v_student_id);  
    END IF;  
  
    COMMIT;  
END;  
/
```

4.1.2 PLSQL Procedure for Updating Department Information

```
CREATE OR REPLACE PROCEDURE ADMIN.update_department (  
    p_department_id IN NUMBER,  
    p_new_department_name IN VARCHAR2  
) AS  
BEGIN  
    UPDATE departments  
    SET name = p_new_department_name  
    WHERE department_id = p_department_id;  
  
    COMMIT;  
END;  
/
```

4.1.3 PLSQL Procedure for Updating Course Information

```
CREATE OR REPLACE PROCEDURE ADMIN.update_course (  
    p_course_id IN NUMBER,  
    p_new_name IN VARCHAR2,  
    p_new_hours IN NUMBER,  
    p_new_max_grade IN NUMBER  
) AS  
BEGIN  
    UPDATE courses  
    SET  
        name = p_new_name,  
        hours = p_new_hours,  
        max_grade = p_new_max_grade  
    WHERE course_id = p_course_id;  
  
    COMMIT;  
END;  
/
```

4.1.4 PLSQL Procedure for Deleting Department Information

```
CREATE OR REPLACE PROCEDURE ADMIN.delete_department (  
    p_department_id IN NUMBER  
) AS  
BEGIN  
    DELETE FROM enrollments  
    WHERE student_id IN (SELECT student_id FROM students WHERE department_id = p_department_id);  
  
    DELETE FROM department_courses WHERE department_id = p_department_id;  
  
    DELETE FROM departments WHERE department_id = p_department_id;  
  
    COMMIT;  
END delete_department;  
/
```

4.1.5 PLSQL Procedure for Deleting Course Information

```
CREATE OR REPLACE PROCEDURE ADMIN.delete_course (  
    p_course_id IN NUMBER  
) AS  
BEGIN  
    DELETE FROM enrollments WHERE course_id = p_course_id;  
  
    DELETE FROM department_courses WHERE course_id = p_course_id;  
  
    DELETE FROM courses WHERE course_id = p_course_id;  
  
    COMMIT;  
END delete_course;  
/
```

4.1.6 PLSQL Function for Calculating GPA

```
CREATE OR REPLACE FUNCTION ADMIN.gpa_stud_info (  
    v_student_id NUMBER,  
    v_semester VARCHAR2,  
    v_year NUMBER  
) RETURN NUMBER  
AS  
    v_max_grade NUMBER;  
    v_gpa NUMBER;  
    v_name VARCHAR2(255);  
BEGIN  
    SELECT SUM(e.grade * c.hours), SUM(c.hours) AS gpa  
    INTO v_gpa, v_max_grade  
    FROM students s  
    JOIN enrollments e ON s.student_id = e.student_id  
    JOIN courses c ON e.course_id = c.course_id  
    WHERE s.student_id = v_student_id AND e.semester = v_semester AND e.year = v_year  
    GROUP BY s.student_id, e.semester, e.year;  
  
    IF v_max_grade > 0 THEN  
        v_gpa := (v_gpa / v_max_grade) * 0.04;  
    ELSE  
        v_gpa := NULL;  
    END IF;  
  
    RETURN v_gpa;  
END;  
/
```

4.2 Testing PLSQL Procedures with Sample Data

To test these procedures, you can execute them with sample data. For example:

```
BEGIN  
    ADMIN.update_stud_info(1, 'Updated Name', 'Updated Street', TO_DATE('1990-05-15', 'YYYY-MM-DD'), 'Updated City', 4, TO_DATE('2022-01-15', 'YYYY-MM-DD'), 103);  
    ADMIN.update_department(101, 'Updated Computer Science');  
    ADMIN.update_course(201, 'Updated Programming', 60, 95);  
    ADMIN.delete_department(103);  
    ADMIN.delete_course(203);  
    DBMS_OUTPUT.PUT_LINE('GPA: ' || ADMIN.gpa_stud_info(1, 'Fall', 2021));  
END;  
/
```

These test examples demonstrate how to call and test each PLSQL procedure and function with sample data. The provided test cases cover updating student information, department information, course information, and deleting departments and courses, along with calculating GPA.

Automation Scripts

5.1 Bash Script for Database Backup

```
#!/bin/bash

# Oracle Database Connection Details
DB_USER=admin
DB_PASSWORD=root
DB_SID=XE

# Backup Directory
BACKUP_DIR="/path/to/backup_directory"

# Admin Schema to be backed up
ADMIN_SCHEMA="ADMIN@XE"

# Date Format for Backup File
DATE_FORMAT=$(date +"%Y%m%d_%H%M%S")

# Export File Name (only the file name, not the full path)
EXPORT_FILE="${ADMIN_SCHEMA}_backup_${DATE_FORMAT}.dmp"

# Timestamp for logging
TIMESTAMP=$(date +"%Y-%m-%d %H:%M:%S")

# Oracle Data Pump Export Command
expdp ${DB_USER}/${DB_PASSWORD}@${DB_SID} DIRECTORY=DATA_PUMP_DIR DUMPFILE=${BACKUP_DIR}/${EXPORT_FILE} SCHEMAS=${ADMIN_SCHEMA}

# Check if the export was successful
if [ $? -eq 0 ]; then
    echo "${TIMESTAMP} Database backup successful. File: ${EXPORT_FILE}"
else
    echo "${TIMESTAMP} Error: Database backup failed."
fi
```

Explanation:

This script uses Oracle Data Pump (expdp) to export the specified schema from the database.

It includes parameters such as database connection details, backup directory, schema to be backed up, and a timestamped export file name.

After the export, it checks the exit status to determine if the backup was successful and logs the result.

5.2 Bash Script for Disk Space Monitoring and Alerts

```
#!/bin/bash

alert_on=60

disk_usage=$(df -h / | awk 'NR==2 {gsub(/^[0-9]/, "", $6); print $6}')

alert_file="/path/to/alert_file.txt"

timestamp=$(date +"%Y-%m-%d %H:%M:%S")

if [ "$disk_usage" -gt "$alert_on" ]; then
    echo "Disk space exceeded $alert_on%, it is $disk_usage% at $timestamp" >> "$alert_file"
else
    echo "Disk space did not exceed $alert_on%, it is $disk_usage% at $timestamp" >> "$alert_file"
fi
```


Explanation:

This script monitors disk space usage using `df` and extracts the percentage usage for the specified directory (in this case, `/`).

It compares the current disk usage with the predefined threshold (`alert_on`) and logs the result with a timestamp to an alert file.

If disk usage exceeds the threshold, an alert is written to the file.

5.3 Scheduling Scripts with Windows Task Scheduler

- Ensure that the Bash executable path is correctly specified in the Task Scheduler.
- Verify that the scripts have the necessary permissions to execute.
- If the script relies on environment variables, set them explicitly in the script or configure Task Scheduler to run the script in a specific directory.
- Regularly check the Task Scheduler logs for any issues with script execution.

By scheduling these tasks with Windows Task Scheduler, it is automating the execution of Bash scripts, simulating the scheduling functionality similar to `cron` on Linux.

Java Application Development

In this section, we'll delve into the Java application development for the University's data management system. The application will utilize Object-Oriented Programming (OOP) principles and interact with the SQL database.

6.1 Database Connection Configuration

```
public class DatabaseManager {  
  
    private static final String URL = "jdbc:oracle:thin:@localhost:1521:XE";  
    private static final String USER = "admin";  
    private static final String PASSWORD = "root";  
  
    public static Connection getConnection() throws SQLException {  
        DriverManager.registerDriver(new OracleDriver());  
        return DriverManager.getConnection(URL, USER, PASSWORD);  
    }  
}
```

6.2 Student Information CRUD Operations

Inserting into the students table, and checking first if the student exists

```
public static boolean studentExists(int studentId) throws SQLException {  
    Connection connection = null;  
    PreparedStatement statement = null;  
    ResultSet resultSet = null;  
  
    try {  
        connection = getConnection();  
        String query = "SELECT COUNT(*) FROM students WHERE student_id = ?";  
  
        statement = connection.prepareStatement(query);  
        statement.setInt(1, studentId);  
        resultSet = statement.executeQuery();  
        return resultSet.next();  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}  
  
public static void insertStudent(StudentDTO student) throws SQLException {  
    Connection connection = null;  
    PreparedStatement preparedStatement = null;  
  
    try {  
        connection = getConnection();  
        String sql = "INSERT INTO STUDENTS (student_id, name, dob, city, street, year, join_date, department_id) VALUES (?, ?, TO_DATE(?, 'YYYY-MM-DD'), ?, ?, ?, ?)";  
        preparedStatement = connection.prepareStatement(sql);  
        preparedStatement.setInt(1, student.getId());  
        preparedStatement.setString(2, student.getName());  
        preparedStatement.setString(3, student.getDob());  
        preparedStatement.setString(4, student.getCity());  
        preparedStatement.setString(5, student.getStreet());  
        preparedStatement.setInt(6, student.getYear());  
        preparedStatement.setDate(7, student.getJoinDate());  
        preparedStatement.setInt(8, student.getDepartmentId());  
        preparedStatement.executeUpdate();  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

Calling the update student procedure

```
public static void updateStudentInfo(int student_id, String name, String street, String dob, String city, int year, String joinDate, int departmentId) throws SQLException {  
    Connection connection = null;  
    CallableStatement callableStatement = null;  
  
    try {  
        connection = getConnection();  
        String sql = "{call update_stud_info(?, ?, TO_DATE(?, 'YYYY-MM-DD'), ?, ?, TO_DATE(?, 'YYYY-MM-DD'), ?)}";  
        callableStatement = connection.prepareCall(sql);  
        callableStatement.setInt(1, student_id);  
        callableStatement.setString(2, name);  
        callableStatement.setString(3, dob);  
        callableStatement.setString(4, city);  
        callableStatement.setInt(5, year);  
        callableStatement.setDate(6, joinDate);  
        callableStatement.setInt(7, departmentId);  
        callableStatement.execute();  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

Calling the Calc GPA function

```
public static Double calculateGPA(int student_id, String semester, int year) throws SQLException {  
    Connection connection = null;  
    CallableStatement callableStatement = null;  
    Double gpa = null;  
  
    try {  
        connection = getConnection();  
        String sql = "{ ? = call gpa_stud_info(?, ?, ?) }";  
        callableStatement = connection.prepareCall(sql);  
        callableStatement.setInt(1, student_id);  
        callableStatement.setString(2, semester);  
        callableStatement.setInt(3, year);  
        callableStatement.execute();  
        gpa = callableStatement.getDouble(1);  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

6.3 Department Information Retrieval

```
public static DepartmentsDTO getDepartmentById(int departmentId) throws SQLException {
    DepartmentsDTO department = null;

    try (Connection connection = getConnection();
        PreparedStatement preparedStatement = connection.prepareStatement(
```

Course Results and Information

```
public static List<CourseResultDTO> getCourseResultsByDepartmentId(int departmentId) throws SQLException {
    List<CourseResultDTO> courseResults = new ArrayList<>();

    try (Connection connection = getConnection();
        PreparedStatement statement = connection.prepareStatement(
```

6.4 Department and Course CRUD Operations

```
public static boolean updateDepartment(int departmentId, String newDepartmentName) throws SQLException {
    Connection connection = null;
    CallableStatement callableStatement = null;

public static boolean updateCourse(int courseId, String newName, int newHours, int newMaxGrade) throws SQLException {
    Connection connection = null;
    CallableStatement callableStatement = null;

public static boolean addDepartment(int departmentId, String departmentName) throws SQLException {
    Connection connection = null;
    PreparedStatement preparedStatement = null;

public static boolean addCourse(int courseId, String courseName, int hours, int maxGrade) throws SQLException {
    Connection connection = null;
    PreparedStatement preparedStatement = null;

    try {

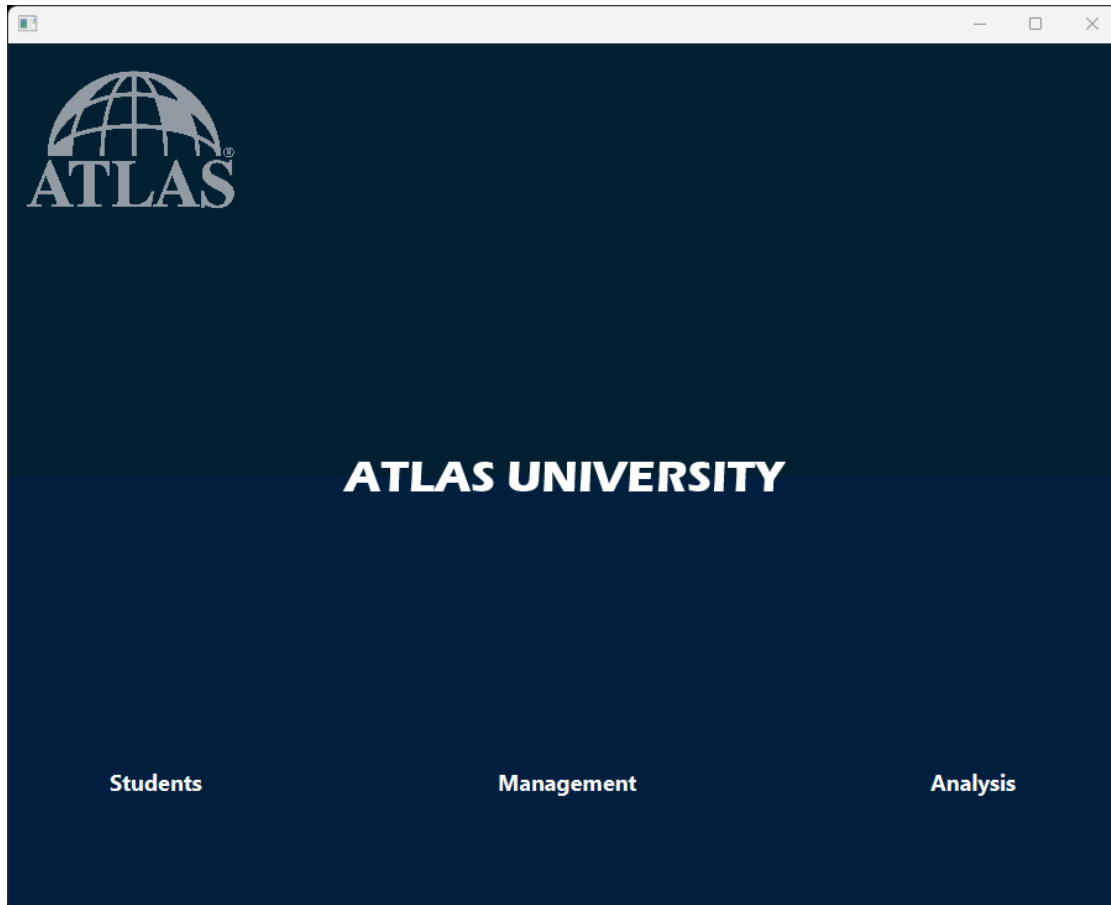
public static boolean deleteCourse(int courseId) throws SQLException {
    if (!courseExists(courseId)) {
        return false; // Course does not exist
    }

public static boolean deleteDepartment(int departmentId) throws SQLException {
    if (!departmentExists(departmentId)) {
        return false; // Department does not exist
    }
```

Calling out these methods ensures a smooth interaction between the database and the application. Making sure that every controller has the appropriate methods to call.

AND HERE IS THE MOMENT TO UNVEIL THE CURTAINS OF THE APP!

1- Main Page



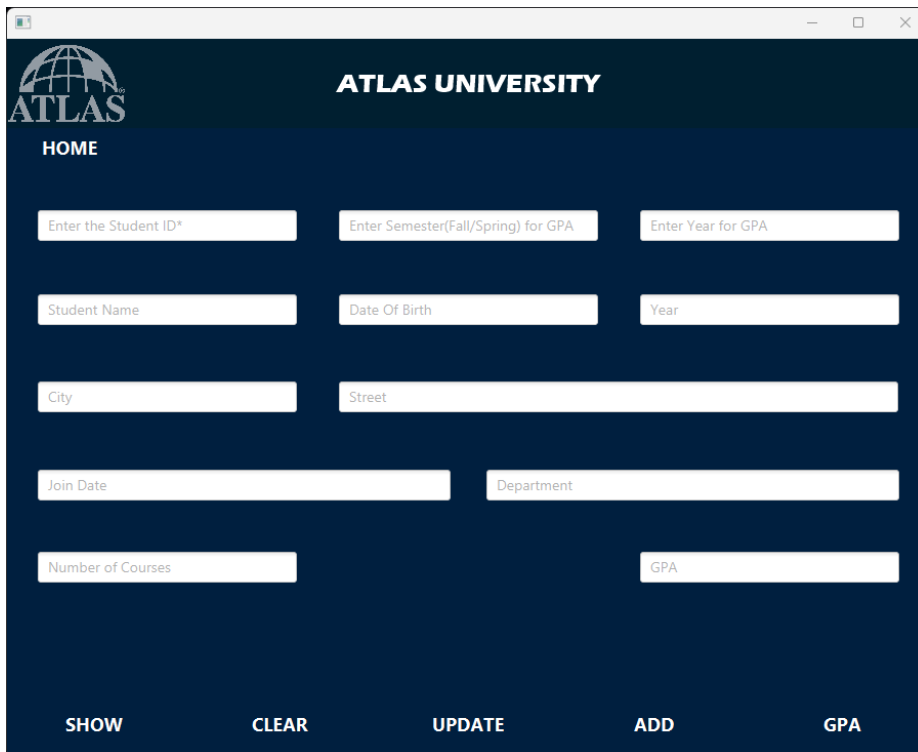
On this page you have the front of the App, and you get to choose your next destination.

You can choose between the Students page; in which you can check any student info or update it.

You can choose the Management page; in which you can check more reports and retrieve departments and courses information, as well applying CRUD there on the Departments or Courses.

You can choose the Analysis page; in which you will see a summary of visuals on almost every important information about Atlas University.

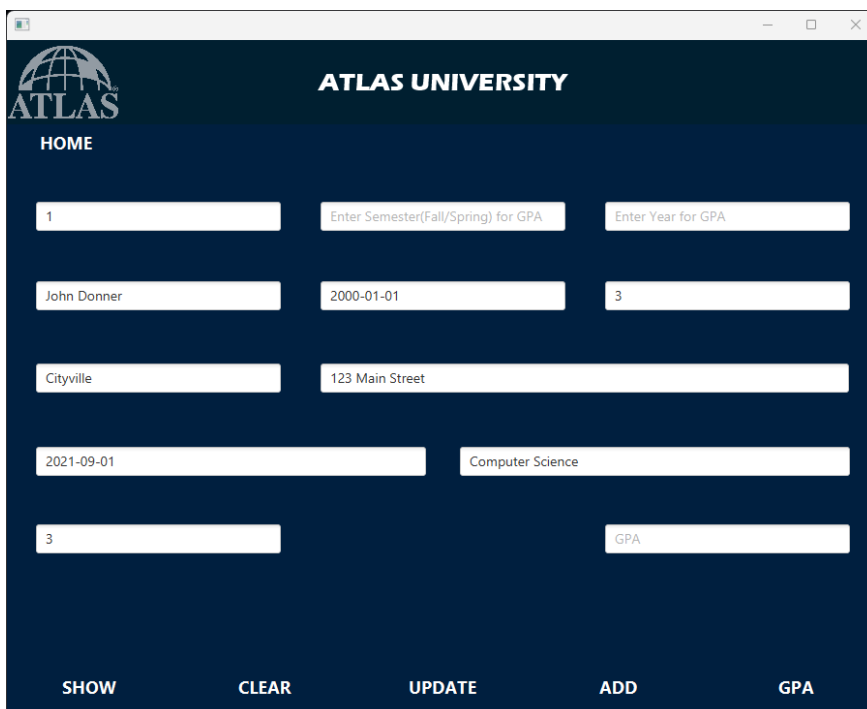
2- Students Page



The screenshot shows a web browser window with the title "ATLAS UNIVERSITY". The page has a dark blue header with the "ATLAS" logo on the left and the text "ATLAS UNIVERSITY" on the right. Below the header, the word "HOME" is displayed. The main content area contains a form with several input fields: "Enter the Student ID*", "Enter Semester(Fall/Spring) for GPA", "Enter Year for GPA", "Student Name", "Date Of Birth", "Year", "City", "Street", "Join Date", "Department", "Number of Courses", and "GPA". At the bottom of the form, there are five buttons: "SHOW", "CLEAR", "UPDATE", "ADD", and "GPA".

On here you can show a student by entering the student ID, which is unique.

For the GPA calculation as it shows, you will need to enter the semester of the course and the year as well. And you can always go to main page with Home button.



The screenshot shows the same web browser window as the previous one, but with sample data entered into the form fields. The "Student ID" field contains "1", the "Semester" field contains "2000-01-01", and the "Year" field contains "3". The "Student Name" field contains "John Donner", the "City" field contains "Cityville", and the "Street" field contains "123 Main Street". The "Join Date" field contains "2021-09-01" and the "Department" field contains "Computer Science". The "Number of Courses" field contains "3" and the "GPA" field is empty. The buttons at the bottom remain the same: "SHOW", "CLEAR", "UPDATE", "ADD", and "GPA".

3- Management Page

The screenshot shows the 'ATLAS UNIVERSITY' management page. On the left, under 'Department Section:', there are two input fields: 'Enter Department ID*' and 'Enter Department Name*'. Below these, under 'Course Section:', there are four input fields: 'Enter Course ID*', 'Enter Course Name*', 'Enter Course Hours', and 'Enter Course Grade'. To the right of these inputs are two empty table placeholders, each displaying 'No columns in table'. At the bottom of the page are five buttons: 'SHOW', 'CLEAR', 'UPDATE', 'ADD', and 'DELETE'.

On the management page, you will need to enter either the Department ID or Course ID to retrieve the info of either of them. It will show an error message if both are filler or both are empty.

Clicking on the table view when a department is chosen, you will be able to retrieve the course and student info (On mouse click feature).

This screenshot shows the same management page but with data populated. In the 'Department Section:' area, the 'Enter Department ID*' field contains the value '1'. The 'Department Name' table now shows data for 'Computer Science' with an 'Average GPA' of 3.59. The 'Course Section:' area has all its input fields empty. The 'Course Name' table shows data for four courses: 'Calculus I' (4 hours, 3 students, 3.45 GPA), 'Web Dev' (4 hours, 2 students, 3.6 GPA), 'Programming' (3 hours, 2 students, 3.68 GPA), and 'Software Eng' (4 hours, 2 students, 3.7 GPA). The 'Student Name' table shows data for seven students: 'Nina Martinez' (3 years, 1 course, 3.84 GPA), 'Quincy Johnson' (3 years, 1 course, 3.72 GPA), 'Charlie Brown' (2 years, 1 course, 3.52 GPA), 'Kevin Harris' (3 years, 1 course, 3.76 GPA), 'John Donner' (3 years, 3 courses, 3.49 GPA), 'Helen Wilson' (4 years, 1 course, 3.92 GPA), and 'Bob Johnson' (4 years, 1 course, 3.28 GPA). The bottom buttons remain the same.

Department Name	Average GPA
Computer Science	3.59

Course Name	Hours	Student Count	Average GPA
Calculus I	4	3	3.45
Web Dev	4	2	3.6
Programming	3	2	3.68
Software Eng	4	2	3.7

Student Name	Year	Course Count	Student Avg GPA
Nina Martinez	3	1	3.84
Quincy Johnson	3	1	3.72
Charlie Brown	2	1	3.52
Kevin Harris	3	1	3.76
John Donner	3	3	3.49
Helen Wilson	4	1	3.92
Bob Johnson	4	1	3.28

ATLAS UNIVERSITY

HOME

Department Section:

1

Enter Department Name*

Course Section:

105

Enter Course Name*

Enter Course Hours

Enter Course Grade

No columns in table

No columns in table

Error

Error

Please enter either a Department ID or a Course ID, not both.

OK

No columns in table

SHOW

CLEAR

UPDATE

ADD

DELETE

ATLAS UNIVERSITY

HOME

Department Section:

Enter Department ID*

Enter Department Name*

Course Section:

102

Enter Course Name*

Enter Course Hours

Enter Course Grade

Department Name	Course Name	Hours	Grade	Student Count	Average GPA
Computer Science	Calculus I	4	100	3	3.45
Mathematics					
History					

Student Name	Year	Semester	GPA
John Donner	3	Spring	3.48
Jane Smith	2	Fall	3.6
Jill Turner	2	Fall	3.28

SHOW

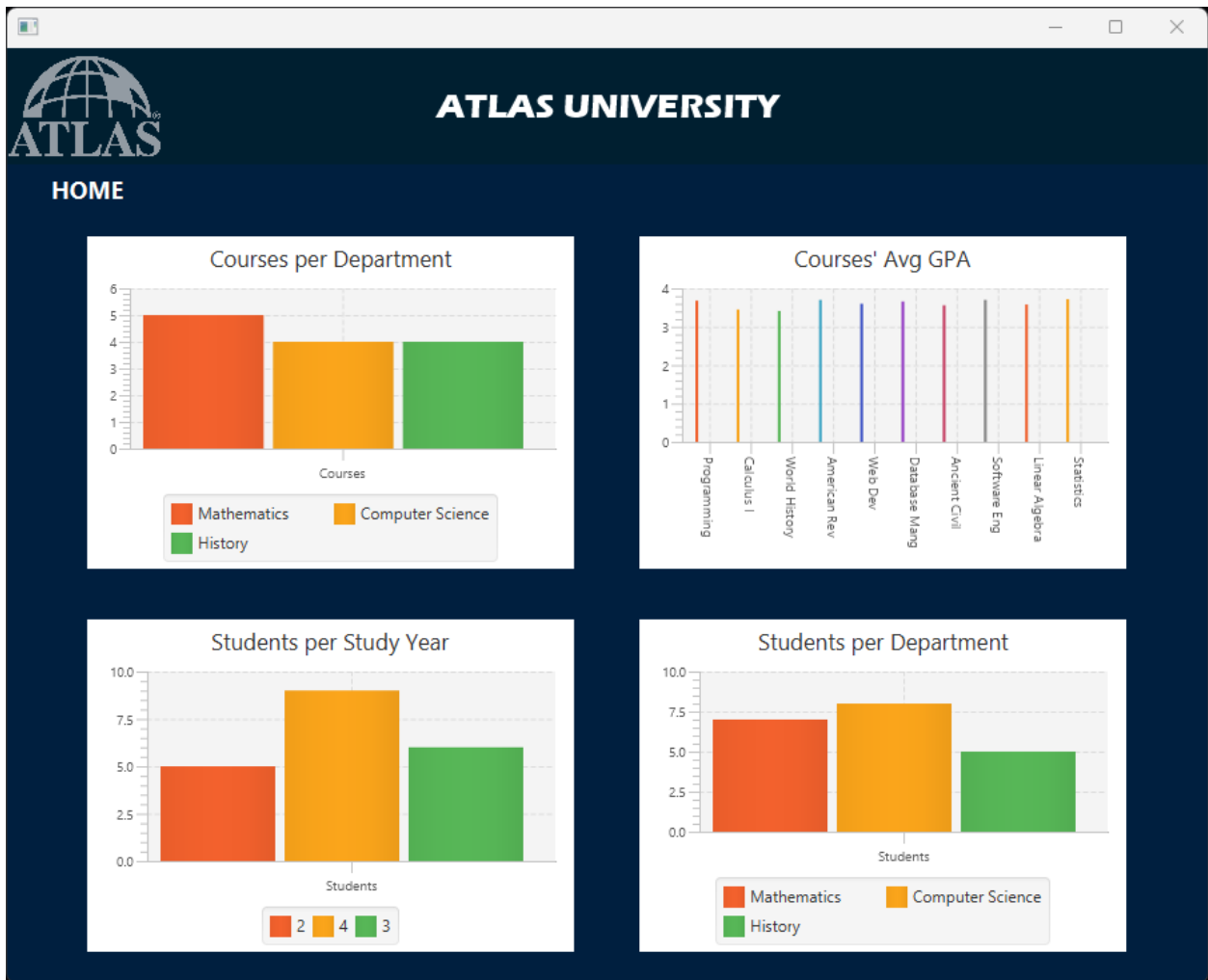
CLEAR

UPDATE

ADD

DELETE

4- Analysis Page



On the Analysis page you will be able to see 4 different relationships:

- Courses per Department: shows how many student is assigned in each department.
- Courses' average GPA: shows the average GPA of the students for a specific course.
- Students per Study Year: shows the number of students in each study year group.
- Students per Department: shows the number of students on each department.

Conclusion

In conclusion, the comprehensive case study on university database design and management has provided a thorough exploration of various aspects, ranging from conceptualization and normalization to SQL implementation, PL/SQL procedures, and Java application development. This journey has equipped you with the skills and understanding needed to design, implement, and build a robust university database system.

Embarking on this case study has allowed you to grasp fundamental concepts such as relational database schema design, entity-relationship diagrams, normalization, and the importance of database design documentation. The SQL implementation, including the creation of tables and population with sample data, demonstrated the practical application of theoretical concepts.

The discussion on PL/SQL procedures and functions further enriched your knowledge by covering essential operations like updating student information, calculating GPA, updating department and course information, and handling deletions. This segment showcased the power and versatility of PL/SQL in managing and manipulating data within an Oracle database.

The automation scripts, implemented in Bash, added a layer of operational efficiency by introducing backup procedures and disk space monitoring. Scheduling these scripts using Windows Scheduler ensures the regular execution of critical tasks, enhancing the system's reliability and data integrity.

The Java application development section presented a robust Database Manager class, showcasing CRUD operations, information retrieval, and various functionalities related to students, departments, courses, and GPA calculations. This section encapsulated the integration of database operations within a Java application, emphasizing modularity and adherence to OOP principles.

Lastly, the inclusion of automation scripts and the Java application acknowledges the importance of real-world applicability and operational efficiency in database systems.

This accomplishment is a testament to dedication and collaborative effort in crafting a holistic case study. Appreciation is extended to those who contributed, guided, and supported me throughout this learning journey. As I continue to explore and apply database management concepts, may this case study serve as a valuable reference and foundation for my future endeavors!