Technical Details: SQL Portion - Alfredo Rosario

This section outlines the step-by-step development process of the *Student Grading and Attendance System*, including database design, logic behind key features, and code snippets. The backend was developed using MySQL to manage relational data effectively, focusing on a simple and scalable structure suitable for a classroom environment.

Step 1: Database Creation

We created a new database named student_system in MySQL Workbench.

```
CREATE DATABASE IF NOT EXISTS student_system;
    USE student_system;
```

Step 2: Creating Tables with Constraints

Six core tables were created to manage students, teachers, courses, grades, attendance, and enrollment relationships. The schema includes appropriate constraints such as primary keys, foreign keys, NOT NULL, CHECK, and ENUM types for data integrity and validation. Password fields were given longer lengths (VARCHAR(255)) to store hashed values.

Students Table

```
CREATE TABLE students (
        id INT AUTO_INCREMENT PRIMARY KEY,
        first_name VARCHAR(50) NOT NULL,
        last_name VARCHAR(50) NOT NULL,
        email VARCHAR(100) NOT NULL,
        password_ VARCHAR(255) NOT NULL
    );
```

Teachers Table

```
    CREATE TABLE teachers (
        id INT AUTO_INCREMENT PRIMARY KEY,
        first_name VARCHAR(50) NOT NULL,
        last_name VARCHAR(50) NOT NULL,
        email VARCHAR(100) NOT NULL,
        staff_password VARCHAR(255) NOT NULL
    );
```

Courses Table

```
    CREATE TABLE courses (
        course_id INT AUTO_INCREMENT PRIMARY KEY,
        course_name VARCHAR(100) NOT NULL,
        teacher_id INT,
        FOREIGN KEY (teacher_id) REFERENCES teachers(id)
    );
```

Enrollments Table

```
CREATE TABLE enrollments (
    student_id INT,
    course_id INT,
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES students(id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);
```

Grades Table

This table stores individual student grades for each course.

```
CREATE TABLE grades (
    grade_id INT AUTO_INCREMENT PRIMARY KEY,
    student_id INT,
    course_id INT,
    grade DECIMAL(5,2) CHECK (grade >= 0 AND grade <= 100),
    FOREIGN KEY (student_id) REFERENCES students(id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);
```

Attendance Table

```
CREATE TABLE attendance (
    attendance_id INT AUTO_INCREMENT PRIMARY KEY,
    student_id INT NOT NULL,
    course_id INT NOT NULL,
    date DATE NOT NULL,
    status ENUM('Present', 'Absent') NOT NULL,
    FOREIGN KEY (student_id) REFERENCES students(id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);
```

Step 3: Sample Data Insertion

Sample data was added to simulate real-world usage, including students, teachers, courses, grade entries, and attendance logs.

```
INSERT INTO students (first_name, last_name, email, password_) VALUES
    ('Herbert', 'Lehman', 'herbert.lehman@student.edu', 'hashed_pw_1'),
    ('Aaron', 'Judge', 'aaron.judge@student.edu', 'hashed_pw_2');

INSERT INTO teachers (first_name, last_name, email, staff_password) VALUES
    ('Alan', 'Turing', 'alan.turing@school.edu', 'hashed_pw_3'),
    ('Bob', 'Smith', 'bob.smith@school.edu', 'hashed_pw_4');

INSERT INTO courses (course_name, teacher_id) VALUES
    ('Math 101', 1),
    ('Intro to Computer Science', 2);

INSERT INTO enrollments (student_id, course_id) VALUES
    (1, 1), (2, 2);

INSERT INTO grades (student_id, course_id, grade) VALUES
    (1, 1, 89.5), (2, 2, 94.0);

INSERT INTO attendance (student_id, course_id, date, status) VALUES
    (1, 1, '2025-03-10', 'Present'),
    (2, 2, '2025-03-10', 'Absent'),
    (1, 1, '2025-03-11', 'Absent'),
    (2, 2, '2025-03-11', 'Present');
```

Step 4: SQL Join Query Example

We used SQL joins to retrieve student grades and course details for reporting purposes.To demonstrate the relationships and how data is retrieved through joins, the following queries are used:

Students and Their Enrolled Courses

```
SELECT s.first_name AS student_first_name, s.last_name AS student_last_name,c.course_name
FROM enrollments e
JOIN students s ON e.student_id = s.id
JOIN courses c ON e.course_id = c.course_id;
```

List All Courses and Their Assigned Teachers

```
SELECT  c.course_name, t.first_name AS teacher_first_name, t.last_name AS teacher_last_name
FROM courses c
JOIN teachers t ON c.teacher_id = t.id;
```

Show All Grades With Student and Course Info

```
SELECT
   s.first_name AS student_first_name,
   s.last_name AS student_last_name,
   c.course_name,
   g.grade
FROM grades g
JOIN students s ON g.student_id = s.id
JOIN courses c ON g.course_id = c.course_id;
```

View Attendance Records by Date

```
SELECT s.first_name, s.last_name, c.course_name, a.date, a.status
FROM attendance a
JOIN students s ON a.student_id = s.id
JOIN courses c ON a.course_id = c.course_id
ORDER BY a.date;
```

View Attendance for a Specific Course and Date

```
SELECT  s.first_name, s.last_name, a.date, a.status
FROM attendance a
JOIN students s ON a.student_id = s.id
WHERE a.course_id = 1 AND a.date = '2025-03-10';
```

Count Present and Absent Students Per Day Per Course

```
SELECT  c.course_name, a.date, a.status,
COUNT(*) AS count
FROM attendance a
JOIN courses c ON a.course_id = c.course_id
GROUP BY c.course_name, a.date, a.status
ORDER BY a.date;
```

Find Students with Grades Above 90

```
SELECT  s.first_name, s.last_name, c.course_name, g.grade
FROM grades g
JOIN students s ON g.student_id = s.id
JOIN courses c ON g.course_id = c.course_id
WHERE g.grade > 90;
```

Step 5: Transaction Example Using PHP

In the PHP portion of the application, we used transactions to ensure that operations like inserting a new grade are executed safely and can be rolled back in case of an error.

“Insert the section of PHP that utilizes my sql and explain”

Logic Behind Key Features:

Data Integrity: Enforced with primary and foreign keys, ENUMs, and CHECK constraints.

Security: Passwords are designed to be hashed before storage, and queries are meant to be executed via prepared statements to prevent SQL injection.

Joins: Multiple-table joins are used to generate reports and views of student performance and attendance.

Scalability: The structure is flexible enough to allow additional features like login systems, filtering by teacher, or semester tracking.

Database Design:

Schema Overview
The database consists of the following six core tables:

students – Stores student information including first name, last name, email, and a hashed password.

teachers – Stores teacher credentials and login information.

courses – Represents the courses being taught and links each course to its respective teacher.

enrollments – A bridge table that establishes a many-to-many relationship between students and courses.

grades – Stores student grades for each course.

attendance – Records student attendance status (Present or Absent) for each course on a specific date.

Each table includes relevant primary and foreign keys to enforce referential integrity. Password fields are defined with VARCHAR(255) to store securely hashed passwords. ENUM and CHECK constraints are used where appropriate to validate input and maintain data consistency.

Relationships Between Tables
One-to-Many: Each teacher can teach multiple courses (courses.teacher_id → teachers.id).

Many-to-Many: Students can enroll in multiple courses and each course can have many students. This relationship is implemented through the enrollments table (student_id, course_id).

Grades Assignment: The grades table references both students and courses, allowing each student to receive a grade per course.

Attendance Tracking: The attendance table references both students and courses, allowing attendance records to be linked to specific dates and students.

Use of Joins for Data Retrieval
Joins are essential for retrieving meaningful data from multiple related tables in a relational database. In this project, SQL joins were used to connect student information with their courses, grades, attendance records, and instructor details. Below are several queries that demonstrate how joins are used in the Student Grading and Attendance System:

1. Retrieve Students and Their Enrolled Courses

```
SELECT  s.first_name AS student_first_name, s.last_name AS student_last_name, c.course_name
FROM enrollments e
JOIN students s ON e.student_id = s.id
JOIN courses c ON e.course_id = c.course_id;
```

This query joins the enrollments, students, and courses tables to display which students are enrolled in which courses.

2. Retrieve All Courses and Their Assigned Teachers

```
SELECT c.course_name, t.first_name AS teacher_first_name, t.last_name AS teacher_last_name
FROM courses c
JOIN teachers t ON c.teacher_id = t.id;
```

This join connects courses to teachers and is used to display the teacher assigned to each course.

3. Display All Grades by Student and Course

```
SELECT s.first_name AS student_first_name, s.last_name AS student_last_name, c.course_name, g.grade
FROM grades g
JOIN students s ON g.student_id = s.id
JOIN courses c ON g.course_id = c.course_id;
```

This query combines grades, students, and courses to show student performance in each course.

4. Show All Attendance Records with Course and Student Info

```
SELECT  s.first_name, s.last_name, c.course_name, a.date, a.status
FROM attendance a
JOIN students s ON a.student_id = s.id
JOIN courses c ON a.course_id = c.course_id
ORDER BY a.date;
```

This join pulls together attendance, students, and courses to display attendance status for each student on each date.

5. View Attendance for a Specific Course on a Specific Date

```
SELECT  s.first_name, s.last_name, a.date, a.status
FROM attendance a
JOIN students s ON a.student_id = s.id
WHERE a.course_id = 1 AND a.date = '2025-03-10';
```

This query is useful for checking attendance for a single course on a specific day.

6. Count Present and Absent Students by Date and Course

```
SELECT  c.course_name, a.date, a.status,
COUNT(*) AS count
FROM attendance a
JOIN courses c ON a.course_id = c.course_id
GROUP BY c.course_name, a.date, a.status
ORDER BY a.date;
```

This grouped query provides summary counts of who was present or absent per course and date.

7. Find Students Who Scored Above 90 in Any Course

```
SELECT s.first_name, s.last_name, c.course_name, g.grade
FROM grades g
JOIN students s ON g.student_id = s.id
JOIN courses c ON g.course_id = c.course_id
WHERE g.grade > 90;
```

This join is used to identify high-performing students across all courses.

These join queries allow the system to pull meaningful insights from normalized data tables while preserving data integrity and scalability.

Technical Details: HTML CSS pHp - Jessi Quezada

1. Essential tools for the website
   Although the aesthetic of the website is important the key things I needed for the website was creating the form and the navigation bar to help the user navigate through out the website

```
</head>
<body>
        <div class="container ">
        <div class="box-form-box">
        <header> Login </header>
        <form action="login.php" method="post">


        <div  class="field input">
        <label for="email"> Email</label>
        <input type="text" name="email" id="email" required>
        </div>

        <div  class="field input">
        <label for="staff_password"> Password</label>
        <input type="password" name="password" id="staff_password" required>
        </div>
        <input type="submit"class="loginbutton" name="login_user" value="Login">
        <div  class="field">

        </div>
</form>
</div>
</div>
```

CSS:

```
.loginbutton {
        height: 35px;
        background: #ae82ff;
        border: none;
        border-radius: 5px;
```

```
        color: white;
        font-size: 16px;
        font-weight: 600;
        font-family: "Open Sans", sans-serif;
        cursor: pointer;
        transition: all .3s;
        width: 100%;
        margin-top: 10px;
        padding: 0px 10px;
}


}
.loginbutton:hover {
        background: #7e42f5;
}



input{

        width: 100%;
        padding: 12px;
        color: #300;
        margin-bottom: 20px;
}
```

This block of code represents a login form that allows users (teachers in this case) to securely enter their credentials and be authenticated. The <form> uses the POST method to send the entered email and password to login.php, where the backend handles login validation. Each input field is styled for consistency and user-friendliness, including proper spacing, padding, and visual feedback.

The .loginbutton CSS styles the login button with a modern look, including a hover effect that changes the background color to show users the interaction between the mouse over the button, the same css command is applied to log out button as well. The input styling ensures all form fields stretch to full width, have consistent padding, and look clean and readable, enhancing overall user experience.

Throughout all the pages except login I had this same line of code ready and it was for the teacher can have an easy navigation experience.

```
        <nav class = "nav">
```

```
<div class = "navdiv">
<div class = "logo"></div>
        <u1>
<li><a href="groupprojecthomepage.php">Home</a></li>
<li><a href="Add_Students.php">Add Students</a></li>
<li><a href="Attendance.php">Attendance</a></li>
<li><a href="Grades.php">Grades</a></li>
<a href="groupproject.php"> <button class = "logoutbutton">Log out</button> </a>
</u1>
</div>
</nav>
```

The <nav> section builds a navigation bar using the nav, navdiv, and logo classes. The links inside the <ul>, let the teacher access different pages like Home, Add Students, Attendance, and Grades. There's also a "Log out" button styled to look consistent with the rest of the menu. The nav and navdiv CSS apply layout and styling, like a background color, making sure it is wide enough to fit the text, and having space in between so the text links are on the right side of the document while the logo is on the left side, giving it a clean look.

CSS Portion:

```
table-container {
  width: 100%;
  overflow-x: auto;
  display: flex;
align-items: center;
justify-content: center;
min-height: 90vh;
width: 100%;

}

.grades_table {
  width: max-content;  /* Makes table expand as wide as it needs */
  border-collapse: collapse;
}

.grades_table th,
.grades_table td {

  padding: 10px;
  border: 1px solid #ccc;
}

.box-form-box-table{
```

```
    background: #fdfdfd;
    display: flex;
    flex-direction: column;
    padding: 25px 25px;
    border-radius: 20px;
    box-shadow: 0 0 128px 0 rgba(0, 0, 0, 0.1),
    0 32px 64px -48px rgba(0,0,0,0.5);
    margin: 0px 10px;

    width: fit-content;
    max-width: 100%;
    overflow: auto;


    }
```

This css portion contains the table of the grades side and a similar style was applied to the attendance table as well the table container is just a wrapper around the two tables making it center on the screen, the box-form-box-table customizes the look of the table, grdaes_table make sure the class is formatted correctly and that the cells are separated using border_collaspe and once again the look of the table was customized using padding and borders.(grades_table was used for both attendance and grades)

PHP section

First I needed to create config.php, a php file that connects the data received from the mysql script from myphpadmin and connects it to the code

```
<?php

$servername = "localhost";
$username = "root";
$password = "";
$db_name = "student_system";
$conn = new mysqli($servername, $username, $password, $db_name, 4306);

if ($conn->connect_error) {
    die("connection failed:" .$conn->connect_error);

}

// Use to test if the connection worked
echo "";
```

?>

This PHP code connects your website to a MySQL database called student_system using the mysqli (MySQL Improved) extension. It sets up the database connection by specifying the server name username, no password, and the database name.

The last number isn't necessary as I have to since I use a different port number in order to access mysql through xampp but if you're under the default port number you can leave the last part empty.

It then checks if the connection was successful with $conn->connect_error. If the connection fails, it stops execution and prints an error message. If the connection works, it shows it echos nothing but if I wanted to see if it works then I fill it with "connection works" then I'll be able to see it on screen, but since I already seen it on the screen I decided to delete the dialouge so I make sure random words don't appear on the screen.

. 2. **Right after I create login.php - which I mainly thought was going to be used for just hte login page it became more efficient to have most of my php code in this document as it doesn't clog up too much for my html codes such as groupproject.php, grades.php etc. Getting started I start writing php for the login section:**

```
include("config.php");
session_start();

if(isset($_POST['login_user'])){
$email = $_POST['email'];
$staff_password = $_POST['password'];
```

One of the important commands used here is the POST command, which is an HTTP method used to send data securely from a form to the server, and in PHP, this data is accessed using the $_POST superglobal array.

This part of the data uses the post command to receive the email and staff password entered by the user

```
$sql = "SELECT * FROM teachers where email = '$email' and staff_password =
'$staff_password'";
$result = mysqli_query($conn,$sql);
$row = mysqli_fetch_array($result, MYSQLI_ASSOC);
$count = mysqli_num_rows($result);
if($count==1){
header("Location:groupprojecthomepage.php");
```

In this section SQL query is run to check is there is a teach with that exact email and password and then mysqli_query is used to fetch the matching row , the amount of

results returned is then countred and if it is exactly one then the user is directed to the home page

```
}
else{
echo '<script>
window.location.href = "groupproject.php"
alert("Login failed. Invalid username or password, please try again")
</script>';
}

}
```

But if the info is incorrect then the user gets an error message and is directed back to the login screen.

For the add students section, I was looking to directly affect the grades page, but I knew I had to use a JOIN tables command, and I knew I had to use insert command multiple times to make sure key parts of the grades table were entered

```
// ADD STUDENTS key notes to make sure it can we need to insert into the students,enrollment and grades table to make sure it appears on the grades table
$errors = array();

if ($_SERVER["REQUEST_METHOD"] == "POST") {
// All the entites we need filled in order to make sure it gets inserted into the three tables without errors
$first_name = $_POST["first_name"];
$last_name = $_POST["last_name"];
$email = $_POST["email"];
$password = $_POST["password_"];
$course_id = $_POST["course_id"];
$grade = $_POST["grade"];

// Insert into the student table
$sql = "INSERT INTO students (first_name, last_name, email, password_)
VALUES ('$first_name', '$last_name', '$email', '$password')";

if ($conn->query($sql) === TRUE) {
// Get new student ID
$student_id = $conn->insert_id;

// Insert into the enrollment table
$conn->query("INSERT INTO enrollments (student_id, course_id)
VALUES ('$student_id', '$course_id')");

// Now inserting into the grades table
```

```
$conn->query("INSERT INTO grades (student_id, course_id, grade)
VALUES ('$student_id', '$course_id', '$grade')");

/// Redirect to homepage
header("Location: groupprojecthomepage.php");
exit();
} else {
echo "Error: " . $conn->error;
}

$conn->close();
}
```

Using similar coding from the previous section, we are using a bit more completed code and as well JOIN Two tables together to satisfy the grades

```
// Joining the tables together for the grades page
// Spaceing them out so it doesn't look too crowded and clean
$sql = "SELECT s.id,s.first_name,
s.last_name,
s.email,
c.course_name,
g.grade
FROM students s
JOIN enrollments e ON s.id = e.student_id
JOIN courses c ON e.course_id = c.course_id
LEFT JOIN grades g ON s.id = g.student_id AND c.course_id = g.course_id";

$result = mysqli_query($conn, $sql);
```

I start with the student table giving an alias s to make it easier to type, then i join two tables together, I join enrollments onto student to see which courses each student is enrolled in, then it joins the courses table to display course details like the name. A LEFT JOIN is used with the grades table to also show each student's grade for a course. Because it's a LEFT JOIN, students without grades will still appear, which helps identify missing grades.

```
// Output the table rows
if (mysqli_num_rows($result) > 0) {
while ($row = mysqli_fetch_assoc($result)) {
echo "<tr>
<td>{$row['id']}</td>
<td>{$row['first_name']}</td>
```

```
<td>{$row['last_name']}</td>
<td>{$row['email']}</td>
<td>" . ($row['course_name'] ?? 'N/A') . "</td>
<td>" . ($row['grade'] ?? 'N/A') . "</td>
</tr>";
}
} else {
   echo "<tr><td colspan='6'>No grades found</td></tr>";
}
```

To display the table I output just the studentid, their name, email, coursename and then the grade
It first checks if the SQL query returned any rows using mysqli_num_rows($result) > 0. If so, it loops through each row using mysqli_fetch_assoc($result) and prints out a <tr> table row with columns for the student's ID, name, email, course name, and grade.

The expressions ($row['course_name'] ?? 'N/A') and ($row['grade'] ?? 'N/A') use the null operator (??) detailing that if that value is NULL or missing, it will display 'N/A' instead. This was done in case no grade or course was filled out in the add students application, but if u no results are found, it prints a single row saying "No grades found."

// ATTENDANCE DATA FORM - moved to attendance.php
Attendance php commands were moved to attendance.php due to avoid the data getting mixed up with the grades table, it would cause up a lot of confusing so it was best to avoid, but the php command is extremely similar to the ones used for grades.