

ATTENDANCE MANAGEMENT SYSTEM

A DBMS PROJECT REPORT

Submitted by

M.V.S.HARSHITH [RA2311003011196]

V.VENNELA REDDY [RA2311003011203]

D.S.ROHITH [RA2311003011219]

Under the Guidance of

Dr.Baranidharan B

(Professor, Department of Computing Technology)

in partial fulfillment of the requirements for the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE ENGINEERING



DEPARTMENT OF COMPUTING TECHNOLOGIES,
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR- 603 203

MAY 2025



Department of Computing technologies
SRM Institute of Science & Technology
Own Work* Declaration Form

This sheet must be filled in (each box ticked to show that the condition has been met). It must be signed and dated along with your student registration number and included with all assignments you submit – work will not be marked unless this is done.

To be completed by the student for all assessments

Degree/ Course : B.TECH

Student Name : M.V.S HARSHITH , V.VENNELA REDDY , D.S.ROHITH

Registration Number : RA2311003011196 , RA2311003011203 , RA2311003011219

Title of Work : ATTENDANCE MANAGEMENT SYSTEM

I / We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is my / our own except where indicated, and that I / We have met the following conditions:

- Clearly referenced / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:

I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.

M.V.S Harshith
RA2311003011196
M.V.S Harshith

V.Vennela Reddy
RA2311003011203
V.Vennela Reddy

D.S. Rohith
RA2311003011219
Rohith D.S

If you are working in a group, please write your registration numbers and sign with the date for every student in your group.



SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that 21CSC205P – Database Management System Project report titled “ATTENDANCE MANAGEMENT SYSTEM” is the bonafide work of “M.V.S.HARSHITH [RA2311003011196], V.VENNELA REDDY[RA2311003011203], D.S.ROHITH[RA2311003011219]” who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. Baranidharan B
PROFESSOR

DEPARTMENT OF
COMPUTING TECHNOLOGIES

SIGNATURE

Dr. G. Niranjana
PROFESSOR & HEAD

DEPARTMENT OF
COMPUTING TECHNOLOGIES



ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to **Dr. Leenus Jesu Martin M**, Dean-CET, SRM Institute of Science and Technology, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor and Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We encompass our sincere thanks to, **Dr. M. Pushpalatha**, Professor and Associate Chairperson, School of Computing and **Dr. C. Lakshmi**, Professor and Associate Chairperson, School of Computing, SRM Institute of Science and Technology, for their invaluable support.

We are incredibly grateful to our Head of the Department, **Dr. G. Niranjana**, Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our DBMS Coordinators, **Dr. Muthu Kumaran A M J** and **Dr. Jayapradha J** Department of Computing Technologies, and our Audit Professor **Dr. Malathy C**, Department of Networking And Communications, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisors, **Dr. Deepajothi S**, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our faculty, **Dr. Baranidharan B**, Department of Computing Technologies, S.R.M Institute of Science and Technology, for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank all the staff and students of Computing Technologies, School of Computing, S.R.M Institute of Science and Technology, for their help during our project. Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support and encouragement

M.V.S.HARSHITH [RA2311003011196]

V.VENNELA REDDY[RA2311003011203]

D.S.ROHITH [RA2311003011219]

ABSTRACT

The QR Code-Based Attendance Management System is an innovative web-based platform designed to simplify attendance tracking in educational institutions. It aims to replace traditional methods like manual roll calls with a digital solution, enhancing accuracy, reducing administrative workload, and ensuring secure attendance recording. By leveraging QR code technology, the system offers a seamless experience for both students and faculty, minimizing human error.

The system supports multi-role authentication, allowing students, faculty, and administrators to access features relevant to their roles. Faculty can create and manage classes, initiate attendance sessions with temporary QR codes valid for 5 minutes, and efficiently monitor attendance records. These time-bound QR codes reduce the chances of proxy attendance. Students can join classes by scanning QR codes and instantly mark their attendance while tracking their attendance history.

An admin dashboard provides comprehensive system management, including user role assignments and class management, ensuring a streamlined and transparent attendance process. The platform's backend uses Node.js and Express, with MySQL and Sequelize ORM for database management, and Handlebars for the front-end templating engine. This tech stack ensures high performance, maintainability, and a user-friendly interface.

Key features include time-bound QR codes and simplified session creation, contributing to enhanced security and user experience. The relational database structure consists of tables for users, classes, attendance sessions, student-class associations, and attendance records. Recent updates include instant QR code generation and improved database relationships, enhancing reliability and usability.

The system significantly reduces manual effort, minimizing errors and promoting an organized attendance management process. The secure QR code generation mechanism also ensures that attendance cannot be falsified, maintaining data integrity. Additionally, the system's design prioritizes ease of navigation, making it accessible for both students and faculty.

Future enhancements may include mobile app integration, real-time reporting, advanced data analytics, and improved role management to meet evolving institutional needs. The system aligns with the trend of technology integration in education, offering a reliable and efficient solution for modern attendance tracking.

TABLE OF CONTENTS

ABSTRACT		iv
TABLE OF CONTENTS		v
LIST OF FIGURES		vi
LIST OF TABLES[if required]		vii
ABBREVIATIONS		viii
CHAPTER NO.	TITLE	PAGE NO.
1. INTRODUCTION		
1.1	General (Introduction to Project)	11
1.2	Project Inspiration	12
1.3	Problem Context	13
1.4	Purpose and Vision	14
1.5	User Specifications	14-15
1.6	Challenges	16-17
1.7	Future Scope	17
2. 1B - EXPERIMENT		
2.1	Problem Statement	18-19
2.2	Data Requirements	20-21
2.3	Functional Requirements	21-22
2.4	ER Diagram	22-23
2.5	Entites and their Attributes	23-24
2.6	Relationships between Entites	24-25
2.7	Entity and Relationship Identification	25-27
3. 2B - EXPERIMENT		
3.1	Entity-Relationship Schema	28
3.2	Database Schemas	28-30
3.3	List of Tables	30-31

4. 3B - EXPERIMENT

4.1	Basic SQL Queries	32
4.2	Stored Procedures	32-34
4.3	Triggers	34-35
4.4	Cursors	35-36
4.5	Functions	37

5. 4B - EXPERIMENT

5.1	Applying 1NF	38
5.2	Applying 2NF	38
5.3	Applying 3NF	39
5.4	Applying BCNF	39

6. Implementation of Concurrency Control and Recovery Mechanisms

6.1	Overview	42-43
6.2	Critical Acid scenarios in this system	43
6.3	Concurrency Control	43-44
6.4	Recovery Mechanisms	44-45
6.5	Transaction Control Mechanism	45-46

7. Project Source Code

47-93

8. Analysis and Findings

94-97

(Screen shots of the implementation with front end)

9. Conclusion and References

9.1	Conclusion	98-99
9.2	References	99-100

LIST OF FIGURES

CHAPTER NO	TITLE	PAGE NO.
2	ER Diagram.....	22
3	Relational Schema Design.....	27
4	Stored Procedure.....	33
4	Trigger.....	34
4	Cursor.....	35
4	Function.....	36
8	Login Page.....	94
8	Student Dashboard.....	94
8	Faculty Dashboard.....	94
8	Join a Class.....	95
8	Student Classes.....	95
8	Student Attendance.....	95
8	QR Code Scanning.....	96
8	Attendance Submission.....	96
8	Faculty Creating New Class.....	97

LIST OF TABLES

CHAPTER NO	TITLE	PAGE NO.
3	Tables List.....	30
3	Users Table.....	31
3	Classes Table.....	31
3	Sessions Table.....	31
3	Student Classes Table.....	31
3	Attendances Table.....	31
5	Users Table in Normalized form.....	39
5	Classes Table in Normalized form.....	39
5	Sessions Table in Normalized form.....	39
5	Student Classes Table in Normalized form.....	39
5	Attendances Table in Normalized form.....	39

ABBREVIATIONS

1. **DBMS – Data Base Management System**
2. **SQL – Structured Query Language**
3. **ER - Entity Relationship**
4. **PK - Primary Key**
5. **FK - Foreign Key**
6. **HTML – Hyper Text Markup Language**
7. **CSS – Cascading Style Sheets**
8. **BCNF – Boyce Code Normal Form**

CHAPTER-1

INTRODUCTION

1.1 GENERAL INTRODUCTION

In the digital era, the management of attendance data has become a critical function for educational institutions, businesses, and organizations worldwide. Accurate attendance tracking is essential not only for administrative efficiency but also for compliance, resource planning, and performance evaluation. Traditional methods of attendance management, such as manual registers and paper-based logs, are increasingly seen as outdated, error-prone, and inefficient. The need for a robust, automated, and user-friendly system has never been greater.

The Attendance Management System is conceptualized as a comprehensive solution to these challenges. This system is designed to streamline the process of recording, managing, and analyzing attendance data, offering a centralized platform that is both powerful and accessible. The Attendance Management System leverages modern software technologies to provide real-time data access, automated reporting, and secure data storage, all within an intuitive and responsive interface. The core objective of the Attendance Management System is to eliminate the inefficiencies and inaccuracies associated with manual attendance tracking.

By automating routine tasks and providing advanced features such as biometric integration, mobile access, and data analytics, the system enables organizations to focus on their primary goals rather than administrative burdens. Furthermore, the system is built with scalability and flexibility in mind, ensuring that it can adapt to the evolving needs of diverse user groups, from small schools to large enterprises. Security and data privacy are at the forefront of the Attendance Management System's design. With sensitive personal data being handled daily, the system incorporates robust authentication, authorization, and encryption protocols to safeguard information against unauthorized access and breaches. Regular security updates and compliance with relevant regulations further enhance the trustworthiness of the platform.

The development of the Attendance Management System follows a user-centric approach, with continuous feedback from stakeholders shaping its features and functionalities. This iterative process ensures that the system remains aligned with user expectations and industry best practices. In summary, the Attendance Management System represents a significant advancement in attendance tracking, combining technological innovation with practical usability to deliver a solution that meets the demands of modern organizations.

In addition to its core features, the Attendance Management System is designed to support integration with existing organizational infrastructure, allowing seamless data sharing and synchronization. The system also includes customizable reporting tools, enabling users to generate attendance summaries and trend analyses tailored to specific needs. By offering multi-device compatibility, the system ensures accessibility from desktops, tablets, and smartphones, making attendance management more flexible and convenient. Furthermore, the platform supports real-time notifications, enabling instant updates for stakeholders regarding attendance status and irregularities. This holistic approach makes the system not only efficient but also adaptable to a wide range of operational scenarios.

1.2 PROJECT INSPIRATION

The inspiration for the Attendance Management System arose from observing the persistent challenges faced by institutions in managing attendance data. In educational settings, for example, teachers often spend valuable instructional time taking attendance manually, leading to lost productivity and potential errors. Similarly, in corporate environments, tracking employee attendance for payroll and compliance purposes can be a cumbersome task, especially when relying on outdated methods.

Personal experiences within academic and professional contexts highlighted the limitations of existing systems. Instances of misplaced attendance sheets, unauthorized alterations, and difficulties in generating timely reports underscored the need for a more reliable and efficient solution. These challenges were particularly acute in organizations with large populations, where manual processes became increasingly unmanageable.

Technological advancements, such as the proliferation of smartphones and biometric devices, presented new opportunities for innovation in attendance management. The project team recognized the potential to harness these technologies to create a system that not only automates attendance tracking but also enhances accuracy and accessibility. The vision was to develop a platform that could seamlessly integrate with existing infrastructure while providing advanced features tailored to the specific needs of different user groups.

The desire to promote transparency and accountability also played a significant role in inspiring the project. By providing real-time access to attendance data and generating comprehensive reports, the Attendance Management System empowers administrators, teachers, and employees to monitor attendance patterns and address issues proactively.

Another source of inspiration was the growing emphasis on data-driven decision-making. Accurate attendance data can provide valuable insights into student engagement, employee productivity, and organizational efficiency. By enabling easy access to such data, the Attendance Management System supports informed decision-making at all levels.

In summary, the Attendance Management System is inspired by the need to modernize attendance tracking, enhance data accuracy, and empower users through technology. The project aims to transform attendance management from a routine administrative task into a strategic asset for organizations.

The project team also drew inspiration from the increasing demand for flexible and remote work arrangements, where traditional attendance methods prove inadequate. The Attendance Management System addresses this by incorporating features such as remote check-ins, location verification, and time-stamped entries, ensuring that attendance tracking remains accurate regardless of physical presence. Additionally, the system's modular architecture allows for easy customization and scalability, accommodating the unique requirements of various industries. By fostering a culture of accountability and data transparency, the system not only streamlines administrative tasks but also supports performance management and compliance initiatives, making it a valuable tool for modern organizations.

1.3 PROJECT CONTEXT

Attendance management is a fundamental aspect of organizational operations, yet traditional methods like paper registers and manual entries are fraught with challenges. These methods are prone to errors, manipulation, and data loss, leading to payroll discrepancies, compliance violations, and diminished trust among stakeholders. Moreover, manual processes are time-consuming, diverting attention from core activities such as teaching or project work. In large organizations, the volume of data makes manual tracking impractical, causing delays in report generation and decision-making.

Key Issues with Manual Attendance Systems

- **Inefficiency:** Manual attendance tracking consumes valuable time and resources, reducing overall productivity.
- **Data Integrity Concerns:** Records are susceptible to tampering, leading to inaccuracies in payroll and academic records.
- **Lack of Real-Time Access:** Delayed data entry hampers timely responses to absenteeism and policy violations.
- **Security and Privacy Risks:** Personal information is vulnerable to unauthorized access due to inadequate security measures.
- **Scalability Issues:** As organizations grow, manual systems struggle to accommodate increasing complexity, resulting in fragmented data and operational inefficiencies.

Introducing the Attendance Management System

To address these challenges, the Attendance Management System offers a unified, automated, and secure platform for tracking attendance. By eliminating manual processes, it enhances data integrity and supports scalability, transforming attendance management into a streamlined and reliable function.

Benefits of the Automated System

- **Operational Efficiency:** Automates routine tasks, freeing up time for strategic activities.
- **Enhanced Accuracy:** Reduces errors associated with manual data entry, ensuring reliable records.
- **Real-Time Monitoring:** Provides immediate access to attendance data, enabling prompt decision-making.
- **Improved Security:** Protects sensitive information through robust security measures, ensuring compliance with data protection regulations.
- **Scalability:** Designed to accommodate organizational growth, supporting additional users, locations, and system integrations.

Implementing an automated Attendance Management System addresses the inherent flaws of traditional methods, offering a more efficient, accurate, and secure approach to tracking attendance. This transformation not only enhances operational efficiency but also aligns with modern organizational needs for accuracy, security, and scalability.

1.4 PURPOSE AND VISION

The primary purpose of the Attendance Management System is to provide organizations with a reliable, efficient, and user-friendly tool for managing attendance data. The system is intended to simplify the process of recording, monitoring, and analyzing attendance, thereby freeing up valuable time and resources for more strategic activities.

The vision for the Attendance Management System is to become the standard for attendance tracking across industries. The system aspires to set new benchmarks in terms of usability, security, and functionality. By leveraging the latest technologies and adhering to best practices in software design, the Attendance Management System seeks to deliver a solution that is both powerful and accessible.

A key aspect of the system's vision is inclusivity. The Attendance Management System is designed to cater to the needs of a diverse user base, including teachers, students, employees, managers, and administrators. The interface is intuitive and customizable, ensuring that users with varying levels of technical expertise can navigate the system with ease.

Automation is at the heart of the system's purpose. Routine tasks, such as attendance recording, report generation, and notifications, are automated to minimize manual intervention and reduce the risk of errors. Advanced features, such as biometric authentication and mobile access, further enhance the system's efficiency and convenience.

Security and data privacy are central to the system's vision. The Attendance Management System implements robust security protocols, including multi-factor authentication, role-based access control, and data encryption, to protect sensitive information. Compliance with relevant regulations, such as the General Data Protection Regulation (GDPR), is a core design principle.

Scalability and integration are also key components of the system's vision. The Attendance Management System is architected to support organizations of all sizes, from small schools to multinational corporations. The system can be easily integrated with other software platforms, such as payroll systems, learning management systems, and HR software, ensuring seamless data flow and operational efficiency.

In conclusion, the purpose and vision of the Attendance Management System are to revolutionize attendance management by providing a secure, automated, and user-friendly solution that meets the evolving needs of modern organizations.

1.5 USER SPECIFICATIONS

The success of the Attendance Management System hinges on its ability to meet the diverse needs of its users. A detailed understanding of user requirements has informed every aspect of the system's design and development. The primary user groups and their specifications are outlined below:

❖ Administrators

Administrators are responsible for configuring the system, managing users, and overseeing attendance policies. Their requirements include:

- Advanced configuration options for attendance rules and schedules
- User management features, including role assignment and permissions
- Access to comprehensive reports and analytics
- Tools for monitoring system health and performance

- Secure backup and recovery options

❖ **Teachers and Supervisors**

These users are responsible for recording and monitoring attendance. Their requirements include:

- Simple and intuitive interfaces for marking attendance
- Real-time access to attendance records
- Ability to generate and print attendance reports
- Notifications for irregular attendance patterns
- Integration with academic or project schedules

❖ **Students and Employees**

The primary concern for these users is transparency and access to their own attendance records. Their requirements include:

- Secure login and access to personal attendance data
- Notifications for absences or late arrivals
- Ability to request corrections or provide explanations for absences
- Access to attendance summaries and trends

❖ **IT Support Staff**

These users are responsible for maintaining the system and troubleshooting issues. Their requirements include:

- Diagnostic and logging tools for system monitoring
- Detailed documentation and user guides
- Tools for data migration and integration with other systems
- Support for regular updates and security patches

❖ **External Stakeholders**

In some cases, external auditors, regulatory bodies, or partners may require access to attendance data. Their requirements include:

- Secure, read-only access to specific reports
- Detailed audit trails and compliance features
- Customizable reporting options
- Key Features Based on User Specifications:
- Responsive web and mobile interfaces
- Multi-language support for diverse user bases
- Customizable dashboards and notifications
- Flexible reporting and data export options
- Robust security and privacy controls
- Integration with third-party applications

By addressing these user specifications, the Attendance Management System aims to deliver a solution that is both functional and user-friendly, promoting widespread adoption and satisfaction.

1.6 CHALLENGES

The development and deployment of the Attendance Management System present several challenges, each requiring careful consideration and strategic planning.

❖ Technical Complexity

Designing a system that balances advanced functionality with ease of use is inherently challenging. The need to support features such as biometric integration, real-time data synchronization, and cross-platform compatibility adds layers of complexity to the development process. Ensuring optimal performance and reliability requires rigorous testing and optimization.

❖ Security and Privacy

With sensitive personal data at stake, security is paramount. Implementing robust authentication, authorization, and encryption mechanisms is complex and requires ongoing vigilance. The system must also comply with data protection regulations, which may vary across jurisdictions.

❖ User Adoption

Transitioning users from manual or legacy systems to the Attendance Management System can be met with resistance. Effective training, clear communication of benefits, and a seamless migration process are essential to encourage adoption and minimize disruption.

❖ Integration with Existing Systems

Many organizations rely on a variety of software platforms for HR, payroll, and academic management. Ensuring seamless integration with these systems is critical for data consistency and operational efficiency. This may involve developing custom APIs, data migration tools, and synchronization mechanisms.

❖ Resource Constraints

Developing a comprehensive attendance management system requires significant investment in terms of time, expertise, and financial resources. Balancing project scope with available resources is a constant challenge, necessitating careful prioritization and project management.

❖ Continuous Improvement

The technological landscape is constantly evolving, with new advancements and emerging threats. Maintaining the relevance and effectiveness of the Attendance Management System requires a commitment to continuous improvement, regular updates, and the incorporation of user feedback.

❖ Testing and Quality Assurance

Ensuring the reliability and stability of the system demands rigorous testing and quality assurance processes. This includes unit testing, integration testing, performance testing, and user acceptance testing. Identifying and resolving bugs and vulnerabilities is an ongoing process.

❖ **Documentation and Support**

Comprehensive documentation and user support are essential for successful adoption and usage. Developing clear, concise, and accessible documentation, as well as providing timely support, are ongoing challenges that require dedicated resources.

❖ **Scalability**

As organizations grow, the system must scale accordingly. Designing the Attendance Management System to handle increasing data volumes, user loads, and transaction rates without compromising performance is a significant technical challenge.

❖ **Customization and Flexibility**

Users have diverse needs and preferences, necessitating a system that is customizable and flexible. Balancing standardization with the ability to tailor the system to specific requirements is a complex endeavor.

1.7 FUTURE SCOPE

The implementation of the Attendance Management System has the potential to deliver significant benefits across a wide range of organizations. By automating attendance tracking, the system reduces administrative overhead, minimizes errors, and enhances data accuracy. Real-time access to attendance data enables timely interventions, whether it be addressing absenteeism in schools or ensuring compliance in workplaces.

The system's analytics and reporting capabilities provide valuable insights into attendance patterns, supporting data-driven decision-making. For example, schools can identify students at risk of chronic absenteeism and implement targeted interventions, while businesses can optimize workforce management and resource allocation.

Looking to the future, the Attendance Management System is poised for continued evolution and expansion. Emerging technologies, such as artificial intelligence and machine learning, offer opportunities to further enhance the system's capabilities. Predictive analytics could be used to forecast attendance trends, while natural language processing could enable voice-based attendance marking.

Integration with wearable devices and Internet of Things (IoT) sensors could provide even more granular and automated attendance tracking. For instance, smart ID cards or facial recognition systems could automatically record attendance as individuals enter a classroom or workplace.

The system's modular architecture allows for easy customization and extension, ensuring that it can adapt to changing organizational needs. As remote work and hybrid learning models become more prevalent, the Attendance Management System can be enhanced to support virtual attendance tracking and integration with online collaboration platforms.

In conclusion, the Attendance Management System is not only a solution to current attendance management challenges but also a platform for future innovation. By embracing technological advancements and maintaining a focus on user needs, the system is well-positioned to remain at the forefront of attendance management for years to come.

CHAPTER-2

1B EXPERIMENT

2.1 PROBLEM STATEMENT

Attendance management is a fundamental requirement across educational institutions, corporate organizations, and various other settings where regular participation or presence is essential. Traditionally, attendance has been recorded manually using paper registers, spreadsheets, or simple logbooks. While these methods are straightforward, they are fraught with inefficiencies, inaccuracies, and vulnerabilities that can significantly impact operational effectiveness.

Challenges of Manual Attendance Systems

- **Inefficiency and Human Error**

Manual attendance tracking is inherently time-consuming. Staff members must dedicate significant portions of their day to recording, verifying, and compiling attendance data. This process diverts attention from core responsibilities and can lead to decreased productivity. Moreover, human error is an ever-present risk. Mistakes in data entry, misplacement of records, or oversight can result in inaccurate attendance logs, which may have downstream effects on payroll, compliance, and performance evaluations.

- **Lack of Real-Time Data Accessibility**

One of the most prominent issues with manual attendance systems is the lack of real-time data accessibility. Teachers, administrators, or managers often have to wait until the end of a day, week, or even month to compile and review attendance records. This delay hinders timely interventions for absenteeism or irregular attendance, which can be particularly detrimental in academic settings where early detection of attendance issues is crucial for student retention and success. In workplaces, delayed or inaccurate attendance data can affect payroll processing, compliance with labor regulations, and overall workforce management.

- **Security and Data Integrity Concerns**

Security and data integrity are significant concerns in traditional attendance management. Physical registers can be lost, damaged, or tampered with, leading to potential disputes and loss of trust among stakeholders. Unauthorized alterations, whether intentional or accidental, can compromise the accuracy of records. Furthermore, the lack of proper backup mechanisms means that once data is lost or corrupted, it is often irretrievable, posing a risk to both organizations and individuals.

- **Scalability Issues**

Scalability is another pressing challenge. As organizations grow, the volume of attendance data increases exponentially. Manual systems struggle to keep pace with this growth, leading to bottlenecks in data entry, storage, and retrieval. In multi-site organizations or institutions with large populations, consolidating attendance data from various sources becomes a logistical nightmare, often resulting in fragmented or incomplete records.

- **Limited Analytical Capabilities**

The absence of automation in conventional attendance systems means that generating analytical reports or insights requires significant manual effort. This limits the ability of organizations to make data-driven decisions regarding resource allocation, policy formulation, or targeted interventions. The inability to quickly identify patterns, such as chronic absenteeism or peak absence periods, can have long-term negative consequences on organizational performance and individual outcomes.

- **Challenges in Remote and Hybrid Work Environments**

The evolving landscape of remote work and hybrid learning has introduced new complexities to attendance management. Existing manual systems are ill-equipped to handle virtual attendance, leading to further gaps in data collection and monitoring. The need for a system that can seamlessly integrate both physical and virtual attendance has become increasingly apparent in the wake of recent global events.

- **The Need for Automated Attendance Management Systems**

Given these challenges, there is a clear and pressing need for a modern, automated, and secure Attendance Management System. Such a system should not only streamline the process of recording and managing attendance but also provide real-time access to accurate data, robust security features, scalability, and analytical capabilities. By addressing the shortcomings of traditional methods, an automated Attendance Management System can enhance operational efficiency, support informed decision-making, and foster greater accountability and transparency within organizations.

Key Benefits of Automated Systems:

- **Enhanced Efficiency:** Automation reduces the time and effort required to record and process attendance data, allowing staff to focus on more strategic tasks.
- **Improved Accuracy:** Automated systems minimize human error, ensuring that attendance records are precise and reliable.
- **Real-Time Monitoring:** Organizations can access up-to-date attendance information, enabling timely interventions and decision-making.
- **Robust Security:** Digital systems offer secure data storage with backup options, protecting against data loss and unauthorized access.
- **Scalability:** Automated solutions can easily accommodate organizational growth, managing increased data volumes without compromising performance.
- **Advanced Analytics:** Built-in reporting tools facilitate the analysis of attendance patterns, helping organizations identify trends and implement effective policies.
- **Adaptability to Remote Work:** Modern systems can track attendance across various settings, including remote and hybrid work environments, ensuring comprehensive coverage.

In conclusion, transitioning from manual to automated attendance management systems is not merely a technological upgrade but a strategic imperative. By embracing automation, organizations can overcome the inherent limitations of traditional methods, leading to improved operational efficiency, data integrity, and overall effectiveness in managing attendance.

2.2 DATA REQUIREMENTS

In developing an effective Attendance Management System, it is crucial to establish comprehensive data requirements that define the information to be collected, maintained, and processed within the system's database. This process involves identifying core entities such as users (students, teachers, employees, administrators), attendance records, classes or departments, and leave requests. For each entity, relevant attributes must be determined—for example, a user's name, role, contact details, and department, or an attendance record's date, status, and remarks.

Additionally, outlining how these entities interrelate is essential. This includes linking attendance records to specific users and sessions, or associating leave requests with approval workflows. Implementing constraints and validation rules, such as ensuring that each attendance entry is unique for a user on a given date or that only authorized personnel can modify records, helps maintain data integrity and reliability. By clearly defining these data requirements, the Attendance Management System can ensure accurate tracking, secure storage, and efficient retrieval of attendance information, ultimately supporting the organization's operational and reporting needs.

USERS Table : This table stores information about all individuals registered in the system, including students, teachers, and administrators.

- `user_id`: Unique identifier for each user (e.g., 101, 102)
- `name`: Full name of the user (e.g., Priya Sharma)
- `email`: User's email address (e.g., priya.sharma@email.com)
- `password`: Encrypted password for authentication
- `user_type`: Role of the user (e.g., Student, Teacher, Admin)
- `created_at`: Date and time when the user was added
- `updated_at`: Date and time when the user details were last updated

CLASSES Table : This table contains details about each class or course offered within the institution.

- `class_id`: Unique identifier for each class (e.g., CSE101)
- `section`: Section of the class (e.g., A, B)
- `name`: Name of the class/course (e.g., Data Structures)
- `semester`: Academic semester (e.g., 4th Semester)
- `class_code`: Code assigned to the class (e.g., DS-2024)
- `faculty_id`: Identifier for the faculty/instructor assigned
- `created_at`: Date and time when the class was created
- `updated_at`: Date and time when the class information was last updated

STUDENT_CLASSES Table : This table establishes the enrollment relationship between students and the classes they attend.

- `id`: Unique identifier for each enrollment record
- `student_id`: Identifier for the student enrolled
- `class_id`: Identifier for the class enrolled in
- `joined_at`: Date and time when the student joined the class

SESSIONS Table : This table tracks the individual sessions or periods conducted for each class.

- `session_id`: Unique identifier for each session (e.g., S1001)
- `class_id`: Identifier for the class to which the session belongs
- `date`: Date of the session (e.g., 2025-05-09)

- `start_time`: Start time of the session (e.g., 09:00 AM)
- `end_time`: End time of the session (e.g., 10:00 AM)
- `faculty_id`: Identifier for the faculty conducting the session
- `created_at`: Date and time when the session was created
- `updated_at`: Date and time when the session details were last updated

ATTENDANCES Table : This table records the attendance status of students for each session.

- `id`: Unique identifier for each attendance record
- `student_id`: Identifier for the student whose attendance is recorded
- `session_id`: Identifier for the session attended
- `attendance_status`: Status of attendance (e.g., Present, Absent, Late)
- `location_status`: Location information (e.g., On-campus, Remote)
- `timestamp`: Date and time when attendance was marked
- `device_info`: Information about the device used to mark attendance
- `created_at`: Date and time when the attendance record was created

2.3 FUNCTIONAL REQUIREMENTS

Functional requirements for the Attendance Management System specify the essential actions and operations the system must perform to support attendance tracking and management. These requirements include the ability to enter, update, retrieve, and delete user, class, session, and attendance data efficiently and accurately. The system should allow multiple users—such as students, teachers, and administrators—to access and interact with the database simultaneously, each according to their assigned roles and permissions. It must support automated processes like bulk attendance uploads, scheduled notifications, and real-time attendance marking. The system should generate comprehensive reports, such as daily attendance summaries, student-wise attendance records, and class-wise statistics, to assist in analysis and decision-making. Additionally, the Attendance Management System must ensure data security through authentication, authorization, and audit trails, while also being scalable to accommodate growing numbers of users and records. These functional requirements collectively define how the system will operate and support the core attendance management processes within the institution.

User & Class Management

- System must allow adding, updating, and deleting user records (students, teachers, administrators).
- Must support creation, modification, and removal of class/course records.
- Should enable enrollment of students into classes and management of these associations.
- All user and class IDs must be unique.

Session Scheduling & Management

- System must allow scheduling, updating, and deleting of class sessions.
- Each session should be linked to a class and an instructor.
- Session dates and times should be validated for logical consistency (e.g., start time before end time).

Attendance Recording

- System must allow marking, editing, and deleting of attendance for each student in every session.
- Must support bulk attendance uploads for a session or class.
- Attendance status (e.g., Present, Absent, Late) must be selected from predefined options.

- Location and device information should be recorded with each attendance entry.
- Timestamp for attendance marking must be auto-recorded.

Reports & Queries

- System must support viewing attendance records filtered by student, class, session, date, or status.
- Must generate attendance reports by class, student, date range, or session.
- Should allow exporting of attendance data and reports in standard formats(e.g., CSV, PDF, EXE).
- Should provide search functionality for users, classes, sessions, and attendance records.

Security & Access Control

- System must provide secure authentication for all users.
- Role-based access control should restrict data access and operations according to user roles (student, teacher, admin).
- Must maintain audit trails of changes to critical data (attendance, user records, class assignments).

Data Validation & Integrity

- All IDs (user, class, session, attendance) must be unique.
- Attendance can only be marked for valid sessions and enrolled students.
- Dates and times must be in valid formats and logically consistent.
- User input (such as names and emails) must be validated for correctness.

Scalability & Performance

- System should be able to handle increasing numbers of users, classes, sessions, and attendance records without performance degradation.
- Must support concurrent access by multiple users.

2.4 ER DIAGRAM

The Entity-Relationship (ER) Diagram is a core tool in designing the Attendance Management System, offering a clear visual model of how data is structured and interrelated. It acts as a blueprint for the database, converting real-world processes like attendance tracking, class handling, and user management into a logical format.

Key entities include Users, Classes, Sessions, Student_Classes, and Attendances, each representing database tables. For instance, “Users” encompasses students, teachers, and admins, while “Classes” and “Sessions” detail course offerings and individual meetings. “Student_Classes” maps student enrollments, and “Attendances” records attendance per session.

Each entity has attributes like user ID, name, email, class ID, section, etc., which uniquely describe and identify records. Relationships illustrate how entities interact—e.g., one class can have many sessions, and students can be enrolled in multiple classes (many-to-many via “Student_Classes”).

Cardinality constraints define these associations, ensuring accuracy—for example, each attendance must link to one student and one session. Primary and foreign keys maintain uniqueness and relational integrity, such as using session ID as a foreign key in the “Attendances” table. This design ensures consistency and simplifies management. In essence, the ER diagram forms the foundation for building a scalable, secure, and efficient attendance system that aligns with institutional requirements and future growth.

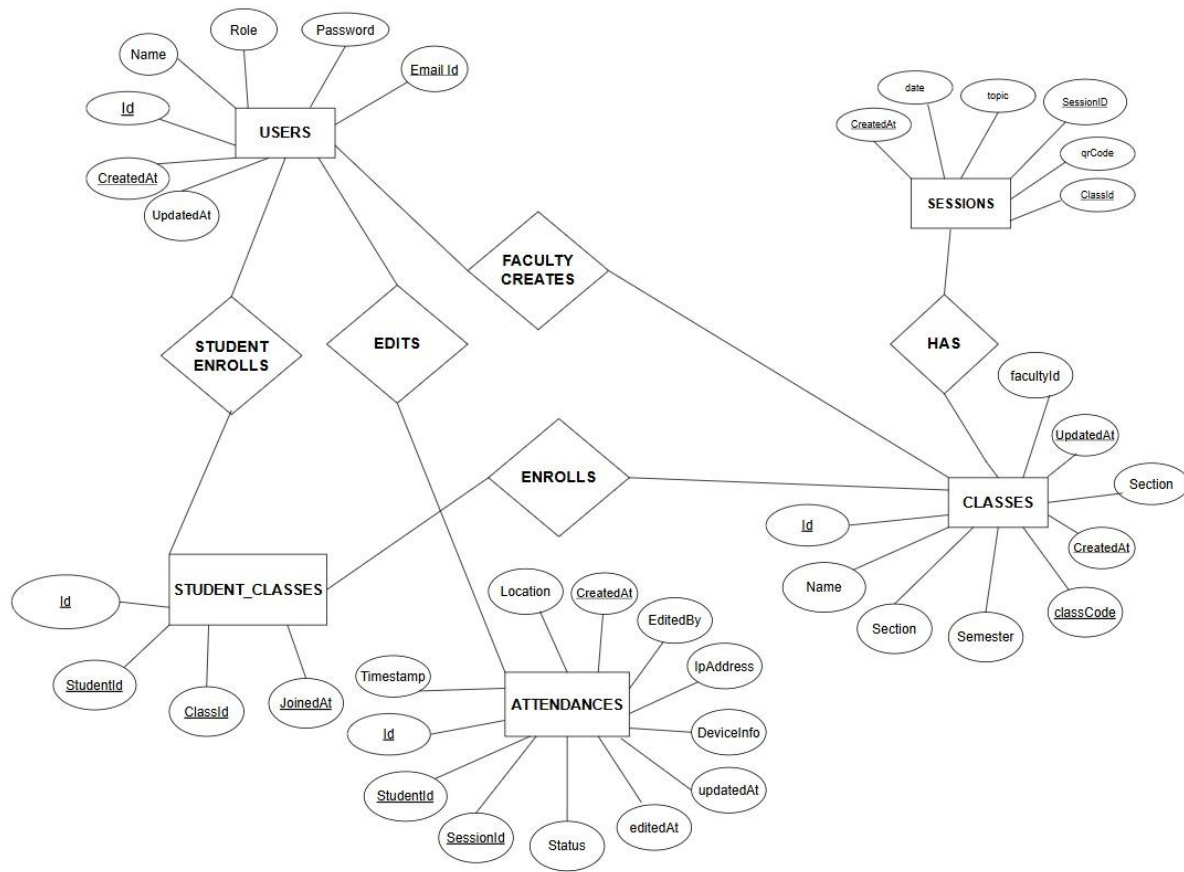


fig 1 : ER DIAGRAM

2.5 ENTITIES AND THEIR ATTRIBUTES

❖ USER Entity

- id (PK) : Integer, Auto-increment - Unique identifier
- Name : String - Full name of the user
- Email : String, Unique - Login credential and contact
- Password : String - Hashed for security
- Role : Enum('student', 'faculty', 'admin') - Access control
- createdAt : DateTime - Account creation time
- updatedAt : DateTime - Last modification time

❖ CLASS Entity

- id (PK) : Integer, Auto-increment - Unique identifier
- Name : String - Course/class name
- Section : String - Section identifier (A, B, etc.)
- Semester : String - Academic term
- classCode : String, Unique - Join code for students
- facultyId (FK) : Integer - References USER(id)
- createdAt : DateTime - Creation time
- updatedAt : DateTime - Last modification time

❖ **SESSION Entity**

- id (PK) : Integer, Auto-increment - Unique identifier
- sessionId : String, Unique - Human-readable identifier
- Topic : String - Session subject/title
- Date : DateTime - Scheduled time
- formLink : String - Optional feedback form
- qrCode : String - Path to QR code image
- classId (FK) : Integer - References CLASS(id)
- createdAt : DateTime - Creation time (for QR expiry)
- updatedAt : DateTime - Last modification time

❖ **STUDENT_CLASS Entity (Junction Table)**

- id (PK) : Integer, Auto-increment - Unique identifier
- studentId (FK) : Integer - References USER(id)
- classId (FK) : Integer - References CLASS(id)
- joinedAt : DateTime - Enrollment time

❖ **ATTENDANCE Entity**

- id (PK) : Integer, Auto-increment - Unique identifier
- studentId (FK) : Integer - References USER(id)
- sessionId (FK) : Integer - References SESSION(id)
- Status : Enum('present', 'absent', 'late') - Attendance status
- Timestamp : DateTime - When attendance was marked
- Location : String - Physical location
- ipAddress : String - Device IP address
- deviceInfo : String - Browser/device information
- editedBy (FK) : Integer - References USER(id)
- editedAt : DateTime - When record was edited
- createdAt : DateTime - Record creation time
- updatedAt : DateTime - Last modification time

2.6 ENTITY RELATIONSHIPS

- USER-CLASS: One-to-Many (Faculty creates classes)
- USER-STUDENT_CLASS: One-to-Many (Student enrolls in classes)
- CLASS-SESSION: One-to-Many (Class has multiple sessions)
- CLASS-STUDENT_CLASS: One-to-Many (Class has multiple students)
- SESSION-ATTENDANCE: One-to-Many (Session has multiple attendance records)
- USER-ATTENDANCE: One-to-Many (Student marks attendance)
- USER-ATTENDANCE: One-to-Many (Faculty/admin edits attendance)

Primary and Foreign Keys

Primary Keys:

- USER: id
- CLASS: id
- SESSION: id
- STUDENT_CLASS: id
- ATTENDANCE: id

Foreign Keys:

- CLASS.facultyId → USER.id
- SESSION.classId → CLASS.id
- STUDENT_CLASS.studentId → USER.id
- STUDENT_CLASS.classId → CLASS.id
- ATTENDANCE.studentId → USER.id
- ATTENDANCE.sessionId → SESSION.id
- ATTENDANCE.editedBy → USER.id

Unique Constraints

- USER.email
- CLASS.classCode
- SESSION.sessionId
- STUDENT_CLASS: Composite (studentId, classId)

Business Rules

- QR codes expire 5 minutes after creation (using SESSION.createdAt)
- Only enrolled students can mark attendance
- Late attendance determined by 15-minute threshold from session start
- Only faculty/admin can edit attendance records
- Audit trail maintained for edited records

2.7 ENTITY AND RELATIONSHIP IDENTIFICATION

❖ ENTITIES

USER

- Represents all system users
- Subtypes: Student, Faculty, Admin (differentiated by role attribute)

CLASS

- Represents academic courses or classes
- Created and managed by faculty

SESSION

- Represents individual class meetings
- Contains QR code for attendance

STUDENT_CLASS

- Junction entity for the many-to-many relationship between students and classes
- Represents class enrollment

ATTENDANCE

- Records student attendance for specific sessions
- Contains verification data and audit information

❖ RELATIONSHIPS

Faculty Creates Classes (USER to CLASS)

- Type: One-to-Many
- A faculty user can create and manage multiple classes
- Each class is created by exactly one faculty member
- Cardinality: 1:N

Student Enrolls in Classes (USER to STUDENT_CLASS)

- Type: One-to-Many
- A student can enroll in multiple classes
- Each enrollment record belongs to exactly one student
- Cardinality: 1:N

Class Has Sessions (CLASS to SESSION)

- Type: One-to-Many
- A class can have multiple attendance sessions
- Each session belongs to exactly one class
- Cardinality: 1:N

Class Enrolls Students (CLASS to STUDENT_CLASS)

- Type: One-to-Many
- A class can have multiple student enrollments
- Each enrollment record is for exactly one class
- Cardinality: 1:N

Session Records Attendance (SESSION to ATTENDANCE)

- Type: One-to-Many
- A session can have multiple attendance records
- Each attendance record belongs to exactly one session
- Cardinality: 1:N

Student Marks Attendance (USER to ATTENDANCE)

- Type: One-to-Many
- A student can mark attendance for multiple sessions
- Each attendance record belongs to exactly one student
- Cardinality: 1:N

User Edits Attendance (USER to ATTENDANCE)

- Type: One-to-Many
- A user (typically faculty or admin) can edit multiple attendance records
- Each edited attendance record is modified by one user at a time
- Cardinality: 1:N

❖ RELATIONSHIP CONSTRAINTS

- A student can only mark attendance for sessions of classes they are enrolled in
- A student can only mark their own attendance
- A faculty member can only create and manage their own classes
- A faculty member can edit attendance records for their own classes
- An admin can edit any attendance record
- A student can only join a class using a valid class code
- Attendance can only be marked within 5 minutes of QR code generation
- Each student can have only one attendance record per session

CHAPTER-3

2B EXPERIMENT

3.1 ENTITY-RELATIONSHIP SCHEMA

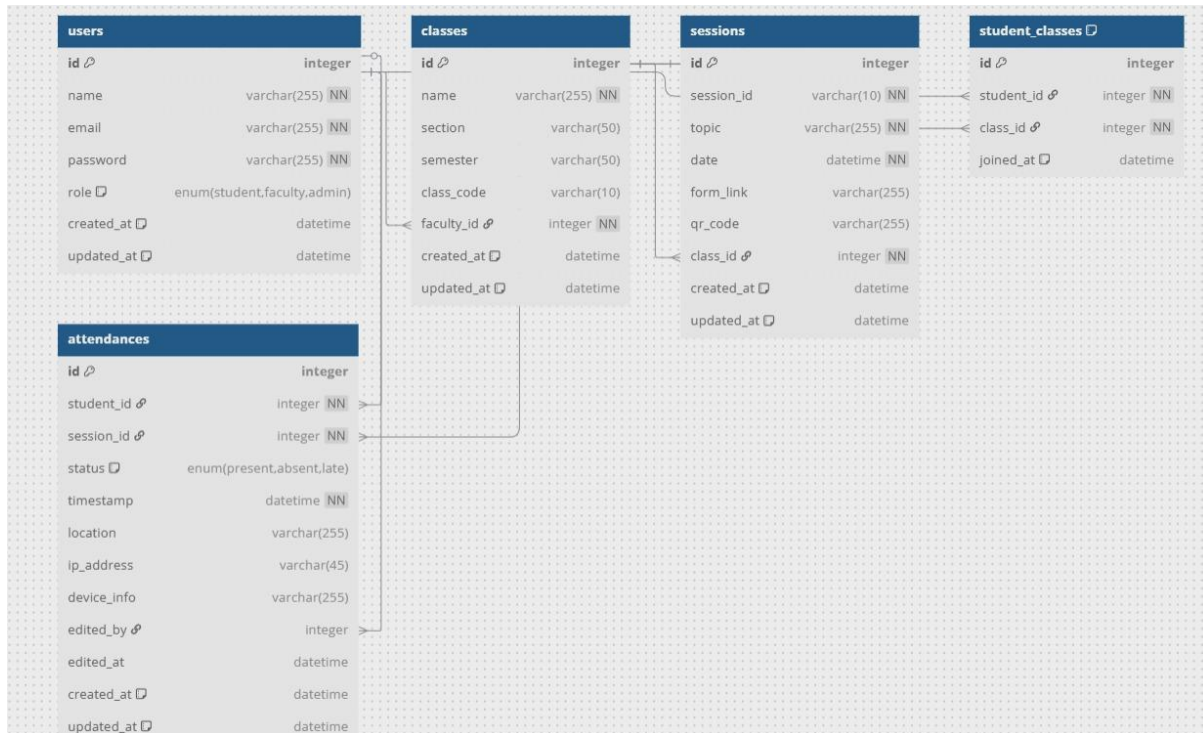


fig 2 : RELATIONAL SCHEMA DESIGN

3.2 DATABASE SCHEMAS

USERS

- Users(
id INTEGER PRIMARY KEY,
name VARCHAR(255) NOT NULL,
email VARCHAR(255) NOT NULL,
password VARCHAR(255) NOT NULL,
role ENUM('student', 'faculty', 'admin'),
created_at DATETIME,
updated_at DATETIME
)

CLASSES

- Classes(
 id INTEGER PRIMARY KEY,
 name VARCHAR(255) NOT NULL,
 section VARCHAR(50),
 semester VARCHAR(50),
 class_code VARCHAR(10),
 faculty_id INTEGER,
 created_at DATETIME,
 updated_at DATETIME,
 FOREIGN KEY (faculty_id) REFERENCES Users(id)
)

SESSIONS

- Sessions(
 id INTEGER PRIMARY KEY,
 session_id VARCHAR(10) NOT NULL,
 topic VARCHAR(255) NOT NULL,
 date DATETIME NOT NULL,
 form_link VARCHAR(255),
 qr_code VARCHAR(255),
 class_id INTEGER NOT NULL,
 created_at DATETIME,
 updated_at DATETIME,
 FOREIGN KEY (class_id) REFERENCES Classes(id)
)

STUDENT_CLASSES

- Student_Classes(
 id INTEGER PRIMARY KEY,
 student_id INTEGER NOT NULL,
 class_id INTEGER NOT NULL,
 joined_at DATETIME,


```

FOREIGN KEY (student_id) REFERENCES Users(id),
FOREIGN KEY (class_id) REFERENCES Classes(id)
)

```

ATTENDANCES

- Attendances(
 - id INTEGER PRIMARY KEY,
 - student_id INTEGER NOT NULL,
 - session_id INTEGER NOT NULL,
 - status ENUM('present', 'absent', 'late'),
 - timestamp DATETIME NOT NULL,
 - location VARCHAR(255),
 - ip_address VARCHAR(45),
 - device_info VARCHAR(255),
 - edited_by INTEGER,
 - edited_at DATETIME,
 - updated_at DATETIME,
 - FOREIGN KEY (student_id) REFERENCES Users(id),
 - FOREIGN KEY (session_id) REFERENCES Sessions(id),
 - FOREIGN KEY (edited_by) REFERENCES Users(id)

3.3 LIST OF TABLES

S.NO	TABLES
1	USERS
2	CLASSES
3	SESSIONS
4	STUDENT_CLASSES
5	ATTENDANCES

Table 1 : tables list

ID	NAME	EMAIL	PASSWORD	ROLE	CREATED AT	UPDATED AT
1	John smith	smith@gmail.com	jklt@234	Faculty	2025-1-12 14:02:34	2025-1-12 14:02:34
2	Rohith	rohith@gmail.com	Sai#2456	Student	2025-2-24 15:45:56	2025-2-24 15:45:56

Table 2 : users table

ID	NAME	SECTION	SEMESTER	CLASS CODE	FACULTY ID	CREATED AT	UPDATED AT
1	DAA	A	Fall 2025	CS101A	2	2025-1-12 14:02:34	2025-1-12 14:02:34
2	DBMS	B	Fall 2025	CS201B	3	2025-2-24 15:45:56	2025-2-24 15:45:56

Table 3 : classes table

ID	SESSION ID	TOPIC	DATE	FORM LINK	QR CODE	CREATED AT	UPDATED AT
1	SE201A	Algorithms	2025-1-12	https://forms.google.com/xyz	/qr.code.org/a	2025-1-12 14:02:34	2025-1-12 14:02:34
2	SE202B	Schemas	2025-1-12	https://forms.google.com/abc	/qr.code.org/b	2025-2-24 15:45:56	2025-2-24 15:45:56

Table 4 : sessions table

ID	STUDENT ID	CLASS ID	JOINED AT
1	5	1	2025-1-12 14:02:34
2	6	2	2025-2-24 15:45:56

Table 5 : student_classes table

ID	STUDENT ID	SESSION ID	STATUS	TIME STAMP	IP ADDRESS	DEVICE INFO	EDITED BY	EDITED AT	CREATED AT
1	John smith	SE201A	PRESENT	long:12.34578, lat:23.34908	:::1	Mozilla//f.123	F123	2025-1-12 14:02:34	2025-1-12 14:02:34
2	Rohith	SE202B	ABSENT	long:12.34578, lat:23.34908	:::2	Mozilla//x.234	F234	2025-2-24 15:45:56	2025-2-24 15:45:56

Table 6 : attendances table

CHAPTER-4

3B EXPERIMENT

4.1 BASIC SQL QUERIES

Basic SQL queries play a crucial role in managing attendance data efficiently within an Attendance Management System. These queries are used to insert, update, retrieve, and delete attendance records from a relational database. For example, INSERT queries add new attendance entries, while UPDATE queries modify existing data, such as correcting an attendance status. SELECT queries are commonly employed to fetch attendance details based on criteria like date, employee ID, or department. Aggregation functions like COUNT can calculate total present or absent days, and JOIN operations link attendance records with user details for comprehensive reports. DELETE queries remove erroneous or obsolete data. Additionally, GROUP BY and ORDER BY clauses help categorize and organize attendance data efficiently. Proper indexing and optimized queries ensure quick access and efficient data management, while constraints and triggers maintain data integrity, preventing anomalies such as duplicate or conflicting records.

4.2 STORED PROCEDURES

- **Identify low attendance students**

```
CREATE DEFINER=root@localhost PROCEDURE identify_low_attendance_students(IN
p_class_id INT, IN p_threshold_percentage DECIMAL(5,2))
BEGIN
    -- Declare variables
    DECLARE done INT DEFAULT FALSE;
    DECLARE student_id INT;
    DECLARE student_name VARCHAR(255);
    DECLARE student_email VARCHAR(255);
    DECLARE attendance_percentage DECIMAL(5,2);
    DECLARE low_attendance_cursor CURSOR FOR
    SELECT
        u.id,
        u.name,
        u.email,
        IFNULL(
            (COUNT(DISTINCT CASE WHEN a.status IN ('present', 'late') THEN s.id END) /
```

```

COUNT(DISTINCT s.id)) * 100,
0
) AS attendance_pct
FROM
users u
JOIN student_classes sc ON u.id = sc.studentId
JOIN sessions s ON sc.classId = s.classId
LEFT JOIN attendances a ON s.id = a.sessionId AND u.id = a.studentId
WHERE
sc.classId = p_class_id
GROUP BY
u.id, u.name, u.email
HAVING
attendance_pct < p_threshold_percentage
ORDER BY
attendance_pct ASC;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
DROP TEMPORARY TABLE IF EXISTS temp_low_attendance;
CREATE TEMPORARY TABLE temp_low_attendance (
student_id INT,
student_name VARCHAR(255),
student_email VARCHAR(255),
attendance_percentage DECIMAL(5,2)
);
OPEN low_attendance_cursor;
read_loop: LOOP
FETCH low_attendance_cursor INTO student_id, student_name, student_email,
attendance_percentage;
IF done THEN
LEAVE read_loop;
END IF;
INSERT INTO temp_low_attendance
VALUES (student_id, student_name, student_email, attendance_percentage);
END LOOP;

```

```

CLOSE low_attendance_cursor;

SELECT * FROM temp_low_attendance;

DROP TEMPORARY TABLE IF EXISTS temp_low_attendance;

END

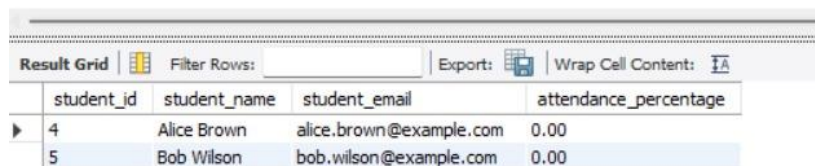
```

- This stored procedure, **identify_low_attendance_students**, identifies students in a specific class whose attendance percentage (based on being present or late) falls below a given threshold. It gathers their details into a temporary table and returns the list of such students sorted by lowest attendance.

```

1 • USE attendance_system;
2 • CALL identify_low_attendance_students(1,18.00);
3

```



	student_id	student_name	student_email	attendance_percentage
▶	4	Alice Brown	alice.brown@example.com	0.00
	5	Bob Wilson	bob.wilson@example.com	0.00

fig.3 : STORED PROCEDURE

4.3 TRIGGERS

- **update_attendance_status**

```

DELIMITER //

CREATE TRIGGER update_attendance_status
BEFORE INSERT ON attendances
FOR EACH ROW
BEGIN
    DECLARE session_start_time DATETIME;
    DECLARE late_threshold INT DEFAULT 15;

    SELECT date INTO session_start_time FROM sessions WHERE id = NEW.sessionId;

    IF NEW.status = 'present' AND NEW.timestamp > DATE_ADD(session_start_time,
    INTERVAL late_threshold MINUTE) THEN
        SET NEW.status = 'late';
    END IF;
END //

DELIMITER ;

```

- This trigger, **update_attendance_status**, automatically updates a student's attendance status to 'late' instead of 'present' if their check-in (timestamp) is more than 15 minutes after the session start time. It runs before inserting a new attendance record.

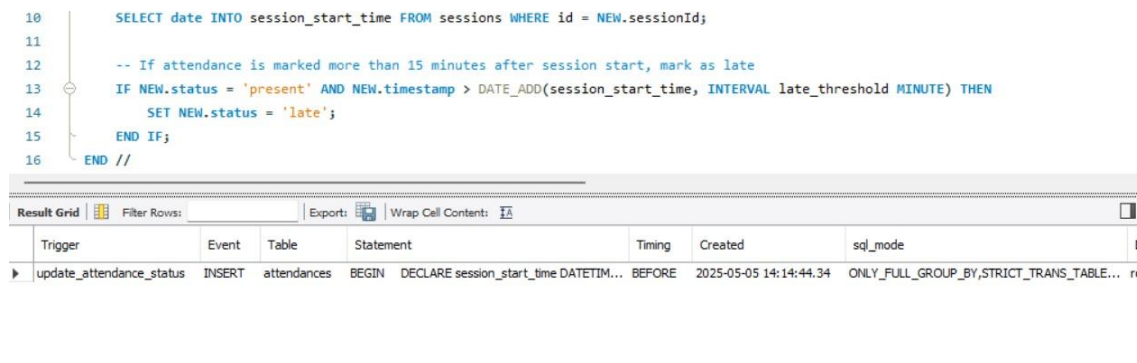


fig 4 : TRIGGER

4.4 CURSORS

- **Identify low attendance students**

DELIMITER //

CREATE PROCEDURE identify_low_attendance_students(IN p_class_id INT, IN p_threshold_percentage DECIMAL(5,2))

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE student_id INT;

DECLARE student_name VARCHAR(255);

DECLARE student_email VARCHAR(255);

DECLARE attendance_percentage DECIMAL(5,2);

DECLARE low_attendance_cursor CURSOR FOR

SELECT u.id, u.name, u.email, IFNULL((COUNT(DISTINCT CASE WHEN a.status IN ('present', 'late') THEN s.id END) / COUNT(DISTINCT s.id)) * 100, 0) AS attendance_pct

FROM users u

JOIN student_classes sc ON u.id = sc.studentId

JOIN sessions s ON sc.classId = s.classId

LEFT JOIN attendances a ON s.id = a.sessionId AND u.id = a.studentId

WHERE sc.classId = p_class_id

GROUP BY u.id, u.name, u.email

HAVING attendance_pct < p_threshold_percentage

ORDER BY attendance_pct ASC;

```

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

DROP TEMPORARY TABLE IF EXISTS temp_low_attendance;

CREATE TEMPORARY TABLE temp_low_attendance (student_id INT, student_name
VARCHAR(255), student_email VARCHAR(255), attendance_percentage DECIMAL(5,2));

OPEN low_attendance_cursor;

read_loop: LOOP

    FETCH low_attendance_cursor INTO student_id, student_name, student_email,
attendance_percentage;

    IF done THEN
        LEAVE read_loop;
    END IF;

    INSERT INTO temp_low_attendance VALUES (student_id, student_name, student_email,
attendance_percentage);

END LOOP;

CLOSE low_attendance_cursor;

SELECT * FROM temp_low_attendance;

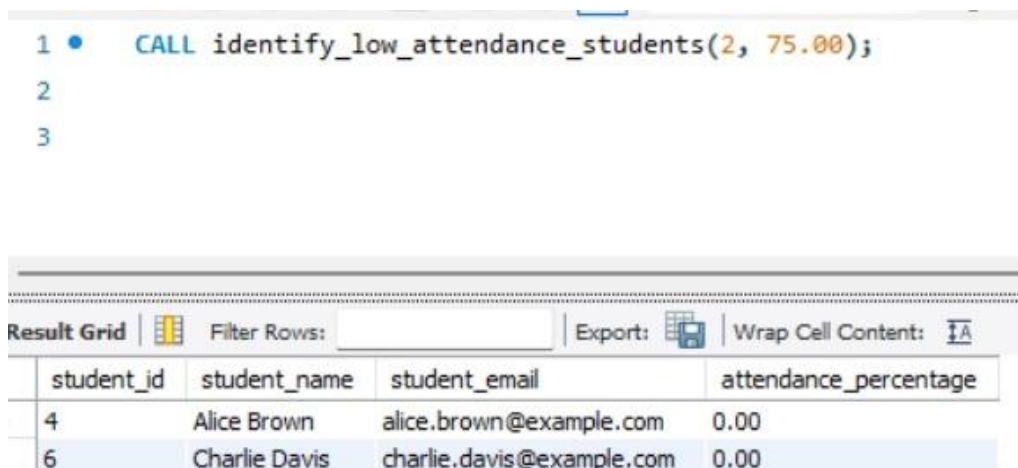
DROP TEMPORARY TABLE IF EXISTS temp_low_attendance;

END //

DELIMITER ;

```

- This cursor-based stored procedure identifies students in a specified class whose attendance percentage falls below a given threshold. It fetches student details using a cursor, stores the qualifying students temporarily, and returns the result as a list.



The screenshot shows a database query editor with the following SQL command:

```
1 • CALL identify_low_attendance_students(2, 75.00);
```

Below the query editor, a "Result Grid" displays the output of the query. The grid has four columns: student_id, student_name, student_email, and attendance_percentage. Two rows of data are shown:

student_id	student_name	student_email	attendance_percentage
4	Alice Brown	alice.brown@example.com	0.00
6	Charlie Davis	charlie.davis@example.com	0.00

fig.5 : CURSOR

4.5 FUNCTIONS

- **get_student_session_status**

```
CREATE DEFINER=root@localhost FUNCTION get_student_session_status(p_student_id INT,  
p_session_id INT) RETURNS varchar(10) CHARSET utf8mb4
```

```
READS SQL DATA
```

```
DETERMINISTIC
```

```
BEGIN
```

```
DECLARE attendance_status VARCHAR(10);
```

```
SELECT status INTO attendance_status
```

```
FROM attendances
```

```
WHERE studentId = p_student_id AND sessionId = p_session_id
```

```
LIMIT 1;
```

```
IF attendance_status IS NULL THEN
```

```
    RETURN 'absent';
```

```
ELSE
```

```
    RETURN attendance_status;
```

```
END IF;
```

```
END
```

- The get_student_session_status function is a stored routine in MySQL designed to retrieve a student's attendance status for a specific session within an Attendance Management System. It accepts two input parameters: p_student_id and p_session_id. The function queries the attendances table to find a matching record based on these parameters. If a corresponding attendance record exists, it returns the associated status (e.g., 'present', 'absent'). If no record is found, the function defaults to returning 'absent'.



fig.6 : FUNCTION

CHAPTER-5

4B EXPERIMENT

Normalization is the systematic approach of decomposing complex data structures into simpler, more manageable tables. By doing so, it ensures that the database adheres to specific rules, known as normal forms, each addressing particular types of redundancy and dependency issues. This structuring facilitates efficient data management and retrieval.

OBJECTIVES OF NORMALIZATION

- **Eliminate Redundancy:** Ensure that each data item is stored only once to prevent duplication.
- **Prevent Anomalies:** Avoid inconsistencies during data insertion, deletion, or updating by maintaining a structured schema.
- **Enhance Data Integrity:** Maintain consistent and accurate data across the database.
- **Simplify Maintenance:** Make the database easier to manage and scale as organizational needs evolve.

5.1 APPLYING 1NF

Definition: A table is in 1NF if all its attributes contain only atomic (indivisible) values, and each record is unique.

Key Rules:

- **Atomicity:** Each column must contain indivisible values; no sets or arrays.
- **Unique Rows:** Each record must be distinct.
- **Unique Column Names:** Every column should have a unique name.
- **Order Irrelevance:** The order of rows and columns does not affect the data integrity.

Example: A table storing multiple phone numbers in a single field violates 1NF. To comply, each phone number should be stored in a separate row or column.

5.2 APPLYING 2NF

Definition: A table is in 2NF if it is in 1NF and all non-key attributes are fully functionally dependent on the entire primary key.

Key Rules:

- **1NF Compliance:** The table must first satisfy all 1NF conditions.
- **Full Functional Dependency:** Non-key attributes must depend on the whole primary key, not just a part of it.
- **Elimination of Partial Dependencies:** Attributes that depend only on a portion of a composite key should be moved to a separate table.

Example : In a table with a composite key (StudentID, CourseID), if the attribute 'StudentName' depends only on 'StudentID', it should be moved to a separate table to achieve 2NF.

5.3 APPLYING 3NF

Definition: A table is in 3NF if it is in 2NF and all the attributes are only dependent on the primary key, eliminating transitive dependencies.

Key Rules:

- **2NF Compliance:** The table must first satisfy all 2NF conditions.
- **No Transitive Dependencies:** Non-key attributes should not depend on other non-key attributes.
- **Direct Dependency:** Every non-key attribute must depend directly on the primary key.

Example: If 'DepartmentName' depends on 'DepartmentID', and 'DepartmentID' depends on 'EmployeeID', then 'DepartmentName' indirectly depends on 'EmployeeID'. To achieve 3NF, 'DepartmentName' should be moved to a separate table.

5.4 APPLYING BCNF

Definition: A table is in BCNF if it is in 3NF and for every functional dependency ($X \rightarrow Y$), X is a super key.

Key Rules:

- **3NF Compliance:** The table must first satisfy all 3NF conditions.
- **Super Key Requirement:** For every functional dependency, the determinant must be a super key.

Example : If in a table, 'Course' determines 'Instructor', but 'Course' is not a super key, this violates BCNF. To comply, the table should be decomposed so that 'Course' becomes a super key in its respective table.

ID	NAME	EMAIL	PASSWORD	ROLE	CREATED AT	UPDATED AT
1	John smith	smith@gmail.com	jdkty@234	Faculty	2025-1-12 14:02:34	2025-1-12 14:02:34
2	Rohith	rohith@gmail.com	Sai#2456	Student	2025-2-24 15:45:56	2025-2-24 15:45:56

Table 7 : users table_NF

FD'S :

- $id \rightarrow name, email, password, role, createdAt, updatedAt$
- $email \rightarrow id, name, password, role, createdAt, updatedAt$

Users table_NF follows 1NF, 2NF, 3NF, BCNF

ID	NAME	SECTION	SEMESTER	CLASS CODE	FACULTY ID	CREATED AT	UPDATED AT
1	DAA	A	Fall 2025	CS101A	2	2025-1-12 14:02:34	2025-1-12 14:02:34
2	DBMS	B	Fall 2025	CS201B	3	2025-2-24 15:45:56	2025-2-24 15:45:56

Table 8 : classes table_NF

FD'S :

- $id \rightarrow name, section, semester, classCode, facultyId, createdAt, updatedAt$
- $classCode \rightarrow id, name, section, semester, facultyId, createdAt, updatedAt$

Classes table_NF follows 1NF, 2NF, 3NF, BCNF

ID	SESSION ID	TOPIC	DATE	FORM LINK	QR CODE	CREATED AT	UPDATED AT
1	SE201A	Algorithms	2025-1-12	https://forms.google.com/xyz	/qr.code.org/a	2025-1-12 14:02:34	2025-1-12 14:02:34
2	SE202B	Schemas	2025-1-12	https://forms.google.com/abc	/qr.code.org/b	2025-2-24 15:45:56	2025-2-24 15:45:56

Table 9 : sessions table_NF

FD'S :

- $id \rightarrow sessionId, topic, date, formLink, qrCode, classId, createdAt, updatedAt$
- $sessionId \rightarrow id, topic, date, formLink, qrCode, classId, createdAt, updatedAt$

Sessions table_NF follows 1NF , 2NF , 3NF , BCNF

ID	STUDENT ID	CLASS ID	JOINED AT
1	5	1	2025-1-12 14:02:34
2	6	2	2025-2-24 15:45:56

Table 10 : student_classes table_NF

FD'S :

- $id \rightarrow studentId, classId, joinedAt$
- $(studentId, classId) \rightarrow id, joinedAt$

Students_Classes table_NF follows 1NF, 2NF , 3NF , BCNF

ID	STUDENT ID	SESSION ID	STATUS	TIME STAMP	IP ADDRESS	DEVICE INFO	EDITED BY	EDITED AT	CREATED AT
1	John smith	SE201A	PRESENT	long:12.34578, lat:23.34908	:::1	Mozilla/f.123	F123	2025-1-12 14:02:34	2025-1-12 14:02:34
2	Rohith	SE202B	ABSENT	long:12.34578, lat:23.34908	:::2	Mozilla/x.234	F234	2025-2-24 15:45:56	2025-2-24 15:45:56

Table 11 : attendances table_NF

FD'S :

- $id \rightarrow studentId, sessionId, status, timestamp, location, ipAddress, deviceInfo, editedBy, editedAt, createdAt, updatedAt$
- $(studentId, sessionId) \rightarrow id, status, timestamp, location, ipAddress, deviceInfo, editedBy, editedAt, createdAt, updatedAt$

Attendances table_NF follows 1NF , 2NF, 3NF, BCNF

CHAPTER 6

IMPLEMENTATION OF CONCURRENCY CONTROL AND RECOVERY MECHANISMS

6.1 OVERVIEW

1. Atomicity

Atomicity ensures that transactions are treated as a single, indivisible unit that either completes entirely or fails entirely.

Implementation in this project:

- **Stored Procedures:** The system uses stored procedures like `mark_attendance` which ensure that all operations within the procedure either complete successfully or are rolled back if an error occurs.
- **Transaction Management:** In operations like marking attendance, the system ensures that both checking if a student is enrolled and updating/inserting attendance records happen as a single atomic operation.
- **Attendance Marking Process:** When a student scans a QR code to mark attendance, the entire process (validating student enrollment, checking session validity, recording attendance) completes as a single unit.

2. Consistency

Consistency ensures that a transaction brings the database from one valid state to another valid state.

Implementation in this project:

- **Foreign Key Constraints:** The database uses foreign key constraints (e.g., `studentId` references `users.id`) to maintain referential integrity.
- **Data Validation:** The system validates data before insertion, such as checking if a student is enrolled in a class before allowing attendance marking.
- **Triggers:** The `update_attendance_status` trigger automatically updates attendance status based on time, ensuring consistent application of business rules.
- **QR Code Time Limit:** The 5-minute time limit for QR codes ensures consistency in the attendance marking process, preventing late submissions from being incorrectly marked as on-time.

3. Isolation

Isolation ensures that concurrent transactions do not interfere with each other.

Implementation in this project:

- **MySQL's Default Isolation Level:** The system uses MySQL's default isolation level (`REPEATABLE READ`), which prevents dirty reads and non-repeatable reads.
- **Session-Based Operations:** Attendance marking for different sessions operates independently, reducing contention between transactions.
- **Unique Constraints:** Constraints like `unique_student_class` in the `student_classes` table prevent race conditions where the same student might be enrolled in a class twice.

4. Durability

Durability ensures that once a transaction is committed, it remains so even in the event of a system failure.

Implementation in this project:

- **MySQL Storage Engine:** Using InnoDB (MySQL's default storage engine), the system provides durability through write-ahead logging and data flushing.
- **Timestamp Tracking:** The system records timestamps for all critical operations (createdAt, updatedAt), helping in recovery and auditing.
- **Attendance Records:** Once attendance is marked, it's permanently stored in the database and can be audited or reviewed later.

6.2 CRITICAL ACID SCENARIOS IN THIS SYSTEM

Class Enrollment Process:

- Atomicity ensures a student is either enrolled in a class completely or not at all
- Consistency ensures enrollment only happens for valid students and classes
- Isolation prevents duplicate enrollments during concurrent registration
- Durability ensures enrollment records persist after system restarts

Attendance Marking Process:

- Atomicity ensures attendance status is completely recorded
- Consistency ensures only enrolled students can mark attendance and status is correctly set based on time
- Isolation prevents race conditions when multiple students mark attendance simultaneously
- Durability ensures attendance records are permanent once committed

Session Creation:

- Atomicity ensures all session details (including QR code generation) are created as a unit
- Consistency ensures sessions are only created for valid classes
- Isolation prevents conflicts when multiple faculty create sessions concurrently
- Durability ensures session details persist after creation

6.3 CONCURRENCY CONTROL

1. Locking-Based Concurrency Control

- **Row-Level Locking:** The MySQL InnoDB engine provides row-level locking, allowing multiple transactions to access different rows in the same table simultaneously. This is crucial when multiple students are marking attendance for the same session.

- **Table-Level Locking:** For operations that affect entire tables, such as generating comprehensive attendance reports using cursors, table-level locks are employed.

2. Isolation Levels

- **REPEATABLE READ:** This default MySQL isolation level prevents dirty reads and non-repeatable reads, ensuring consistent data views during transactions.
- **Application for Attendance Marking:** When multiple students scan a QR code simultaneously, this isolation level ensures that each attendance record is processed correctly without interference.

3. Deadlock Prevention

- **Timeout Mechanisms:** The system implements timeout mechanisms for attendance marking to prevent indefinite waiting.
- **Transaction Ordering:** Critical operations like enrollment and attendance marking follow a consistent order of resource access to minimize deadlock possibilities.

4. Optimistic Concurrency Control

- **Version-Based Control:** For faculty editing attendance records, the system uses timestamps (updatedAt) to detect conflicting updates.
- **Conditional Updates:** Updates to attendance records include conditions based on the last known state to prevent lost updates.

5. Concurrency Hotspots Management

- **QR Code Scanning:** The highest concurrency occurs during QR code scanning for attendance. The system handles this through efficient database indexing and minimizing transaction duration.
- **Class Enrollment Periods:** During peak enrollment periods, the system manages concurrency through unique constraints and proper transaction isolation.

6.4 RECOVERY MECHANISMS

1. Transaction Logging

- **Redo Logs:** MySQL's InnoDB redo logs record all changes made to the database, allowing recovery to the most recent committed state after a crash.
- **Undo Logs:** These logs enable the system to roll back incomplete transactions after a failure.

2. Backup Strategies

- **Regular Backups:** The system performs scheduled backups of the entire database to enable point-in-time recovery.
- **Incremental Backups:** Between full backups, incremental backups capture only changes, reducing backup time and storage requirements.

3. Checkpoint Mechanisms

- **Database Checkpoints:** Regular checkpoints minimize recovery time by reducing the number of log records that need to be processed after a failure.

4. Application-Level Recovery

- **Session State Preservation:** For faculty creating attendance sessions, the system preserves session creation state to allow recovery from browser crashes or network issues.

- **QR Code Regeneration:** If a QR code display fails, faculty can regenerate it without creating duplicate attendance sessions.

5. Error Handling and Reporting

- **Detailed Error Logging:** All database errors are logged with context information to facilitate diagnosis and recovery.
- **User Notification:** When recoverable errors occur, users receive appropriate notifications and guidance on how to proceed.

6. Data Consistency Verification

- **Integrity Checks:** After recovery, the system verifies referential integrity between related tables (users, classes, sessions, attendances).
- **Attendance Reconciliation:** If inconsistencies are detected in attendance records, the system provides tools for faculty to reconcile the data.

6.5 TRANSACTION CONTROL MECHANISM

1. Explicit Transaction Control

- The system uses explicit transaction boundaries for complex operations:

```
BEGIN;
```

```
-- Operations for enrolling students in classes
```

```
-- or marking attendance for multiple students
```

```
COMMIT;
```

- For multi-step operations, savepoints are implemented to allow partial rollbacks when needed :

```
BEGIN;
```

```
SAVEPOINT before_attendance_update;
```

```
-- Update attendance records
```

```
-- If validation fails:
```

```
ROLLBACK TO SAVEPOINT before_attendance_update;
```

```
COMMIT;
```

2. Stored Procedure Transaction Management

- The `mark_attendance` procedure encapsulates transaction logic to ensure atomicity when recording student attendance. It validates student enrollment before making any changes to attendance records, ensuring all operations succeed or fail as a unit.

3. Trigger-Based Transaction Control

- The `update_attendance_status` trigger automatically updates attendance status based on time. This trigger operates within the same transaction as the attendance insertion, ensuring consistency.

4. Isolation Level Configuration

The system configures appropriate isolation levels for different operations:

- For attendance marking: REPEATABLE READ to prevent inconsistencies when multiple students mark attendance
- For reporting functions: READ COMMITTED to improve performance while maintaining consistency

5. Error Handling and Rollback

Comprehensive error handling is implemented in critical operations:

BEGIN;

-- Try to perform operations

-- If an error occurs:

HANDLER:

ROLLBACK;

-- Log error details

COMMIT;

6. Autocommit Management

- For simple operations, the system relies on MySQL's autocommit functionality, while disabling it for complex multi-step processes to maintain transaction integrity.

7. Distributed Transaction Handling

- When integrating with external systems (like notification services), transaction consistency is ensured through two-phase commit protocols or compensating transactions.

8. Deadlock Management

- The system implements deadlock detection and prevention strategies, including transaction retry logic for operations that might encounter deadlocks during peak usage.
- These transaction control mechanisms ensure that the Attendance Management System maintains data integrity even during concurrent operations and system failures.

CHAPTER 7

PROJECT SOURCE CODE

FRONTEND :

LOGIN PAGE :

```
<div class="flex flex-col items-center">
<div class="w-full max-w-6xl">
  <!-- Login/Register Section -->
  <div class="bg-white rounded-lg shadow-md overflow-hidden mb-8">
    <div class="flex flex-col md:flex-row">
      <div class="md:w-1/2 p-8 flex flex-col justify-center">
        <div class="flex space-x-4">
          <a href="/users/login" class="bg-blue-600 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded">
            Login
          </a>
          <a href="/users/register" class="bg-gray-200 hover:bg-gray-300 text-gray-800 font-bold py-2 px-4 rounded">
            Register
          </a>
        </div>
      </div>
      <div class="md:w-1/2 bg-blue-600 p-8 flex items-center justify-center">
        
      </div>
    </div>
  </div>
</div>
```

STUDENT DASHBOARD:

```
<div class="w-full">
  <h1 class="text-2xl font-bold mb-6">Faculty Dashboard</h1>
```

```

<!-- Stats Cards -->
<div class="grid grid-cols-1 md:grid-cols-3 gap-6 mb-8">
  <div class="bg-white rounded-lg shadow-md p-6">
    <div class="flex items-center">
      <div class="p-3 rounded-full bg-blue-100 text-blue-600 mr-4">
        <i class="fas fa-book text-2xl"></i>
      </div>
      <div>
        <p class="text-gray-500">My Classes</p>
        <p class="text-2xl font-bold">{{classesCount}}</p>
      </div>
    </div>
  </div>
</div>

<div class="bg-white rounded-lg shadow-md p-6">
  <div class="flex items-center">
    <div class="p-3 rounded-full bg-green-100 text-green-600 mr-4">
      <i class="fas fa-users text-2xl"></i>
    </div>
    <div>
      <p class="text-gray-500">Total Students</p>
      <p class="text-2xl font-bold">{{studentsCount}}</p>
    </div>
  </div>
</div>

<div class="bg-white rounded-lg shadow-md p-6">
  <div class="flex items-center">
    <div class="p-3 rounded-full bg-purple-100 text-purple-600 mr-4">
      <i class="fas fa-calendar-check text-2xl"></i>
    </div>
    <div>

```

```

    <p class="text-gray-500">Sessions Created</p>
    <p class="text-2xl font-bold">{{sessionsCount}}</p>
  </div>
</div>
</div>
</div>
<!-- Quick Actions -->
<div class="bg-white rounded-lg shadow-md p-6 mb-8">
  <h2 class="text-xl font-semibold mb-4">Quick Actions</h2>
  <div class="grid grid-cols-1 md:grid-cols-3 gap-4">
    <a href="/classes/create" class="bg-blue-600 hover:bg-blue-700 text-white text-center py-3 px-4 rounded">
      <i class="fas fa-plus-circle mr-2"></i> Create New Class
    </a>
    <a href="/classes/my-classes" class="bg-green-600 hover:bg-green-700 text-white text-center py-3 px-4 rounded">
      <i class="fas fa-book mr-2"></i> Manage My Classes
    </a>
    <a href="#" id="create-session-btn" class="bg-purple-600 hover:bg-purple-700 text-white text-center py-3 px-4 rounded">
      <i class="fas fa-qrcode mr-2"></i> Create New Session
    </a>
  </div>
</div>

<!-- Recent Classes -->
<div class="bg-white rounded-lg shadow-md p-6 mb-8">
  <h2 class="text-xl font-semibold mb-4">My Classes</h2>

  {{#if classes.length}}
  <div class="overflow-x-auto">
    <table class="min-w-full bg-white">
      <thead>

```

```

<tr>
  <th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">Class Name</th>
  <th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">Section</th>
  <th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">Semester</th>
  <th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">Class Code</th>
  <th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">Students</th>
  <th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">Actions</th>
</tr>
</thead>
<tbody class="divide-y divide-gray-200">
  {{#each classes}}
    <tr>
      <td class="py-3 px-4">{{this.name}}</td>
      <td class="py-3 px-4">{{this.section}}</td>
      <td class="py-3 px-4">{{this.semester}}</td>
      <td class="py-3 px-4">
        <span class="px-2 py-1 text-xs font-semibold rounded-full bg-blue-100 text-blue-800">
          Class Code: {{this.classCode}}
        </span>
        {{#unless this.classCode}}
          <span class="text-red-500">Code missing</span>
        {{/unless}}
      </td>
      <td class="py-3 px-4">{{this.students.length}}</td>
      <td class="py-3 px-4">
        <a href="/classes/details/{{this.id}}" class="text-blue-600 hover:text-blue-800 mr-2">
          <i class="fas fa-eye"></i> View

```

```

        </a>

        <a href="/sessions/create/{{this.id}}" class="text-green-600 hover:text-green-
800">

            <i class="fas fa-plus-circle"></i> Add Session

        </a>

    </td>

</tr>

{{/each}}

</tbody>

</table>

</div>

{{else}}

    <div class="bg-yellow-100 text-yellow-800 p-4 rounded">

        <p>You haven't created any classes yet. <a href="/classes/create" class="font-bold
underline">Create a class</a> to get started.</p>

    </div>

{{/if}}

</div>

<!-- Recent Sessions -->

<div class="bg-white rounded-lg shadow-md p-6">

    <h2 class="text-xl font-semibold mb-4">Recent Sessions</h2>

    {{#if recentSessions.length}}

    <div class="overflow-x-auto">

        <table class="min-w-full bg-white">

            <thead>

                <tr>

                    <th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600
uppercase tracking-wider">Class</th>

                    <th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600
uppercase tracking-wider">Topic</th>

                    <th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600
uppercase tracking-wider">Date</th>

```

```

        <th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600
uppercase tracking-wider">Session ID</th>

        <th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600
uppercase tracking-wider">Attendance</th>

        <th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600
uppercase tracking-wider">Actions</th>

    </tr>

</thead>

<tbody class="divide-y divide-gray-200">

    {{#each recentSessions}}

    <tr>

        <td class="py-3 px-4">{{this.class.name}}</td>

        <td class="py-3 px-4">{{this.topic}}</td>

        <td class="py-3 px-4">{{formatDate this.date}}</td>

        <td class="py-3 px-4">

            <span class="px-2 py-1 text-xs font-semibold rounded-full bg-purple-100 text-
purple-800">

                {{this.sessionId}}

            </span>

            </td>

            <td class="py-3 px-4">{{this.attendanceCount}} /
{{this.class.students.length}}</td>

            <td class="py-3 px-4">

                <a href="/sessions/details/{{this.id}}" class="text-blue-600 hover:text-blue-800
mr-2">

                    <i class="fas fa-eye"></i> View

                </a>

                <a href="/attendance/session/{{this.id}}" class="text-green-600 hover:text-green-
800">

                    <i class="fas fa-clipboard-check"></i> Attendance

                </a>

            </td>

        </tr>

    {{/each}}

```



```

        </tbody>
    </table>
</div>
{{else}}
<div class="bg-yellow-100 text-yellow-800 p-4 rounded">
    <p>No sessions created yet. Create a class and add sessions to see them here.</p>
</div>
{{/if}}
</div>
</div>
<!-- Create Session Modal -->
<div id="create-session-modal" class="fixed inset-0 bg-gray-800 bg-opacity-75 flex items-center justify-center z-50 hidden">
    <div class="bg-white rounded-lg shadow-xl w-full max-w-md">
        <div class="bg-blue-600 text-white px-6 py-4 rounded-t-lg">
            <h3 class="text-xl font-bold">Create New Session</h3>
        </div>
        <div class="p-6">
            {{#if classes.length}}
            <form action="/sessions/create-quick" method="POST">
                <div class="mb-4">
                    <label for="classId" class="block text-gray-700 text-sm font-bold mb-2">Select
Class</label>
                    <select id="classId" name="classId" class="shadow appearance-none border rounded
w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
required>
                        <option value="">-- Select a Class --</option>
                        {{#each classes}}
                        <option value="{{this.id}}">{{this.name}} ({{this.section}})</option>
                        {{/each}}
                    </select>
                </div>
                <div class="mb-4">

```

```

        <label for="topic" class="block text-gray-700 text-sm font-bold mb-2">Topic</label>

        <input type="text" id="topic" name="topic" class="shadow appearance-none border
rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-
outline" required>

    </div>

    <div class="mb-4">

        <label for="date" class="block text-gray-700 text-sm font-bold mb-2">Date</label>

        <input type="date" id="date" name="date" class="shadow appearance-none border
rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-
outline" required>

    </div>

    <div class="mb-4">

        <label for="formLink" class="block text-gray-700 text-sm font-bold mb-2">Google
Form Link (Optional)</label>

        <input type="url" id="formLink" name="formLink" class="shadow appearance-none
border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none
focus:shadow-outline">

    </div>

    <div class="flex justify-end">

        <button type="button" id="cancel-session-btn" class="bg-gray-300 hover:bg-gray-
400 text-gray-800 font-bold py-2 px-4 rounded mr-2">

            Cancel

        </button>

        <button type="submit" class="bg-blue-600 hover:bg-blue-700 text-white font-bold
py-2 px-4 rounded">

            Create Session

        </button>

    </div>

</form>

{{else}}

<div class="bg-yellow-100 text-yellow-800 p-4 rounded mb-4">

    <p>You need to create a class first before creating a session.</p>

</div>

<div class="flex justify-end">

    <a href="/classes/create" class="bg-blue-600 hover:bg-blue-700 text-white font-bold
py-2 px-4 rounded">

```

```

        Create Class
    </a>
</div>
{{/if}}
</div>
</div>
</div>

<script>
    // Modal functionality
    const modal = document.getElementById('create-session-modal');
    const openBtn = document.getElementById('create-session-btn');
    const closeBtn = document.getElementById('cancel-session-btn');

    openBtn.addEventListener('click', () => {
        modal.classList.remove('hidden');
    });

    if (closeBtn) {
        closeBtn.addEventListener('click', () => {
            modal.classList.add('hidden');
        });
    }

    // Close modal when clicking outside
    window.addEventListener('click', (e) => {
        if (e.target === modal) {
            modal.classList.add('hidden');
        }
    });

    // Set default date to today

```

```

const dateInput = document.getElementById('date');
if (dateInput) {
  const today = new Date();
  const yyyy = today.getFullYear();
  const mm = String(today.getMonth() + 1).padStart(2, '0');
  const dd = String(today.getDate()).padStart(2, '0');
  dateInput.value = `${yyyy}-${mm}-${dd}`;
}
</script>

```

FACULTY DASHBOARD:

```

<div class="w-full">
  <h1 class="text-2xl font-bold mb-6">Student Dashboard</h1>

  <!-- Stats Cards -->
  <div class="grid grid-cols-1 md:grid-cols-3 gap-6 mb-8">
    <div class="bg-white rounded-lg shadow-md p-6">
      <div class="flex items-center">
        <div class="p-3 rounded-full bg-blue-100 text-blue-600 mr-4">
          <i class="fas fa-book text-2xl"></i>
        </div>
        <div>
          <p class="text-gray-500">Enrolled Classes</p>
          <p class="text-2xl font-bold">{{enrolledClassesCount}}</p>
        </div>
      </div>
    </div>

    <div class="bg-white rounded-lg shadow-md p-6">
      <div class="flex items-center">
        <div class="p-3 rounded-full bg-green-100 text-green-600 mr-4">
          <i class="fas fa-check-circle text-2xl"></i>

```

```

</div>
<div>
  <p class="text-gray-500">Attendance Rate</p>
  <p class="text-2xl font-bold">{{attendanceRate}}%</p>
</div>
</div>
</div>

```

```

<div class="bg-white rounded-lg shadow-md p-6">
  <div class="flex items-center">
    <div class="p-3 rounded-full bg-purple-100 text-purple-600 mr-4">
      <i class="fas fa-calendar-check text-2xl"></i>
    </div>
    <div>
      <p class="text-gray-500">Upcoming Sessions</p>
      <p class="text-2xl font-bold">{{upcomingSessionsCount}}</p>
    </div>
  </div>
</div>
</div>
</div>

```

```

<!-- Quick Actions -->
<div class="bg-white rounded-lg shadow-md p-6 mb-8">
  <h2 class="text-xl font-semibold mb-4">Quick Actions</h2>
  <div class="grid grid-cols-1 md:grid-cols-4 gap-4">
    <a href="/attendance/scan" class="bg-red-600 hover:bg-red-700 text-white text-center py-3 px-4 rounded">
      <i class="fas fa-qrcode mr-2"></i> Scan QR Code
    </a>
    <a href="/classes/join" class="bg-blue-600 hover:bg-blue-700 text-white text-center py-3 px-4 rounded">
      <i class="fas fa-plus-circle mr-2"></i> Join a Class
    </a>
  </div>
</div>

```

```
<a href="/classes/my-joined-classes" class="bg-green-600 hover:bg-green-700 text-white text-center py-3 px-4 rounded">
```

```
<i class="fas fa-book mr-2"></i> View My Classes
```

```
</a>
```

```
<a href="/attendance/history" class="bg-purple-600 hover:bg-purple-700 text-white text-center py-3 px-4 rounded">
```

```
<i class="fas fa-history mr-2"></i> Attendance History
```

```
</a>
```

```
</div>
```

```
</div>
```

```
<!-- Recent Classes -->
```

```
<div class="bg-white rounded-lg shadow-md p-6 mb-8">
```

```
<h2 class="text-xl font-semibold mb-4">My Classes</h2>
```

```
{{#if classes.length}}
```

```
<div class="overflow-x-auto">
```

```
<table class="min-w-full bg-white">
```

```
<thead>
```

```
<tr>
```

```
<th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">Class Name</th>
```

```
<th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">Faculty</th>
```

```
<th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">Semester</th>
```

```
</tr>
```

```
</thead>
```

```
<tbody class="divide-y divide-gray-200">
```

```
{{#each classes}}
```

```
<tr>
```

```
<td class="py-3 px-4">{{this.name}}</td>
```

```
<td class="py-3 px-4">{{this.faculty.name}}</td>
```

```
<td class="py-3 px-4">{{this.semester}}</td>
```

```
</tr>
```

```

        {{/each}}
    </tbody>
</table>
</div>
{{else}}
    <div class="bg-yellow-100 text-yellow-800 p-4 rounded">
        <p>You haven't joined any classes yet. <a href="/classes/join" class="font-bold underline">Join a class</a> to get started.</p>
    </div>
{{/if}}
</div>

```

```

<!-- Recent Attendance -->
<div class="bg-white rounded-lg shadow-md p-6">
    <h2 class="text-xl font-semibold mb-4">Recent Attendance</h2>
    {{#if recentAttendance.length}}
    <div class="overflow-x-auto">
        <table class="min-w-full bg-white">
            <thead>
                <tr>
                    <th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">Class</th>
                    <th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">Session</th>
                    <th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">Date</th>
                    <th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">Status</th>
                    <th class="py-3 px-4 bg-gray-100 text-left text-xs font-semibold text-gray-600 uppercase tracking-wider">Timestamp</th>
                </tr>
            </thead>
            <tbody class="divide-y divide-gray-200">
                {{#each recentAttendance}}

```

```

<tr>
  <td class="py-3 px-4">{{this.session.class.name}}</td>
  <td class="py-3 px-4">{{this.session.topic}}</td>
  <td class="py-3 px-4">{{formatDate this.session.date}}</td>
  <td class="py-3 px-4">{{statusBadge this.status}}</td>
  <td class="py-3 px-4">{{formatDateTime this.timestamp}}</td>
</tr>
{{/each}}
</tbody>
</table>
</div>
<div class="mt-4 text-right">
  <a href="/attendance/history" class="text-blue-600 hover:text-blue-800">
    View Full History <i class="fas fa-arrow-right ml-1"></i>
  </a>
</div>
{{else}}
<div class="bg-yellow-100 text-yellow-800 p-4 rounded">
  <p>No attendance records found. Attend a class session to see your records here.</p>
</div>
{{/if}}
</div>
</div>

```

BACKEND

MARKING ATTENDANCE

```

const express = require('express');
const router = express.Router();

const { ensureAuthenticated, ensureFaculty, ensureAdmin, ensureStudent } =
  require('../middleware/auth');

const { Attendance, Session, Class, User, StudentClass } = require('../models');
const { Op } = require('sequelize');

```



```
const { isQRValid, getAttendanceStatus, getQRTimeRemaining } = require('../utils/qr-timer');
const { markAbsentForExpiredSession, shouldMarkAbsent } = require('../utils/mark-absent');
```

```
// Student: Mark attendance page
```

```
router.get('/mark/:sessionId', ensureStudent, async (req, res) => {
```

```
  try {
```

```
    // Find session by sessionId
```

```
    const session = await Session.findOne({
```

```
      where: { sessionId: req.params.sessionId },
```

```
      include: [
```

```
        {
```

```
          model: Class,
```

```
          include: [
```

```
            {
```

```
              model: User,
```

```
              as: 'students',
```

```
              where: { id: req.user.id },
```

```
              required: false
```

```
            }
```

```
          ]
```

```
        }
```

```
      ]
```

```
    });
```

```
    if (!session) {
```

```
      req.flash('error_msg', 'Invalid session code');
```

```
      return res.redirect('/student/dashboard');
```

```
    }
```

```
    // Check if student is enrolled in this class
```

```
    const isEnrolled = session.class.students.some(student => student.id === req.user.id);
```

```

if (!isEnrolled && req.user.role !== 'admin') {
  req.flash('error_msg', 'You are not enrolled in this class');
  return res.redirect('/student/dashboard');
}

// Check if student has already marked attendance
const existingAttendance = await Attendance.findOne({
  where: {
    sessionId: session.id,
    studentId: req.user.id
  }
});

if (existingAttendance) {
  req.flash('error_msg', 'You have already marked your attendance for this session');
  return res.redirect(/sessions/student-view/${session.id});
}

res.render('student/mark-attendance', {
  session,
  class: session.class,
  title: 'Mark Attendance'
});
} catch (err) {
  console.error(err);
  req.flash('error_msg', 'An error occurred');
  res.redirect('/student/dashboard');
}
});

// Student: Mark attendance by QR
router.post('/mark-by-qr/:sessionId', ensureStudent, async (req, res) => {

```

```

try {
  const { qrCode } = req.body;

  // Check if QR code is provided
  if (!qrCode) {
    req.flash('error_msg', 'QR code is required to mark attendance');
    return res.redirect('/attendance/mark/${req.params.sessionId}');
  }

  // Find session by sessionId
  const session = await Session.findOne({
    where: { sessionId: req.params.sessionId }
  });

  if (!session) {
    req.flash('error_msg', 'Invalid session code');
    return res.redirect('/student/dashboard');
  }

  // Check if student is enrolled in this class
  const studentClass = await StudentClass.findOne({
    where: {
      studentId: req.user.id,
      classId: session.classId
    }
  });

  if (!studentClass && req.user.role !== 'admin') {
    req.flash('error_msg', 'You are not enrolled in this class');
    return res.redirect('/student/dashboard');
  }
}

```

```

// Check if student has already marked attendance
const existingAttendance = await Attendance.findOne({
  where: {
    sessionId: session.id,
    studentId: req.user.id
  }
});

if (existingAttendance) {
  req.flash('error_msg', 'You have already marked your attendance for this session');
  return res.redirect(/sessions/student-view/${session.id});
}

// Check QR code validity and determine attendance status
const qrValid = isQRValid(session, qrCode);
const attendanceStatus = getAttendanceStatus(session, qrValid);

if (!qrValid) {
  req.flash('error_msg', 'QR code has expired. Attendance can only be marked within 10
minutes of session creation. ');
  return res.redirect('/student/dashboard');
}

// Mark attendance with appropriate status
await Attendance.create({
  sessionId: session.id,
  studentId: req.user.id,
  status: attendanceStatus, // 'present' or 'late' based on time elapsed
  timestamp: new Date(),
  ipAddress: req.ip,
  deviceInfo: req.headers['user-agent']
});

```

```

// Prepare status message based on attendance status
let statusMessage = 'Attendance marked successfully';
if (attendanceStatus === 'late') {
  req.flash('warning_msg', 'Attendance marked as LATE (marked between 5-10 minutes
after session creation)');
} else if (attendanceStatus === 'present') {
  req.flash('success_msg', 'Attendance marked as PRESENT (marked within 5 minutes of
session creation)');
}

req.flash('success_msg', 'Attendance marked successfully');
res.redirect(/sessions/student-view/${session.id});
} catch (err) {
  console.error(err);
  req.flash('error_msg', 'An error occurred while marking attendance');
  res.redirect('/student/dashboard');
}
});

// Student: Submit attendance
router.post('/mark/:sessionId', ensureStudent, async (req, res) => {
  try {
    const { location, deviceInfo } = req.body;

    // Check if location is provided
    if (!location) {
      req.flash('error_msg', 'Location is required to mark attendance');
      return res.redirect(/attendance/mark/${req.params.sessionId});
    }

    // Find session by sessionId
    const session = await Session.findOne({

```

```

    where: { sessionId: req.params.sessionId }
  });

  if (!session) {
    req.flash('error_msg', 'Invalid session code');
    return res.redirect('/student/dashboard');
  }

  // Check if student is enrolled in this class
  const studentClass = await StudentClass.findOne({
    where: {
      studentId: req.user.id,
      classId: session.classId
    }
  });

  if (!studentClass && req.user.role !== 'admin') {
    req.flash('error_msg', 'You are not enrolled in this class');
    return res.redirect('/student/dashboard');
  }

  // Check if student has already marked attendance
  const existingAttendance = await Attendance.findOne({
    where: {
      sessionId: session.id,
      studentId: req.user.id
    }
  });

  if (existingAttendance) {
    req.flash('error_msg', 'You have already marked your attendance for this session');
    return res.redirect(/sessions/student-view/${session.id});
  }

```

```

}

// Check QR code validity and determine attendance status
const qrValid = isQRValid(session);

if (!qrValid) {
  req.flash('error_msg', 'QR code has expired. Attendance can only be marked within 10
minutes of session creation. ');
  return res.redirect('/student/dashboard');
}

// Determine attendance status based on time elapsed
const attendanceStatus = getAttendanceStatus(session);

// Mark attendance with appropriate status
await Attendance.create({
  sessionId: session.id,
  studentId: req.user.id,
  status: attendanceStatus, // 'present' or 'late' based on time elapsed
  timestamp: new Date(),
  location: location || 'Not provided',
  ipAddress: req.ip,
  deviceInfo: deviceInfo || req.headers['user-agent']
});

// Prepare status message based on attendance status
let statusMessage = 'Attendance marked successfully';
if (attendanceStatus === 'late') {
  req.flash('warning_msg', 'Attendance marked as LATE (marked between 5-10 minutes
after session creation)');
} else if (attendanceStatus === 'present') {
  req.flash('success_msg', 'Attendance marked as PRESENT (marked within 5 minutes of
session creation)');
}

```

```

    }

    req.flash('success_msg', 'Attendance marked successfully');
    res.redirect(/sessions/student-view/${session.id});
  } catch (err) {
    console.error(err);
    req.flash('error_msg', 'An error occurred while marking attendance');
    res.redirect('/student/dashboard');
  }
});

```

```

// Student: View attendance history
router.get('/history', ensureStudent, async (req, res) => {
  try {
    // Get filter parameters
    const { classId, status, startDate, endDate } = req.query;

    // Build where clause for attendance records
    const attendanceWhere = { studentId: req.user.id };

    // Apply status filter if provided
    if (status) {
      attendanceWhere.status = status;
    }

    // Get all classes the student is enrolled in
    const studentClasses = await StudentClass.findAll({
      where: { studentId: req.user.id }
    });

    // Get class IDs the student is enrolled in
    const classIds = studentClasses.map(sc => sc.classId);
  }
});

```



```

if (classIds.length === 0) {
  // Student is not enrolled in any classes
  return res.render('student/attendance-history', {
    attendanceRecords: [],
    summary: {
      totalSessions: 0,
      presentCount: 0,
      lateCount: 0,
      absentCount: 0,
      attendancePercentage: 0
    },
    classes: [],
    filters: {
      classId: classId || "",
      status: status || "",
      startDate: startDate || "",
      endDate: endDate || ""
    },
    title: 'Attendance History'
  });
}

// Build session where clause
const sessionWhere = {};

// Apply class filter if provided, otherwise use all enrolled classes
if (classId && classIds.includes(parseInt(classId))) {
  sessionWhere.classId = classId;
} else {
  sessionWhere.classId = { [Op.in]: classIds };
}

```

```

// Apply date filters if provided
if (startDate) {
  sessionWhere.date = {
    ...sessionWhere.date,
    [Op.gte]: new Date(startDate)
  };
}

if (endDate) {
  sessionWhere.date = {
    ...sessionWhere.date,
    [Op.lte]: new Date(endDate)
  };
}

// Get all sessions for the filtered classes
const allSessions = await Session.findAll({
  where: sessionWhere,
  include: [{ model: Class }]
});

// Get all attendance records for the student with the applied filters
const attendanceWhereFinal = { ...attendanceWhere };
if (allSessions.length > 0) {
  attendanceWhereFinal.sessionId = { [Op.in]: allSessions.map(s => s.id) };
} else {
  // No sessions match the criteria
  attendanceWhereFinal.sessionId = -1; // Ensure no records are returned
}

const attendanceRecords = await Attendance.findAll({

```

```

where: attendanceWhereFinal,
include: [
  {
    model: Session,
    include: [{ model: Class }]
  }
],
order: [['timestamp', 'DESC']]
});

```

```

// Calculate summary statistics
const totalSessions = allSessions.length;

const presentCount = attendanceRecords.filter(record => record.status ===
'present').length;

const lateCount = attendanceRecords.filter(record => record.status === 'late').length;
const absentCount = totalSessions - (presentCount + lateCount);

// Calculate attendance percentage
const attendancePercentage = totalSessions > 0
  ? Math.round(((presentCount + lateCount) / totalSessions) * 100)
  : 0;

// Find sessions where the student was absent
const attendedSessionIds = attendanceRecords.map(record => record.sessionId);

const absentSessions = allSessions.filter(session
=> !attendedSessionIds.includes(session.id));

// Prepare summary object
const summary = {
  totalSessions,
  presentCount,
  lateCount,
  absentCount,

```

```

    attendancePercentage
  };

  // Get all classes for filter dropdown
  const classes = await Class.findAll({
    where: { id: { [Op.in]: classIds } }
  });

  res.render('student/attendance-history', {
    attendanceRecords,
    absentSessions,
    summary,
    classes,
    filters: {
      classId: classId || "",
      status: status || "",
      startDate: startDate || "",
      endDate: endDate || ""
    },
    title: 'Attendance History'
  });
} catch (err) {
  console.error('Error in attendance history:', err);
  req.flash('error_msg', 'An error occurred while fetching attendance history');
  res.redirect('/student/dashboard');
}
});

// Faculty: View attendance for a session
router.get('/session/:id', ensureFaculty, async (req, res) => {
  try {
    // Find session

```

```

const session = await Session.findByPk(req.params.id, {
  include: [
    {
      model: Class,
      where: { facultyId: req.user.id },
      required: true
    }
  ]
});

if (!session && req.user.role !== 'admin') {
  req.flash('error_msg', 'Session not found or you do not have permission');
  return res.redirect('/classes/my-classes');
}

// Get all students in the class
const classObj = await Class.findByPk(session.classId, {
  include: [
    {
      model: User,
      as: 'students',
      through: { attributes: [] }
    }
  ]
});

// Get attendance records for this session
const attendanceRecords = await Attendance.findAll({
  where: { sessionId: session.id },
  include: [
    {
      model: User,

```

```

      as: 'student',
      attributes: ['id', 'name', 'email']
    }
  ]
});

```

```

// Create a map of student IDs to attendance records

```

```

const attendanceMap = {};
attendanceRecords.forEach(record => {
  attendanceMap[record.studentId] = record;
});

```

```

// Prepare data for rendering

```

```

const studentsWithAttendance = classObj.students.map(student => {
  return {
    id: student.id,
    name: student.name,
    email: student.email,
    attendance: attendanceMap[student.id] || null
  };
});

```

```

// Calculate attendance stats

```

```

const totalStudents = classObj.students.length;
const presentCount = attendanceRecords.filter(record => record.status ===
'present').length;
const lateCount = attendanceRecords.filter(record => record.status === 'late').length;
const absentCount = totalStudents - (presentCount + lateCount);

```

```

// Calculate separate percentages for present and late students

```

```

const presentPercentage = totalStudents > 0 ? Math.round(presentCount / totalStudents *
100) : 0;
const latePercentage = totalStudents > 0 ? Math.round(lateCount / totalStudents * 100) : 0;

```

```
const attendancePercentage = totalStudents > 0 ? Math.round((presentCount + lateCount) /
totalStudents * 100) : 0;
```

```
res.render('faculty/session-attendance', {
  session,
  class: classObj,
  studentsWithAttendance,
  presentCount,
  lateCount,
  absentCount,
  presentPercentage,
  latePercentage,
  attendancePercentage,
  title: Attendance: ${session.topic}
});
} catch (err) {
  console.error(err);
  req.flash('error_msg', 'An error occurred while fetching attendance records');
  res.redirect('/classes/my-classes');
}
});
```

```
// Faculty: Mark attendance manually
router.post('/mark-manual', ensureFaculty, async (req, res) => {
  try {
    const { sessionId, studentId, status } = req.body;

    // Find session
    const session = await Session.findByPk(sessionId, {
      include: [
        {
          model: Class,
```

```

        where: { facultyId: req.user.id },
        required: true
    }
]
});

if (!session && req.user.role !== 'admin') {
    req.flash('error_msg', 'Session not found or you do not have permission');
    return res.redirect('/classes/my-classes');
}

// Check if student is enrolled in this class
const studentClass = await StudentClass.findOne({
    where: {
        studentId,
        classId: session.classId
    }
});

if (!studentClass) {
    req.flash('error_msg', 'Student is not enrolled in this class');
    return res.redirect(/attendance/session/${sessionId});
}

// Check if attendance record exists
const existingAttendance = await Attendance.findOne({
    where: {
        sessionId,
        studentId
    }
});

```



```

if (existingAttendance) {
  // Update existing record
  await existingAttendance.update({
    status,
    editedBy: req.user.id,
    editedAt: new Date()
  });
} else {
  // Create new record
  await Attendance.create({
    sessionId,
    studentId,
    status,
    timestamp: new Date(),
    location: 'Marked manually by faculty',
    editedBy: req.user.id,
    editedAt: new Date()
  });
}

req.flash('success_msg', 'Attendance updated successfully');
res.redirect(/attendance/session/${sessionId});
} catch (err) {
  console.error(err);
  req.flash('error_msg', 'An error occurred while updating attendance');
  res.redirect(/attendance/session/${req.body.sessionId});
}
});

// Faculty: Bulk mark attendance
router.post('/bulk-mark', ensureFaculty, async (req, res) => {
  try {

```

```

const { sessionId, status } = req.body;

// Find session
const session = await Session.findByPk(sessionId, {
  include: [
    {
      model: Class,
      where: { facultyId: req.user.id },
      required: true,
      include: [
        {
          model: User,
          as: 'students',
          attributes: ['id'],
          through: { attributes: [] }
        }
      ]
    }
  ]
});

if (!session && req.user.role !== 'admin') {
  req.flash('error_msg', 'Session not found or you do not have permission');
  return res.redirect('/classes/my-classes');
}

// Get all students in the class
const studentIds = session.class.students.map(student => student.id);

// Get students who already have attendance records
const existingAttendance = await Attendance.findAll({
  where: { sessionId: session.id },

```

```

    attributes: ['studentId']
  });

const markedStudentIds = existingAttendance.map(record => record.studentId);

// Filter out students who already have attendance records
const unmarkedStudentIds = studentIds.filter(id => !markedStudentIds.includes(id));

// Create attendance records for unmarked students
if (unmarkedStudentIds.length > 0) {
  const attendanceRecords = unmarkedStudentIds.map(studentId => ({
    sessionId: session.id,
    studentId,
    status,
    timestamp: new Date(),
    location: 'Marked in bulk by faculty',
    editedBy: req.user.id,
    editedAt: new Date()
  }));

  await Attendance.bulkCreate(attendanceRecords);

  req.flash('success_msg', Attendance marked as ${status} for
  ${unmarkedStudentIds.length} students);
} else {
  req.flash('info_msg', 'All students already have attendance records');
}

res.redirect(/attendance/session/${session.id});
} catch (err) {
  console.error(err);
  req.flash('error_msg', 'An error occurred while marking attendance in bulk');
}

```

```

    res.redirect('/classes/my-classes');
  }
});

// Faculty: Edit attendance
router.put('/edit/:id', ensureFaculty, async (req, res) => {
  try {
    const { status, location, timestamp } = req.body;

    // Find attendance record
    const attendance = await Attendance.findByPk(req.params.id, {
      include: [
        {
          model: Session,
          include: [
            {
              model: Class,
              where: { facultyId: req.user.id },
              required: true
            }
          ]
        }
      ]
    });

    if (!attendance && req.user.role !== 'admin') {
      req.flash('error_msg', 'Attendance record not found or you do not have permission');
      return res.redirect('/classes/my-classes');
    }

    // Update attendance record
    await attendance.update({

```

```

    status,
    location,
    timestamp: new Date(timestamp),
    editedBy: req.user.id,
    editedAt: new Date()
  });

  req.flash('success_msg', 'Attendance record updated successfully');
  res.redirect('/attendance/session/${attendance.sessionId}');
} catch (err) {
  console.error(err);
  req.flash('error_msg', 'An error occurred while updating attendance record');
  res.redirect('/classes/my-classes');
}
});

// Faculty: Delete attendance
router.delete('/delete/:id', ensureFaculty, async (req, res) => {
  try {
    // Find attendance record
    const attendance = await Attendance.findPk(req.params.id, {
      include: [
        {
          model: Session,
          include: [
            {
              model: Class,
              where: { facultyId: req.user.id },
              required: true
            }
          ]
        }
      ]
    })
  }

```

```

    ]
  });

  if (!attendance && req.user.role !== 'admin') {
    req.flash('error_msg', 'Attendance record not found or you do not have permission');
    return res.redirect('/classes/my-classes');
  }

  const sessionId = attendance.sessionId;

  // Delete attendance
  await attendance.destroy();

  req.flash('success_msg', 'Attendance record deleted successfully');
  res.redirect(/attendance/session/${sessionId});
} catch (err) {
  console.error(err);
  req.flash('error_msg', 'An error occurred while deleting attendance record');
  res.redirect('/classes/my-classes');
}
});

// Admin: View all attendance records
router.get('/all', ensureAdmin, async (req, res) => {
  try {
    // Get query parameters for filtering
    const { classId, sessionId, studentId, startDate, endDate } = req.query;

    // Build where clause
    const whereClause = {};
    if (sessionId) whereClause.sessionId = sessionId;
    if (studentId) whereClause.studentId = studentId;

```

```
// Date range filter
if (startDate && endDate) {
  whereClause.timestamp = {
    [Op.between]: [new Date(startDate), new Date(endDate)]
  };
} else if (startDate) {
  whereClause.timestamp = {
    [Op.gte]: new Date(startDate)
  };
} else if (endDate) {
  whereClause.timestamp = {
    [Op.lte]: new Date(endDate)
  };
}
```

```
// Get attendance records
const attendanceRecords = await Attendance.findAll({
  where: whereClause,
  include: [
    {
      model: Session,
      include: [
        {
          model: Class,
          where: classId ? { id: classId } : {}
        }
      ]
    },
    {
      model: User,
      as: 'student',
```

```

      attributes: ['id', 'name', 'email']
    },
    {
      model: User,
      as: 'editor',
      attributes: ['id', 'name', 'email']
    }
  ],
  order: [['timestamp', 'DESC']]
});

// Get all classes for filter dropdown
const classes = await Class.findAll();

// Get all sessions for filter dropdown
const sessions = await Session.findAll({
  where: classId ? { classId } : {},
  include: [
    {
      model: Class
    }
  ]
});

// Get all students for filter dropdown
const students = await User.findAll({
  where: { role: 'student' }
});

res.render('admin/attendance', {
  attendanceRecords,
  classes,

```



```

    sessions,
    students,
    filters: {
      classId,
      sessionId,
      studentId,
      startDate,
      endDate
    },
    title: 'All Attendance Records'
  });
} catch (err) {
  console.error(err);
  req.flash('error_msg', 'An error occurred while fetching attendance records');
  res.redirect('/admin/dashboard');
}
});

// Export attendance records
// Faculty: Mark absent students for expired sessions
router.post('/mark-absent/:sessionId', ensureFaculty, async (req, res) => {
  try {
    const sessionId = req.params.sessionId;

    // Find session
    const session = await Session.findByPk(sessionId, {
      include: [{ model: Class }]
    });

    if (!session) {
      req.flash('error_msg', 'Session not found');
      return res.redirect('/faculty/dashboard');
    }
  }
});

```

```

}

// Check if the faculty member is the owner of the class
if (session.class.facultyId !== req.user.id && req.user.role !== 'admin') {
  req.flash('error_msg', 'You are not authorized to manage this session');
  return res.redirect('/faculty/dashboard');
}

// Check if QR code has expired
if (!shouldMarkAbsent(session)) {
  req.flash('error_msg', 'Cannot mark absent students yet. QR code is still valid. ');
  return res.redirect(/sessions/details/${sessionId});
}

// Mark absent students
const result = await markAbsentForExpiredSession(sessionId);

if (result.success) {
  if (result.count > 0) {
    req.flash('success_msg', `${result.count} students marked as absent for this session);
  } else {
    req.flash('info_msg', 'All students have already been marked for this session');
  }
} else {
  req.flash('error_msg', result.message || 'Failed to mark absent students');
}

res.redirect(/sessions/details/${sessionId});
} catch (err) {
  console.error('Error marking absent students:', err);
  req.flash('error_msg', 'An error occurred while marking absent students');
  res.redirect('/faculty/dashboard');
}

```

```
}  
});
```

```
router.get('/export', ensureAdmin, async (req, res) => {  
  try {  
    // Get query parameters for filtering  
    const { classId, sessionId, studentId, startDate, endDate, format } = req.query;  
  
    // Build where clause  
    const whereClause = {};  
    if (sessionId) whereClause.sessionId = sessionId;  
    if (studentId) whereClause.studentId = studentId;  
  
    // Date range filter  
    if (startDate && endDate) {  
      whereClause.timestamp = {  
        [Op.between]: [new Date(startDate), new Date(endDate)]  
      };  
    } else if (startDate) {  
      whereClause.timestamp = {  
        [Op.gte]: new Date(startDate)  
      };  
    } else if (endDate) {  
      whereClause.timestamp = {  
        [Op.lte]: new Date(endDate)  
      };  
    }  
  
    // Get attendance records  
    const attendanceRecords = await Attendance.findAll({  
      where: whereClause,  
      include: [  

```

```

    {
      model: Session,
      include: [
        {
          model: Class,
          where: classId ? { id: classId } : {}
        }
      ]
    },
    {
      model: User,
      as: 'student',
      attributes: ['id', 'name', 'email']
    }
  ],
  order: [['timestamp', 'DESC']]
});

// Format data for export
const exportData = attendanceRecords.map(record => {
  return {
    'Student Name': record.student.name,
    'Student Email': record.student.email,
    'Class': record.session.class.name,
    'Session Topic': record.session.topic,
    'Session Date': record.session.date.toISOString().split('T')[0],
    'Status': record.status,
    'Timestamp': record.timestamp.toISOString(),
    'Location': record.location || 'Not provided'
  };
});

```

```

// Generate CSV
if (format === 'csv') {
  const csvHeader = Object.keys(exportData[0]).join(',') + '\n';
  const csvRows = exportData.map(record => {
    return Object.values(record).map(value => {
      // Wrap values with commas in quotes
      return typeof value === 'string' && value.includes(',') ? `${value}` : value;
    }).join(',');
  }).join('\n');

  const csv = csvHeader + csvRows;

  res.setHeader('Content-Type', 'text/csv');
  res.setHeader('Content-Disposition', 'attachment; filename=attendance_export.csv');
  res.send(csv);
} else {
  // Default to JSON
  res.setHeader('Content-Type', 'application/json');
  res.setHeader('Content-Disposition', 'attachment; filename=attendance_export.json');
  res.send(JSON.stringify(exportData, null, 2));
}
} catch (err) {
  console.error(err);
  req.flash('error_msg', 'An error occurred while exporting attendance records');
  res.redirect('/attendance/all');
}
});

// Student: QR Code Scanner Page
router.get('/scan', ensureStudent, (req, res) => {
  try {
    res.render('student/scan-qr', {

```

```

    title: 'Scan QR Code'
  });
} catch (err) {
  console.error('Error loading QR scanner page:', err);
  req.flash('error_msg', 'An error occurred while loading the QR scanner');
  res.redirect('/student/dashboard');
}
});

```

```

// Student: Mark attendance via QR code with location data
router.post('/mark-by-qr', ensureStudent, async (req, res) => {
  try {
    const { sessionId, latitude, longitude, accuracy, deviceInfo } = req.body;

    // Validate required fields
    if (!sessionId || !latitude || !longitude) {
      return res.status(400).json({
        success: false,
        message: 'Session ID and location data are required'
      });
    }
  }

```

```

// Find session by sessionId
const session = await Session.findOne({
  where: { sessionId },
  include: [{ model: Class }]
});

```

```

if (!session) {
  return res.status(404).json({
    success: false,
    message: 'Invalid session code or session not found'
  });
}

```

```

    });
}

// Check if student is enrolled in this class
const studentClass = await StudentClass.findOne({
  where: {
    studentId: req.user.id,
    classId: session.classId
  }
});

if (!studentClass && req.user.role !== 'admin') {
  return res.status(403).json({
    success: false,
    message: 'You are not enrolled in this class'
  });
}

// Check if student has already marked attendance
const existingAttendance = await Attendance.findOne({
  where: {
    sessionId: session.id,
    studentId: req.user.id
  }
});

if (existingAttendance) {
  return res.status(400).json({
    success: false,
    message: 'You have already marked your attendance for this session'
  });
}

```

```

// Format location data
const locationData = Lat: ${latitude}, Long: ${longitude}, Accuracy: ${accuracy}m;

// Check QR code validity and determine attendance status
const qrValid = isQRValid(session);

if (!qrValid) {
  return res.status(400).json({
    success: false,
    message: 'QR code has expired. Attendance can only be marked within 10 minutes of
session creation.'
  });
}

// Determine attendance status based on time elapsed
const attendanceStatus = getAttendanceStatus(session);

// Mark attendance with appropriate status
const attendance = await Attendance.create({
  sessionId: session.id,
  studentId: req.user.id,
  status: attendanceStatus, // 'present' or 'late' based on time elapsed
  timestamp: new Date(),
  location: locationData,
  ipAddress: req.ip,
  deviceInfo: deviceInfo || req.headers['user-agent']
});

// Prepare status message based on attendance status
let statusMessage = 'Attendance marked successfully';
if (attendance.status === 'late') {

```



```

    statusMessage = 'Attendance marked as LATE (marked between 5-10 minutes after
session creation)';

    } else if (attendance.status === 'present') {

        statusMessage = 'Attendance marked as PRESENT (marked within 5 minutes of session
creation)';

    }

return res.status(200).json({
    success: true,
    message: statusMessage,
    data: {
        className: session.class.name,
        sessionTopic: session.topic,
        timestamp: attendance.timestamp,
        status: attendance.status,
        timeRemaining: getQRTimeRemaining(session)
    }
});
} catch (err) {
    console.error('Error marking attendance via QR:', err);
    return res.status(500).json({
        success: false,
        message: 'An error occurred while marking attendance'
    });
}
});

module.exports = router;

```

CHAPTER 8

ANALYSIS AND FINDINGS

LOGIN PAGE

fig 7 : LOGIN PAGE

STUDENT DASHBOARD

CLASS NAME	FACULTY	SEMESTER
Database management System	teacher	4
operating systems	teacher	3
Data structures and algorithms	teacher	3
DAA	teacher	4

fig 8 : STUDENT DASHBOARD

FACULTY DASHBOARD

CLASS NAME	SECTION	SEMESTER	CLASS CODE	STUDENTS	ACTIONS
Database management System	T1	4	Class Code: CL-5380WWV	3	View Add Session
operating systems	T1	3	Class Code: CL-8447037	1	View Add Session
Data structures and algorithms	T1	3	Class Code: CL-7499H3M	1	View Add Session
DAA	T1	4	Class Code: CL-88294UJ	2	View Add Session

fig 9 : FACULTY DASHBOARD

FRONTEND (STUDENT)

The screenshot shows the 'Join a Class' form in the QR Attendance System. The form has a blue header with the title 'Join a Class'. Below the header is a text input field for 'Class Code' with a placeholder text 'Enter the class code shared by your faculty'. There are two buttons: 'Join Class' and 'Cancel'. Below the form is a section titled 'How to Join a Class' with a list of four steps: 1. Ask your faculty for the class code, 2. Enter the class code in the field above, 3. Click 'Join Class' to enroll in the class, and 4. Once joined, you'll be able to view class details and mark attendance.

fig 10 : JOIN A CLASS

The screenshot shows the 'My Classes' page in the QR Attendance System. The page has a blue header with the title 'My Classes' and a 'Join New Class' button. Below the header are five class cards. Each card has a blue header with the class name and a 'Leave Class' button. The classes are: Database management System (T1 - 4), operating systems (T1 - 3), Data structures and algorithms (T1 - 3), DAA (T1 - 4), and java lectures (T1 - 3). Each card also displays the faculty name 'teacher' and the class code.

fig 11 : STUDENT CLASSES

The screenshot shows the 'Attendance History' page in the QR Attendance System. The page has a blue header with the title 'Attendance History' and a 'Back to Dashboard' button. Below the header is a 'Filter Records' section with dropdown menus for 'Class' (All Classes) and 'Status' (All Statuses), and input fields for 'Start Date' (09-05-2025) and 'End Date' (09-05-2025). There are 'Reset' and 'Apply Filters' buttons. Below the filter section is an 'Attendance Summary' section with four cards: 'Total Sessions' (46), 'Present' (8), 'Late' (2), and 'Absent' (36). A progress bar shows the 'Overall Attendance Rate: 22%' and a text '8 / 46 sessions attended'. At the bottom, there is a 'Missed Classes' section with a '36 Sessions' indicator.

fig 12 : STUDENT ATTENDANCE

FRONTEND (QR CODE SCANNING)

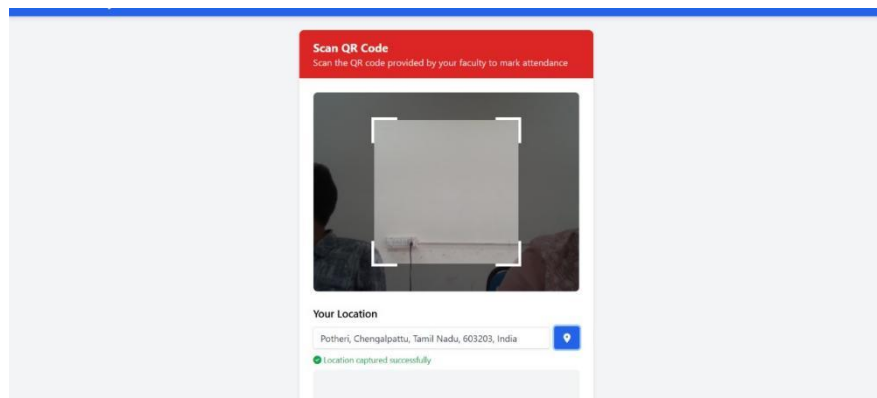


fig 13 : QR CODE SCANNING

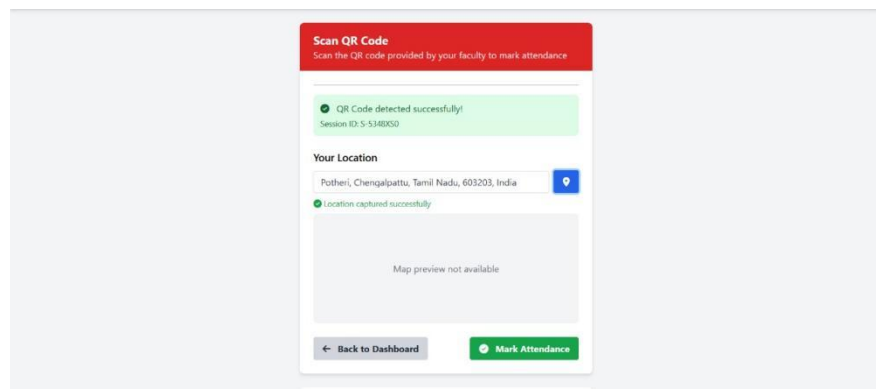


fig 14 : ATTENDANCE SUBMISSION

FRONTEND (FACULTY)

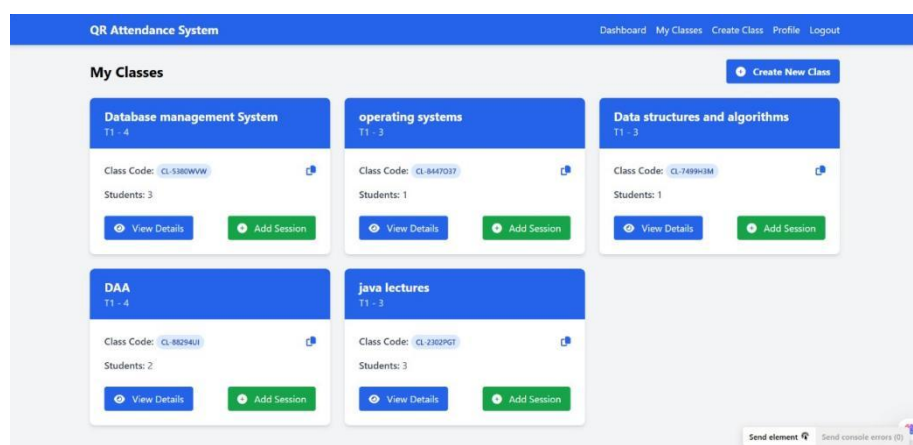


fig 15 : FACULTY CLASS DASHBOARD

The screenshot shows a web application titled "QR Attendance System". The top navigation bar includes links for "Dashboard", "My Classes", "Create Class", "Profile", and "Logout". The main content area features a "Create New Class" modal form. This form contains three input fields labeled "Class Name", "Section", and "Semester". Below these fields are two buttons: "Create Class" and "Cancel". The footer of the page displays the copyright notice "© 2025 QR Code-Based Attendance Management System" and a small utility bar with "Send element" and "Send console errors (0)" options.

QR Attendance System

Dashboard My Classes Create Class Profile Logout

Create New Class

Class Name

Section

Semester

Create Class Cancel

© 2025 QR Code-Based Attendance Management System

Send element Send console errors (0)

fig 15 : FACULTY CREATING NEW CLASS

CHAPTER 9

CONCLUSIONS AND REFERENCES

9.1 CONCLUSION

The QR Code-Based Attendance Management System represents a comprehensive solution for modernizing attendance tracking in educational institutions. This project successfully integrates several key database concepts and technologies to create a robust, scalable, and user-friendly system.

KEY ACHIEVEMENTS

- **Normalized Database Design:** The system implements a fully normalized database schema (achieving 3NF and BCNF) with five well-structured tables (users, classes, sessions, student_classes, and attendances) that minimize redundancy and maintain data integrity.
- **ACID Compliance:** The implementation ensures Atomicity, Consistency, Isolation, and Durability through proper transaction management, constraints, and MySQL's InnoDB engine capabilities.
- **Advanced SQL Features:** The project leverages advanced database features including stored procedures, triggers, views, and cursors for efficient data manipulation and reporting.
- **Concurrency Control:** The system effectively handles concurrent operations through appropriate isolation levels, locking mechanisms, and transaction boundaries, ensuring data consistency even during peak usage.
- **QR Code Integration:** The innovative use of QR codes with a 5-minute time limit provides a secure and efficient method for attendance marking, reducing fraud and simplifying the process for both students and faculty.
- **Role-Based Access:** The implementation of distinct user roles (student, faculty, admin) with appropriate permissions ensures proper data access control and system security.
- **Comprehensive Reporting:** Through views and stored procedures with cursors, the system provides detailed attendance reports and analytics for decision-making.

TECHNICAL HIGHLIGHTS

- Effective use of foreign key constraints to maintain referential integrity
- Implementation of junction tables for many-to-many relationships
- Timestamp tracking for audit and recovery purposes
- Transaction management for critical operations
- Proper indexing for performance optimization
- Trigger-based business rule enforcement
- Business Value

The system delivers significant value to educational institutions by:

- Reducing administrative overhead in attendance management
- Minimizing attendance fraud through secure QR code implementation
- Providing real-time attendance data for faculty and administrators

- Enabling data-driven decision making through comprehensive reporting
- Improving the student experience through a modern, mobile-friendly interface

FUTURE ENHANCEMENTS

While the current implementation is robust and feature-complete, potential future enhancements could include:

- Integration with learning management systems
- Advanced analytics and predictive modeling for attendance patterns
- Mobile application development for improved accessibility
- Biometric verification as an additional security layer
- Geofencing to ensure attendance is marked within class premises

This QR Code-Based Attendance Management System successfully demonstrates the application of database concepts, transaction management, and concurrency control in solving a real-world problem, resulting in a practical solution that enhances educational processes.

9.2 REFERENCES

- Elmasri, R., & Navathe, S. B. (2016). Fundamentals of Database Systems (7th ed.). Pearson.
- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). Database System Concepts (7th ed.). McGraw-Hill Education.
- MySQL Documentation. (2023). InnoDB and the ACID Model. Retrieved from <https://dev.mysql.com/doc/refman/8.0/en/mysql-acid.html>
- Ramakrishnan, R., & Gehrke, J. (2003). Database Management Systems (3rd ed.). McGraw-Hill.
- Connolly, T., & Begg, C. (2015). Database Systems: A Practical Approach to Design, Implementation, and Management (6th ed.). Pearson.
- Bernstein, P. A., & Newcomer, E. (2009). Principles of Transaction Processing (2nd ed.). Morgan Kaufmann.
- Date, C. J. (2004). An Introduction to Database Systems (8th ed.). Pearson.
- Özsu, M. T., & Valduriez, P. (2020). Principles of Distributed Database Systems (4th ed.). Springer.
- Weikum, G., & Vossen, G. (2002). Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann.
- Gray, J., & Reuter, A. (1993). Transaction Processing: Concepts and Techniques. Morgan Kaufmann.

- Hellerstein, J. M., Stonebraker, M., & Hamilton, J. (2007). Architecture of a Database System. *Foundations and Trends in Databases*, 1(2), 141-259.
- Kemper, A., & Neumann, T. (2014). HyPer: A Hybrid OLTP&OLAP Main Memory Database System Based on Virtual Memory Snapshots. *IEEE 27th International Conference on Data Engineering*.
- Node.js Documentation. (2023). MySQL Integration. Retrieved from <https://nodejs.org/en/docs/>
- Express.js Documentation. (2023). Database Integration. Retrieved from <https://expressjs.com/en/guide/database-integration.html>
- Sequelize ORM Documentation. (2023). Transactions and Hooks. Retrieved from <https://sequelize.org/docs/v6/other-topics/transactions/>
- QR Code Standards ISO/IEC 18004:2015. Information technology — Automatic identification and data capture techniques — QR Code bar code symbology specification.
- Sharma, A., & Bosch, J. (2019). QR Code-Based Attendance Management System: A Case Study. *International Journal of Computer Applications*, 182(39), 33-37.
- Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6), 377-387.
- Bernstein, P. A., Hadzilacos, V., & Goodman, N. (1987). *Concurrency Control and Recovery in Database Systems*. Addison-Wesley.
- Passport.js Documentation. (2023). Authentication Concepts. Retrieved from <https://www.passportjs.org/docs/>