DISPLAY CLASS DIAGRAM FOR COLLEGE MANAGEMENT SYSTEM Manual

display class diagram for college management system is an essential step in designing an effective software solution that streamlines administrative tasks, enhances data management, and improves overall operational efficiency within educational institutions. A class diagram visually represents the structure of the system by illustrating classes, their attributes, methods, and the relationships among them. For a college management system, creating a comprehensive class diagram helps developers and stakeholders understand the system's architecture, identify key components, and facilitate smooth implementation. In this article, we will explore the significance of class diagrams in college management systems, discuss the main components involved, and provide guidance on designing an effective diagram.

Understanding the Importance of Class Diagrams in College Management Systems

What is a Class Diagram?

A class diagram is a type of static structure diagram in UML (Unified Modeling Language) that depicts the system's classes, their properties, behaviors, and relationships. It serves as a blueprint for developers to understand how data is organized and how different entities interact within the system.

Why Use a Class Diagram for a College Management System?

Implementing a college management system involves managing diverse data entities such as students, courses, faculty, departments, and more. A class diagram provides several benefits:

- Clarifies system structure: Offers a visual overview of how entities relate.
- Facilitates communication: Bridges the gap between technical teams and non-technical stakeholders.
- Guides development: Helps in designing databases, user interfaces, and business logic.
- Identifies dependencies: Highlights how classes depend on or interact with each other.
- Aids in maintenance: Simplifies understanding of the system for future updates or debugging.

Key Components of a College Management System Class Diagram

Designing a class diagram for a college management system involves identifying critical entities and their interactions. Typically, the main classes include:

1. Student

Attributes:

- StudentID
- Name
- DateOfBirth
- Address
- PhoneNumber
- Email

Methods:

- RegisterCourse()
- DropCourse()
- ViewTranscript()

2. Faculty

Attributes:

- FacultyID
- Name
- Department
- Designation
- Email
- PhoneNumber

Methods:

- AssignCourse()
- EvaluateStudent()

3. Course

Attributes:

- CourseID
- CourseName
- Credits
- Semester
- Department

Methods:

- AssignInstructor()
- EnrollStudent()

4. Department

Attributes:

- DepartmentID
- DepartmentName
- Building
- HeadOfDepartment

Methods:

- AddCourse()
- RemoveCourse()

5. Enrollment

Attributes:

- EnrollmentID
- StudentID
- CourseID
- EnrollmentDate
- Grade

Methods:

- CalculateGPA()

6. Administrator

Attributes:

- AdminID
- Name
- Email
- PhoneNumber

Methods:

- AddUser()
- RemoveUser()
- ManageDepartments()

Designing the Class Diagram: Step-by-Step Approach

Step 1: Identify the Main Entities

Start by listing all the major entities involved in the college system. These typically include students, faculty, courses, departments, enrollments, and administrators.

Step 2: Define Attributes and Methods

For each entity, determine the relevant attributes (data fields) and methods (functions or behaviors). Focus on what data each entity holds and what operations it performs.

Step 3: Establish Relationships

Identify how classes are related. Common relationships include:

- Associations: e.g., students enroll in courses.
- Aggregations: e.g., a department contains multiple courses.

- Inheritances: e.g., different types of users (students, faculty, admins) may inherit from a common User class.
- Multiplicity: specify how many instances of one class are related to another (e.g., one department offers many courses).

Step 4: Draw the Diagram

Using UML notation, create boxes for each class, listing attributes and methods, and connect them with appropriate relationship lines. Use arrows and labels to specify the nature of relationships.

Sample Class Diagram Overview for a College Management System

Below is a simplified overview of how classes might be related:

- User (abstract class) ? inherits Student, Faculty, Administrator
- Department contains multiple Courses
- Course is taught by Faculty (many-to-one)
- Student enrolls in Courses via Enrollment
- Enrollment links Student and Course, tracks Grade
- Administrator manages Departments, Courses, and Users

This structure ensures clarity and supports the functionalities typical of a college management system.

Advanced Features to Consider in the Class Diagram

When designing a comprehensive class diagram, consider including additional features such as:

- Attendance Management: Classes for tracking attendance records.
- Examinations and Results: Classes for exams, marks, and GPA calculation.
- Fee Management: Classes for handling fee payments, receipts, and dues.
- Library Management: Entities for books, borrowers, loans.
- Notifications: Classes for sending alerts or messages to students and staff.

Including these components makes the system more robust and capable of handling real-world college management needs.

Tools for Creating Class Diagrams

Several tools can aid in designing and visualizing class diagrams effectively:

- Lucidchart
- Microsoft Visio
- StarUML
- draw.io (diagrams.net)
- Enterprise Architect

These tools offer drag-and-drop interfaces, UML templates, and export options to share diagrams with stakeholders.

Conclusion

Creating a detailed display class diagram for a college management system is a foundational step toward building a robust, scalable, and maintainable software solution. By clearly defining classes, their attributes, methods, and relationships, developers can ensure that the system meets the needs of administrative staff, faculty, students, and other stakeholders. Properly crafted class diagrams serve as blueprints that guide database design, system architecture, and application development, ultimately leading to a more efficient and user-friendly college management experience. Whether you are designing a new system or improving an existing one, investing time in a well-structured class diagram is invaluable for success.

Display Class Diagram for College Management System: An In-Depth Review

A display class diagram for a college management system is a crucial component in the realm of software engineering, especially when designing systems that require clear visualization of data structures and relationships. This diagram not only provides a blueprint for developers but also acts as a communication tool among stakeholders, ensuring everyone has a shared understanding of how the system operates at a structural level. In this comprehensive review, we will delve into the significance of class diagrams in college management systems, explore their core components, analyze their features and limitations, and discuss best practices for designing effective display class diagrams.

Understanding the Role of Class Diagrams in College Management Systems

What is a Class Diagram?

A class diagram is a type of static structure diagram in the Unified Modeling Language (UML) that depicts the system's classes, their attributes, operations, and the relationships among objects. It

essentially models the blueprint of the system's data architecture and how different entities interact with each other.

Importance in College Management Systems

College management systems handle various complex data entities such as students, faculty, courses, departments, and administrative activities. A well-designed class diagram:

- Facilitates clarity in system design,
- Serves as documentation for future maintenance,
- Guides developers during implementation,
- Helps in identifying potential design flaws early,
- Assists in aligning system functionalities with institutional requirements.

Core Components of the Display Class Diagram

Classes

Classes are the primary building blocks of the diagram, representing entities like Student, Faculty, Course, Department, Enrollment, etc. Each class encapsulates relevant attributes and behaviors.

Attributes

Attributes define the data held by each class, such as student ID, name, date of birth, course code, etc. Clear attribute specification ensures accurate data management.

Operations (Methods)

Operations specify the functionalities associated with classes, like enroll(), assignInstructor(), or calculateGPA(). Including methods helps in understanding how classes interact with data.

Relationships

Relationships illustrate how classes are connected:

- Associations: e.g., a Student enrolls in many Courses.
- Inheritance (Generalization): e.g., UndergraduateStudent and GraduateStudent inherit from Student.
- Aggregation and Composition: e.g., a Department composes multiple Courses.

- Multiplicity: indicates how many objects participate in a relationship, e.g., one Department has many Courses.

Constraints and Notes

Additional notes or constraints can be added to clarify specific rules, such as prerequisites or enrollment limits.

Designing an Effective Display Class Diagram

Step-by-Step Approach

- Identify Core Entities: List all key participants such as Student, Course, Faculty, Department, etc.
- Determine Relationships: Establish how entities relate, including cardinality and nature.
- Define Attributes and Methods: Specify essential data points and behaviors.
- Use UML Standards: Maintain consistency with UML notation for clarity.
- Validate with Stakeholders: Ensure the diagram aligns with institutional needs and processes.

Best Practices

- Keep it simple: Focus on essential classes and relationships.
- Use clear naming conventions: To avoid ambiguity.
- Incorporate inheritance wisely: To reduce redundancy.
- Maintain consistency: In symbols, line styles, and annotations.
- Iterate and refine: Based on feedback and evolving requirements.

Features and Benefits of a Well-Designed Display Class Diagram

- Clear Visualization: Provides an at-a-glance understanding of system structure.
- Facilitates Communication: Acts as a common reference among developers, designers, and stakeholders.
- Supports System Scalability: Easy to update as new features or entities are added.
- Enhances Maintainability: Simplifies troubleshooting and future enhancements.
- Promotes Reusability: Identifies common attributes and behaviors for reuse.

Key Features

- Modular design with distinct classes.
- Well-defined relationships with multiplicities.
- Use of inheritance for specialized classes.
- Annotated with constraints for business rules.

Common Challenges and Limitations

While class diagrams are invaluable tools, they come with certain limitations:

- Oversimplification: Can omit dynamic behaviors or processes.
- Complexity in Large Systems: Diagrams can become cluttered and hard to interpret.
- Static Viewpoint: Does not portray system behaviors over time.
- Learning Curve: Requires understanding of UML notation.
- Maintenance Overhead: Requires regular updates to reflect system changes.

Strategies to Mitigate Challenges

- Break down large diagrams into modules.
- Use packages to organize classes.
- Complement with sequence or activity diagrams for dynamic behavior.
- Maintain clear documentation alongside diagrams.

Practical Example: Display Class Diagram for College Management System

Imagine a simplified class diagram for a college management system:

- Student class with attributes: studentID, name, dateOfBirth, email.
- Course class with attributes: courseCode, title, credits.
- Faculty class with attributes: facultyID, name, department.
- Department class with attributes: departmentID, name.
- Enrollment relationship between Student and Course with attributes: enrollmentDate, grade.
- Instructor relationship between Faculty and Course.
- Inheritance: UndergraduateStudent and GraduateStudent inherit from Student.
- Aggregation: Department contains multiple Courses.

This diagram would include association lines with multiplicities, such as "a student enrolls in many courses" (1..) and "a course has many students" (0..).

Conclusion and Final Thoughts

A display class diagram for college management system is an essential artifact in the software development lifecycle, providing a structured and visual representation of the system's static architecture. Its importance lies in fostering clear communication, guiding development, and ensuring maintainability. When designed thoughtfully, with adherence to UML standards and consideration of the system's complexity, such diagrams can greatly enhance the efficiency and accuracy of system implementation.

However, developers and analysts should remain aware of the limitations, especially in large or dynamic systems, and complement class diagrams with other UML diagrams to capture behaviors and workflows comprehensively. Ultimately, a well-crafted class diagram acts as a blueprint that ensures the college management system is robust, scalable, and aligned with institutional goals.

In summary, investing time and effort into creating a detailed and accurate display class diagram pays dividends in the long term, leading to smoother development processes and more reliable systems that serve the educational institution's needs effectively.

QuestionAnswer What are the key components included in a display class diagram for a college management system? A display class diagram typically includes classes such as Student, Course, Professor, Department, Enrollment, and Administration, along with their attributes and relationships to illustrate how the system components interact. How does a class diagram help in designing a college management system? It helps visualize the structure of the system by showing classes, their attributes, methods, and relationships, enabling developers to understand data flow, identify modular components, and facilitate efficient system implementation. What are common relationships depicted in a college management system class diagram? Common relationships include associations (e.g., Student enrolls in Course), inheritances (e.g., GraduateStudent inherits from Student), aggregations (e.g., Department contains Professors), and dependencies among classes. Can you provide an example of a class and its attributes in a college management system diagram? Yes, for example, a 'Student' class may have attributes like studentID, name, dateOfBirth, email, and phoneNumber, with methods such as registerCourse() and viewTranscript(). How do you represent inheritance in a display class diagram for a college management system? Inheritance is represented using a solid line with a closed arrowhead pointing from the subclass to the superclass, such as a 'GraduateStudent' class inheriting from 'Student'. What tools can be used to create a display class diagram for a college management system? Tools like UML diagram editors such as Lucidchart, draw.io, Microsoft Visio, StarUML, and Visual Paradigm can be used to create clear and professional class diagrams for college management systems.

Related keywords: college management system, class diagram, UML diagram, system architecture, software design, object-oriented design, class relationships, system modeling, university management, class diagram tools

Thesis Documentation For Payroll System

Thesis Documentation for Payroll System

Thesis documentation for payroll system is a crucial component in the development and presentation of a comprehensive research project or system proposal. It serves as a detailed guide that outlines the processes, methodologies, and technical specifics involved in designing, implementing, and evaluating a payroll system. Proper documentation not only provides clarity for stakeholders but also ensures that the system adheres to best practices, legal standards, and organizational policies. In this article, we will explore the essential elements of thesis documentation for payroll systems, its significance, and best practices to ensure a thorough and effective presentation.

Understanding the Importance of Thesis Documentation for Payroll System

What is a Payroll System?

A payroll system is a vital administrative tool used by organizations to calculate employee wages, deduct applicable taxes, and manage other compensation-related processes. It automates payroll calculations, reduces manual errors, ensures compliance with legal requirements, and streamlines employee payments.

Why is Thesis Documentation Necessary?

Thesis documentation for a payroll system validates the research, development, and implementation phases. It provides a comprehensive record that:

- Demonstrates the system?s functionality and features
- Justifies the choice of technology and methodology
- Guides future maintenance and upgrades
- Serves as an academic requirement for thesis submission
- Facilitates understanding among stakeholders, developers, and users

Key Components of Thesis Documentation for Payroll System

1. Introduction

This section introduces the project, its background, purpose, and scope.

- Background of the Study: Discuss the importance of payroll systems in organizational management.
- Problem Statement: Identify issues with manual payroll processing or existing systems.
- Objectives: Define the main goal and specific objectives of the research.
- Significance of the Study: Explain how the system benefits the organization and users.
- Scope and Limitations: Clarify what the system will cover and constraints faced.

2. Review of Related Literature and Studies

Present existing payroll systems, their features, advantages, and limitations. Include scholarly articles, technical references, and previous research to justify your system?s development.

3. Methodology

Describe the approach and procedures used in developing the payroll system.

- System Development Life Cycle (SDLC): Outline phases such as planning, analysis, design, development, testing, and deployment.
- System Analysis: Gather requirements through interviews, surveys, or questionnaires.
- System Design: Create data flow diagrams (DFD), entity-relationship diagrams (ERD), and interface mockups.
- Technology Stack: Specify programming languages, database systems, frameworks, and tools used.
- Implementation Details: Discuss coding standards, algorithms, and modules.
- Testing Procedures: Describe testing strategies, such as unit testing, integration testing, and user acceptance testing.

4. System Design and Architecture

Provide detailed diagrams and descriptions of the system layout.

- Data Flow Diagrams (DFD): Visualize data movement within the system.
- Entity-Relationship Diagram (ERD): Show database structure and relationships.
- User Interface Design: Mockups and descriptions of screens and user interactions.
- System Architecture: Outline hardware and software components involved.

5. Implementation

Explain how the system was built, including code snippets, database setup, and interface

development.

- Programming Languages Used: e.g., PHP, Java, Python.
- Database Management System: e.g., MySQL, Oracle.
- Features and Modules: List payroll computation, employee management, deductions, reports, etc.
- Security Measures: Access control, data encryption, user authentication.

6. Testing and Evaluation

Detail the testing process and results to ensure system reliability.

- Test Cases: List scenarios and expected outcomes.
- Results and Findings: Summarize test outcomes, bug reports, and fixes.
- User Feedback: Incorporate comments from actual users during testing.
- System Evaluation: Assess performance, usability, and compliance.

7. Summary, Conclusions, and Recommendations

Summarize the project, highlight key findings, and suggest future improvements.

- Summary: Restate the objectives and how they were achieved.
- Conclusions: State whether the system meets the initial goals.
- Recommendations: Propose enhancements, additional features, or areas for further research.

Best Practices in Thesis Documentation for Payroll System

Maintain Clear and Organized Content

Use logical structure, consistent headings, and subheadings. Include tables, diagrams, and flowcharts to aid understanding.

Use Precise and Technical Language

Ensure technical accuracy and clarity. Define technical terms for non-technical readers.

Include Visual Aids

Diagrams, screenshots, and charts make complex information more understandable and engaging.

Proper Referencing and Citations

Document sources of literature, tools, and technologies used according to academic standards.

Thorough Testing and Validation

Provide detailed testing procedures and results to demonstrate system reliability and robustness.

Proofreading and Review

Eliminate grammatical errors and ensure consistency throughout the document.

Conclusion

Thesis documentation for payroll system is a comprehensive record that encapsulates the entire development process, from conception to implementation and testing. It plays a vital role in showcasing the technical and managerial aspects of the system, serving both academic and practical purposes. By adhering to structured guidelines and best practices, developers and researchers can produce an effective and professional thesis that effectively communicates their work, supports future system enhancements, and contributes valuable insights into payroll management solutions.

Keywords: thesis documentation, payroll system, system development, system analysis, system design, system implementation, testing, report writing, academic thesis, payroll management system

Thesis Documentation for Payroll System: An In-Depth Expert Guide

Creating a comprehensive thesis documentation for a payroll system is a crucial step in showcasing the technical, managerial, and operational aspects of a software solution designed to streamline employee compensation processes. As organizations increasingly rely on automated systems to manage salaries, taxes, benefits, and compliance, a well-structured thesis not only demonstrates academic rigor but also provides practical insights into system design, implementation, and evaluation. This article explores the essential components and best practices for developing an effective thesis documentation for a payroll system, aiming to serve as a detailed guide for students, researchers, and developers.

Understanding the Purpose of Thesis Documentation for Payroll System

Before diving into the structural elements, it?s vital to grasp why thesis documentation for a payroll system is important:

- Academic Contribution: Demonstrates understanding of system analysis, design, and implementation processes.
- Practical Reference: Serves as a blueprint for future development or enhancement of payroll systems.
- Project Validation: Provides evidence of feasibility, functionality, and efficiency.
- Stakeholder Communication: Helps stakeholders understand system features, benefits, and technical details.

Key Components of Payroll System Thesis Documentation

A comprehensive thesis documentation typically encompasses several interconnected sections. Each part plays a role in narrating the development journey, technical architecture, and evaluation of the payroll system.

- Title Page and Abstract

- Title Page: Clearly states the project title, author?s name, institution, and submission date.
- Abstract: A concise summary (150-250 words) highlighting the purpose, methods, major findings, and significance of the payroll system.

- Table of Contents and List of Figures/Tables

- Ensures easy navigation through the document.
- Lists all chapters, sections, illustrations, and appendices with page numbers.

- Introduction

This section sets the stage by explaining the background, significance, and scope of the project.

- Background of the Study: Discuss the importance of payroll management, common challenges, and the need for automation.

- Problem Statement: Clearly articulates the issues the system aims to solve, such as manual errors, time consumption, or compliance risks.
- Objectives: Defines the general and specific goals, e.g., ?Develop a user-friendly payroll management system that automates salary computation and report generation.?
- Scope and Limitations: Outlines what the system covers and constraints faced during development.
- Significance of the Study: Highlights benefits to stakeholders, including HR personnel, management, and employees.

- Review of Related Literature and Studies

A critical analysis of existing payroll systems, theories, and technologies.

- Literature Review: Summarizes academic research, articles, and books related to payroll processing, automation, and HR management.
- Related Studies: Discusses similar projects or systems, their strengths, limitations, and innovations.

- Methodology

Describes the approach and procedures used in developing the payroll system.

- System Development Life Cycle (SDLC): Details the chosen methodology (Waterfall, Agile, etc.).
- System Analysis: Gathering user requirements through interviews, questionnaires, or observations.
- System Design: Technical design, including data flow diagrams (DFD), entity-relationship diagrams (ERD), and user interface sketches.
- Implementation: Technologies used (programming language, database management system, frameworks).
- Testing and Evaluation: Types of testing (unit, integration, system, acceptance) and evaluation criteria.

- System Analysis

An in-depth exploration of the current payroll processes and the requirements for the new system.

- Current System Description: Manual procedures, bottlenecks, and issues.
- Needs Analysis: Stakeholder interviews, surveys, or focus groups.
- Requirements Specification: Functional requirements (salary calculation, report generation) and non-functional requirements (security, usability).

- System Design

A detailed blueprint of the proposed payroll system.

- Data Flow Diagram (DFD): Illustrates data movement within the system.
- Entity-Relationship Diagram (ERD): Models data structures and relationships.
- System Architecture: Hardware and software architecture diagram.
- User Interface Design: Sample screens and user experience considerations.
- Process Flowcharts: Visualize processes like salary computation, deduction application, and report generation.

- Implementation

Details on how the system was built, including code snippets, database schemas, and interface layouts.

- Programming Languages and Tools: For example, Java, PHP, Python, or C; MySQL, Oracle, or SQL Server.
- Database Design: Tables, keys, relationships, and normalization.
- Modules and Features:
- Employee Management
- Attendance and Timekeeping
- Salary Computation and Deduction
- Tax and Benefit Calculation
- Report Generation
- User Authentication and Security

- Testing and Validation

Reports on the testing phase, including test cases, results, and issues encountered.

- Test Cases: Inputs, expected outputs, actual results.
- Bug Tracking: Description of bugs and fixes.
- Performance Testing: System efficiency under load.
- User Acceptance Testing (UAT): Feedback from actual users.

- System Evaluation and Recommendations

Assessment of the system?s effectiveness and areas for improvement.

- Evaluation Criteria:
- Accuracy of salary computations
- Ease of use
- Security measures
- System reliability
- User Feedback: Surveys or interviews.
- Recommendations: Suggested enhancements, scalability, integration with other HR modules.

- Summary, Conclusions, and Future Work

- Summarizes key findings.
- Concludes on the success and limitations of the system.
- Suggests future developments such as mobile accessibility, integration with accounting software, or AI-based analytics.

- References

Lists all sources, articles, books, and online resources cited.

- Appendices

Includes supplementary materials:

- Sample forms and reports
- User manuals

- Code snippets
- Data dictionaries

Best Practices in Thesis Documentation for Payroll System

To ensure clarity, professionalism, and comprehensiveness, consider the following best practices:

- Consistency: Use uniform terminology, formatting, and numbering.
- Clarity: Write in clear, precise language suitable for both technical and non-technical readers.
- Visuals: Incorporate diagrams, charts, and screenshots to enhance understanding.
- Documentation Standards: Follow academic and industry standards for citations, diagrams, and coding documentation.
- Validation: Include evidence of testing and validation to showcase system reliability.

Conclusion

Developing a thesis documentation for a payroll system is not just an academic exercise; it is a reflection of the project?s technical depth, practicality, and impact. It requires meticulous planning, systematic analysis, detailed design, rigorous testing, and critical evaluation. A well-crafted thesis serves as a valuable resource for future developers, provides stakeholders with confidence in the system?s capabilities, and contributes to the broader field of HRIS (Human Resource Information System) development.

By adhering to the outlined components and best practices, students and researchers can produce a comprehensive, professional, and informative thesis that effectively communicates their work and advances the understanding of payroll system automation. Whether for academic purposes or real-world application, thorough documentation remains the backbone of successful system development and deployment.

QuestionAnswer What are the essential components to include in a thesis documentation for a payroll system? A comprehensive thesis documentation for a payroll system should include the introduction, problem statement, objectives, literature review, system analysis and design, implementation, testing, results, conclusion, and references. How should I structure the system analysis in my payroll system thesis? The system analysis should detail the current payroll processes, identify gaps or inefficiencies, gather user requirements, and include diagrams like flowcharts and data flow diagrams to illustrate system functionalities. What are the best practices for documenting the system design in a payroll thesis? Best practices include providing detailed design diagrams (UML diagrams, ER diagrams), explaining system architecture, data structures, user interface layouts, and flow of data within the payroll system. How can I demonstrate the implementation process in my payroll system thesis? You should include code snippets,

screenshots of the system in action, step-by-step descriptions of the development process, and explanations of how the system components work together. What testing methodologies should be documented in a payroll system thesis? Document testing methodologies such as unit testing, integration testing, system testing, and user acceptance testing, along with test cases, results, and bug fixes to validate the system's functionality. How do I include security considerations in my payroll system thesis? Discuss security features like user authentication, data encryption, access control, and data privacy measures implemented to protect sensitive payroll information. What are the common challenges faced during payroll system development that should be documented? Challenges may include handling complex calculations, ensuring data accuracy, integrating with other systems, managing user requirements, and maintaining data security. How can I effectively present the results and evaluation of my payroll system thesis? Present results through performance metrics, user feedback, system efficiency analysis, and comparisons with previous manual or existing systems to demonstrate improvements. What references and citations are important in a payroll system thesis documentation? Include references to related literature, technical manuals, industry standards, programming languages used, and any existing payroll system frameworks or best practices. How do I ensure my payroll system thesis documentation is comprehensive and professional? Ensure clarity, consistency, proper formatting, thorough explanations, inclusion of diagrams and visuals, and adherence to academic or institutional guidelines for thesis writing.

Related keywords: thesis documentation, payroll system, payroll management, system design, software development, payroll automation, database design, system implementation, user manual, system analysis

Read the full article online: Thesis Documentation For Payroll System