DSAA2031 - Final Group Project Specification

1. Project Overview

This group project is designed to give you hands-on experience in designing, modeling, and implementing a real-world database system. By working collaboratively, you will apply theoretical knowledge from the course—including conceptual modeling, normalization, SQL programming, and system design—to build a functional database application.

*You will form groups of **3–5 people** and complete a full-cycle database design project, from requirement analysis to SQL implementation and performance testing.*

2. Project Topic Selection and Requirements

2.1 Choosing or Proposing Your Topic

Students may select from **one of the scenarios** listed below, or they can **propose a custom project** that matches the course objectives. All custom proposals must be approved by the instructor or teaching assistant by **April 18, 2025.**

Tip: Select a topic that you find personally interesting or relevant to your future career—it will help you stay engaged and motivated.

Note: The scenarios provided below are not strict blueprints—*you are encouraged to extend, customize, or reinterpret them using your own ideas and domain understanding.* For example, you may introduce new user roles, add more tables, change the workflow. As long as the system meets the complexity and functionality requirements (see Section 2.4), you are free to be creative and demonstrate your design thinking.

2.2 Scenario 1 - Laboratory Project Payroll Management System

Background

In university research labs, students often participate in multiple academic projects led by different faculty members. These projects receive separate budgets, and each project leader is responsible for allocating wages to the students who contribute to their project. However, wage distribution is often inconsistent and manually recorded, leading to errors and unfairness—especially during the year-end budget reconciliation.

Objective

Design a database system that helps lab administrators and teachers manage project budgets, track student participation, and distribute wages fairly and transparently.

Users & Roles

- **Teacher**: Manages projects, assigns students, sets performance scores, and allocates wages.
- **Student**: Views their own project participation, performance evaluation, and wage history.

Core Features

- Each project is led by one teacher and may include multiple students.
- A student can participate in multiple projects.
- Wages cannot exceed the remaining budget of a project.
- Teachers can only manage their own projects and assigned students.
- Students can only view their own wage and participation records.
- The system suggests wage distribution based on student performance and payment history.

2.3 Scenario 2 - Campus Event Registration and Management System

Background

University student clubs and organizations frequently host events such as workshops, lectures, competitions, and social gatherings. However, many events still rely on manual methods (e.g., spreadsheets or online forms) for registration, attendance, and feedback collection. This creates issues with participant tracking, overbooking, and post-event data analysis.

Objective

Build a database system to support the end-to-end process of managing campus events, including event publishing, student registration, participation tracking, and post-event feedback.

Users & Roles

- **Organizer (Club Leader or Event Manager)**: Creates and publishes events, tracks registrants, confirms attendance, views participant feedback.
- **Student (Participant)**: Browses upcoming events, registers for events, checks registration status, submits event feedback.

Core Features

- Events have limited seats; registration should respect the capacity.
- A student can register for multiple events, but cannot register twice for the same event.
- After attending an event, students can submit feedback.
- Organizers can only manage their own events.
- System should prevent duplicate registrations and overbooking.

2.4 Requirement for All Projects

2.4.1 System Design Requirements

Every project must meet the following baseline requirements:

Item	Minimum Requirement
Number of user roles	At least **2 distinct roles** (e.g., Student & Teacher)
Number of database tables	At least **4 relational tables**
Table relationships	Must include **at least 1 many-to-many relationship** (e.g., students and projects)
Data operations	Must include **CRUD** operations for **all tables**
SQL complexity	At least **3 non-trivial SQL queries** using joins, subqueries, aggregation, or grouping
Access control	Implement **basic role-based data access restrictions** (e.g., students can only see their data)

2.4.2 Data Volume

To effectively test your system and simulate real-world behavior, you are expected to populate your database with **a meaningful amount of sample data**.
Minimum Requirements

- At least **300 records in total** (across all tables)
- At least **one table** should contain **over 100 records**
- The data should reflect **realistic scenarios**, such as:
    - Multiple students participating in several projects
    - Monthly wage distributions over a year
    - Projects with different budget statuses

Data Generation Guidelines

- **You are responsible for generating your own data.**
- Data can be generated:
    - **Manually**, for small tables (e.g., 10–20 records)
    - **Programmatically**, using scripts (e.g., Python)

- **With AI assistance**, by prompting a large language model (like ChatGPT) to generate structured sample data under reasonable assumptions

Examples of Reasonable Assumptions

- A student may join 1–3 projects during a semester
- Each project pays students monthly, based on performance
- Wage amount ranges from 200 to 1000 RMB/month
- Project budgets range from 5,000 to 50,000 RMB
- Performance is scored from 1 (poor) to 5 (excellent)

3. Detailed Project Requirements

In this project, you will follow a structured database system development lifecycle, adapted from standard practices. You are expected to understand the purpose of each phase, and implement a basic version of each step based on your project topic.

3.1 Requirement Analysis

In this phase, you will need to clarify the system goals, identify the roles involved, describe user actions, and define how data is processed and stored, based on your selected scenario. This is important for helping you structure your ideas and prepare for design.

1. **Organizational and Role Analysis**

   - Identify **key user roles** in your system (e.g., Student, Teacher, Admin).

   - For each role, describe:
     - Their responsibilities
     - The data they can access (read/write)
     - The functions they can perform
     - ….

2. **Business Process Analysis**
   - Describe the main operations of the system from the user perspective.
   - Clearly describe:

     - Who initiates the action?
     - What steps or data are involved?
     - What is the expected output?

Bonus: Draw a **use-case diagram** or **flowchart** to represent user actions clearly and professionally, this will earn extra credit.

3. Data Flow Diagrams (DFDs) ***(Optional)***

Visualize how data moves through the system, and what processes handle it.

- Provide:
    - Level 0 DFD (Context Diagram) – Shows the system as a whole and its external interactions
    - Level 1 DFD – Break down system into modules (e.g., User Management, Project Assignment, Salary Allocation)
    - Level 2 DFD (optional) – Further detail a specific module if needed

Bonus: Accurate DFDs with consistent notation (processes, data flows, stores, and external entities clearly labeled)

3.2 Conceptual Design

At this stage, you will translate your understanding of the system into a structured **data model** that reflects the real-world objects and how they are connected. You have already learned ER diagrams—this is your chance to apply that knowledge in a practical project.

- **Identify core entities** (e.g., Student, Project, Teacher, Payroll).
- **Define relationships** between entities (e.g., many-to-many between students and projects).
- **List key attributes** for each entity, clearly indicate:
    - **Primary keys** for each entity
    - **Foreign keys** for relationships

3.3 Logical Design

This phase focuses on transforming your ER diagram into precise **relational schemas**, choosing appropriate **data types**, and defining **constraints** that enforce data integrity.

- Translate each entity and relationship in your ER diagram into relational tables
    - Include entity tables (e.g., Student, Project)
    - Include relationship/association tables (e.g., Student_Project)
- For each table, define:
    - Attribute names and data types (e.g., VARCHAR, INT, DATE, DECIMAL)
    - Primary keys and Foreign keys
    - Constraints (e.g., NOT NULL, CHECK, UNIQUE)

4.4 Implementation

In this phase, you will implement your system in the form of an application that connects to a fully functional database. This includes both back-end logic and a basic user interface.

1. Database Setup

Create database and tables based on your design; Insert data.

2. Application Development

Develop a simple application (web-based) that connects to your database and simulates real interaction of different users.

Tool Recommendations

Back-End: Python Friendly (Recommended for Beginners)

Framework	Description
Flask	Lightweight, easy to learn, ideal for small apps
Django	Powerful and full-featured, but heavier for beginners

Front-End (The templates are allowed and can be customized for your project):

Framework	Description
HTML + JS	Minimal UI with forms/buttons
Bootstrap	For quick responsive styling
React/Vue	Optional (for experienced teams)

3.5 Performance Testing

Even if your project is small, it's important to consider performance, especially as data size grows. This helps you learn how databases scale and what strategies improve efficiency.

1. **Test Key Operations**

   - Measure **average time required** for core operations, such as Querying student wage history, Viewing all students in a project, Updating a salary record, Joining multiple tables for summary statistics.

2. **Analyze and Report**

   - Present a **table or summary** of test results: Operation name, test dataset size, average runtime
   - Discuss which operations are **slow or inefficient**, and hypothesize why

3. **Attempt Optimization *(Optional)***
   - ***a)*** Add or revise indexes on frequently queried fields
   - ***b)*** Try rewriting complex queries
   - ***c)*** Compare before and after runtime for at least one optimization attempt

Bonus: Analyzing the root cause (e.g., missing index, inefficient join) of the bottleneck and applying and testing at least one optimization method.

4 Submission

Each student must submit a compressed project package that includes *code, documentation, and a short video demonstration*.

Submission Checklist

1. Source Code
2. Demonstration Video: A screen recording (5–10 mins) showing the system. The video must include supporting operations from different roles.
3. Project Report (Word or PDF): it should include all sections in Detailed Project Requirement (Section 3), Summary & Reflections and Team Roles & Contribution.
4. A separate document titled `Team_Evaluation.docx` that records the scores you assigned to your teamate.

Reports must be well formatted, properly sectioned, and written in clear technical language.

5 Scoring Rubrics (Total: 12 Points)

1. Project Implementation (5 Points, assessed by TAs)

Criteria	Excellent (4-5)	Good (3)	Fair (2)	Poor (0-1)
Functionality	Fully functional system covering all core features with clear role-based behavior; excellent integration of database and application.	Mostly functional; minor missing features or bugs; core flows work well.	System works but lacks key features or has several bugs.	Non-functional or major features missing; unable to demonstrate main requirements.
Code & System Quality	Clean, modular, and well-documented code; clear structure in	Code mostly clean with minor issues; structure is	Code is disorganized or lacks	Code is messy, poorly documented, or

	both front-end and back-end; effective database connectivity.	understandable.	documentation; hard to follow.	non-functional.
Database Design & Query Logic	Schema is well-designed and normalized; uses views/triggers appropriately; SQL logic is correct and efficient.	Schema meets requirements; basic SQL logic works; minor inefficiencies.	Database structure has issues (e.g., redundancy); weak SQL implementation.	Poor or incorrect database structure; queries mostly non-functional.
Demonstration & Presentation	Clear, well-edited video; demonstrates system from multiple roles; good explanation of features.	Video covers most features; may lack polish or clarity.	Video is unclear, lacks role coverage or skips major features.	Missing, incomplete, or unclear demo video.

2. Project Report (5 Points, assessed by TAs)

Criteria	Excellent (4-5)	Good (3)	Fair (2)	Poor (0-1)
Understanding & Insight	Demonstrates solid understanding of system analysis and database design principles; insightful analysis and clear reasoning in decisions.	Shows good understanding of key concepts with mostly accurate explanations.	Demonstrates basic understanding; some sections are vague or unclear.	Lacks understanding of core concepts; significant errors or confusion.
Design Documentation	Includes all required sections: role analysis, ER diagram, relational schema, etc.; diagrams are clear and relevant.	Includes most key sections and diagrams; some minor issues or omissions.	Missing several components or diagrams are unclear.	Many required sections missing or poorly written; diagrams incomplete or absent.
Testing & Performance Analysis	Includes meaningful test data, performance results, and (if applicable) optimization insights.	Basic test results shown with some explanation.	Minimal or unclear performance testing.	No performance evaluation or irrelevant discussion.
Structure &	Well-organized,	Generally	Poorly organized;	Disorganized and hard

Clarity	easy to follow; uses appropriate formatting, headings, and technical language.	organized; minor formatting or clarity issues.	difficult to follow.	to understand; lacks formatting or structure.

3. Peer Evaluation (2 Points, assessed by group members)

Evaluation Methodology

Each group member scores every other member on a scale of 0 to 2:

0 points: Did not contribute or was unhelpful.

1 point: Contributed somewhat but did not meet expectations.

2 points: Contributed significantly and was very helpful.

Each member will not score themselves.

Individual Score Calculation

The individual score for each group member will be calculated as the average of the scores given to them by their peers.

Criteria	Excellent (2)	Fair (1)	Poor (0)
Individual Contribution	Received an average score of 1.5 to 2 from peers, indicating strong contribution and collaboration.	Received an average score of 0.5 to 1.4 from peers, indicating some contribution but with noticeable gaps.	Received an average score below 0.5 from peers, indicating minimal or no contribution.

4. Bonus Clarification

You may earn up to **2 bonus points** in the **final project** based on outstanding performance and exceptional work, but the total score for the lab and the final project combined **will not exceed 20** points.

For example, if your base score is 19 and you earn 2 bonus points, your final score is still **capped at 20**.