Add icon Add cover

Hospital/Healthcare Management System

Database Fundamentals

Complexity: Advanced | **Time Estimate:** 10-12 hours

Project Objectives

By the end of this project, learners will be able to:

- Design and normalize a relational database schema that models a hospital management domain effectively.
- Develop conceptual, logical, and physical database models to ensure scalability and maintainability.
- Implement CRUD operations and complex queries using SQL and JDBC.
- Apply indexing, hashing, searching, and sorting algorithms to optimize database access.
- Integrate database operations into a JavaFX application interface for practical interaction.
- Compare relational and NoSQL designs for unstructured data storage such as patient notes or medical logs.
- Measure and document performance improvement through optimization and indexing.

Project Description

This stage of the Hospital Management System focuses on building the data layer and persistence logic that will power the full application in later modules. Students will design a relational database for a hospital management system, implement the schema using SQL, and integrate it with a JavaFX application that performs essential operations — including patient registration, appointment scheduling, medical records management, and reporting. The system must reflect real-world hospital logic (patients, doctors, departments, appointments, prescriptions, medical inventory) while incorporating data structure and algorithm concepts (caching, indexing, sorting, searching) to enhance performance.

Epics and User Stories

Epic 1: Database Design and Modeling

User Story 1.1: As a database designer, I want to create conceptual, logical, and physical database models so that the data structure is clear and efficient.

Acceptance Criteria:

- Conceptual ERD includes all major entities and relationships.
- Logical model defines attributes, primary keys, and foreign keys.
- Physical model includes SQL data types, constraints, and normalization (up to 3NF).

User Story 1.2: As a database administrator, I want to define indexes and relationships so that queries run efficiently, and data integrity is maintained.

Acceptance Criteria:

- Appropriate primary and foreign keys are applied.
- Indexes are defined on frequently searched or joined columns.
- Referential integrity and constraints are enforced.

Epic 2: Data Access and CRUD Operations

User Story 2.1: As an administrator, I want to add, update, and delete patient and doctor data from the JavaFX interface so that I can manage hospital records.

Acceptance Criteria:

- All CRUD operations are functional via the JavaFX interface.

- Input validation and feedback messages are implemented.
- Database constraints prevent duplicate or invalid entries.

User Story 2.2: As a receptionist, I want to view patients and their medical history from the application so that I can assist with registration and appointment scheduling.

Acceptance Criteria:

- Patient listings are displayed dynamically from the database.
- Pagination or search filters improve usability.
- Data is retrieved using parameterized queries through JDBC.

Epic 3: Searching, Sorting, and Optimization

User Story 3.1: As a receptionist, I want to search for patients quickly by name or ID so that I can locate records efficiently.

Acceptance Criteria:

- Patient search is case-insensitive and responsive.
- Search optimization is achieved through indexing and/or hashing.
- Measurable improvement in query response time is documented.

User Story 3.2: As a developer, I want to implement in-memory caching and sorting so that frequently accessed or sorted data is retrieved faster.

Acceptance Criteria:

- Caching layer implemented using in-memory structures (e.g., maps or lists).
- Sorting and searching algorithms integrated and documented.
- Cache invalidation logic ensures data consistency after updates.

Epic 4: Performance and Query Optimization

User Story 4.1: As an analyst, I want to generate performance reports comparing pre- and post-optimization so that I can measure system efficiency.

Acceptance Criteria:

- Query execution times recorded before and after optimization.
- Indexes and caching demonstrate measurable performance gains.
- Report clearly communicates methodology and findings.

User Story 4.2: As a developer, I want to explore storing patient notes or medical logs in a NoSQL format so that unstructured data can be efficiently handled.

Acceptance Criteria:

- NoSQL data model created for patient notes or medical logs.
- Justification provided for why NoSQL is suitable for the use case.
- Optional implementation comparison included in performance report.

Epic 5: Reporting and Documentation

User Story 5.1: As a project contributor, I want to produce documentation and reports so that the project can be reviewed, reused, or extended.

Acceptance Criteria:

- ERD diagrams and database documentation included.
- SQL scripts and sample data provided.
- A README file describes setup, dependencies, and execution steps.

Technical Requirements

Database

- Use MySQL or PostgreSQL for the relational database implementation.
- Database schema normalized to Third Normal Form (3NF).
- Include at least the following entity groups: Patients, Doctors, Departments, Appointments, Prescriptions, PrescriptionItems, PatientFeedback, MedicalInventory.
- Define indexes on high-frequency columns (e.g., patient name, doctor ID, appointment date, department ID).
- Implement foreign keys to maintain referential integrity.

Application Layer (JavaFX + JDBC)

- Use JavaFX for user interaction: patient registration, appointment scheduling, CRUD operations, and reporting.
- Database access implemented using JDBC with parameterized queries to prevent SQL injection.
- Logical separation of concerns (Controller → Service → DAO).
- In-memory structures (e.g., Map, List) used to cache and sort results.
- Include an admin panel for patient/doctor management and performance monitoring.

Data Structures & Algorithms Integration

- **Hashing / Caching:** Use in-memory data structures to speed up lookups.
- **Sorting & Searching:** Apply appropriate algorithms for ordering and filtering data.
- **Indexing Concept:** Explain how in-memory structures mirror database index logic.
- **Performance Measurement:** Demonstrate efficiency improvements from optimization efforts.

Deliverables

	DELIVERABLE	DESCRIPTION
1	Database Design Document	Conceptual, logical, and physical models with ERDs and explanations.
2	SQL Implementation Script	Scripts for creating tables, constraints, indexes, and inserting sample data.
3	JavaFX Application	Functional interface that performs CRUD, search, and reporting using JDBC.
4	Performance Report	Comparative analysis showing before/after optimization metrics (indexing, caching).
5	NoSQL Design (Optional)	Documented schema or implementation for storing unstructured data (e.g., patient notes).
6	README File	Setup guide, dependencies, how to run SQL scripts, and usage instructions.
7	Testing Evidence	Screenshots or reports showing correct query results and validation outcomes.

Evaluation Criteria

	CATEGORY	EVALUATION CRITERIA	POINTS
1	Database Design (25 pts)	Conceptual, logical, and physical models are complete, normalized (3NF), and well-documented with ERDs.	25
2	SQL Implementation (20 pts)	SQL schema is syntactically correct, includes constraints, sample data, indexes, and supports complex queries.	20
3	JavaFX + JDBC Integration (20 pts)	Functional CRUD operations, UI usability, safe JDBC handling, and error feedback are correctly implemented.	20
4	DSA Application (15 pts)	Clear use of caching, sorting, and searching with performance justification and accurate mapping to indexing concepts.	15
5	Performance Optimization (10 pts)	Performance gains demonstrated through timing, analysis, and documentation.	10
6	Documentation & Code Quality (10 pts)	README completeness, code organization, comments, and adherence to clean coding practices.	10
7	**Total**		**100 pts**