AltSchool of Data Engineering Tinyuka 2024 Second Semester Project Exam

E-Learning Platform Database

This project serves to consolidate and demonstrate your comprehensive understanding of MongoDB database concepts, operations, and best practices. Through building a real-world application database, you will showcase your ability to design, implement, and optimize a NoSQL database solution that addresses practical business requirements.

The project simulates the data management challenges faced by modern web applications, requiring you to think critically about data modeling, performance optimization, and scalability considerations that are essential in today's technology landscape.

Project Overview

Create a complete database system for an online e-learning platform using MongoDB. This project will demonstrate your understanding of MongoDB fundamentals, document design, queries, aggregation, and database administration.

Project Requirements

Technical Requirements

- **MongoDB Version**: v8.0 or higher
- **Environment**: MongoDB Compass (GUI) and MongoDB Shell (CLI)
- **Programming Language**: Python with PyMongo library for MongoDB operations
- **Required Libraries**: pymongo, pandas (for data manipulation), datetime (for date operations)
- **Documentation**: Markdown format for all documentation
- **Data Format**: JSON for data exports and imports
- **Query Execution**: All queries and operations must be demonstrated in a Jupyter Notebook that shows both the executed Python code and their corresponding results/outputs

Functional Requirements

Your database system must support the following business operations:

1. **User Management**: Registration, authentication, profile management for students and instructors
2. **Course Management**: Course creation, publishing, categorization, and content organization
3. **Enrollment System**: Student course enrollment, progress tracking, completion status
4. **Assessment System**: Assignment creation, submission handling, grading, and feedback
5. **Analytics and Reporting**: Performance metrics, enrollment statistics, revenue tracking
6. **Search and Discovery**: Course search functionality with filtering and sorting capabilities

Performance Requirements

- Database queries must execute efficiently with appropriate indexing
- Aggregation pipelines should handle analytical queries for reporting
- Data retrieval operations should be optimized for common use cases
- The system should support concurrent read/write operations

Data Integrity Requirements

- Implement proper data validation rules and constraints

- Maintain referential integrity between related collections
- Handle duplicate data prevention and error scenarios
- Ensure data consistency across all operations

Documentation Requirements

- Comprehensive README with setup and usage instructions
- Inline code comments explaining complex queries and logic
- Schema documentation with field descriptions and relationships
- Performance analysis documentation with before/after comparisons

Learning Objectives

By completing this project, students will demonstrate proficiency in:

- Database and collection creation
- Document insertion and data modeling
- CRUD operations (Create, Read, Update, Delete)
- Query optimization and indexing
- Aggregation framework
- Data validation and schema design
- Error handling and data integrity

Project Scenario

You are tasked with building the database backend for “EduHub” – an online learning platform where:

- Students can enroll in courses
- Instructors can create and manage courses
- Courses contain multiple lessons and assignments
- Students can submit assignments and receive grades
- The system tracks student progress and completion rates

Database Structure

Collections to Create:

1. **users** – Students and instructors
2. **courses** – Course information
3. **enrollments** – Student course enrollments
4. **lessons** – Individual lessons within courses
5. **assignments** – Course assignments
6. **submissions** – Student assignment submissions

Part 1: Database Setup and Data Modeling (20 points)

Task 1.1: Create Database and Collections

```
# Connect to MongoDB and create database

from pymongo import MongoClient

from datetime import datetime

import pandas as pd

# Establish connection
```

```
client = MongoClient('mongodb://localhost:27017/')

db = client['eduhub_db']

# Create collections with validation rules
```

Task 1.2: Design Document Schemas

Create sample documents for each collection with appropriate data types and structure:

Users Collection Schema Example:

```
# Sample user document structure

user_schema = {

    "_id": "ObjectId (auto-generated)",

    "userId": "string (unique)",

    "email": "string (unique, required)",

    "firstName": "string (required)",

    "lastName": "string (required)",

    "role": "string (enum: ['student', 'instructor'])",

    "dateJoined": "datetime",

    "profile": {

        "bio": "string",

        "avatar": "string",

        "skills": ["string"]

    },

    "isActive": "boolean"

}
```

Courses Collection Schema Example:

```
# Sample course document structure

course_schema = {

    “_id”: “ObjectId (auto-generated)”,

    “courseId”: “string (unique)”,

    “title”: “string (required)”,

    “description”: “string”,

    “instructorId”: “string (reference to users)”,

    “category”: “string”,

    “level”: “string (enum: [‘beginner’, ‘intermediate’, ‘advanced’])”,

    “duration”: “number (in hours)”,

    “price”: “number”,

    “tags”: [“string”],

    “createdAt”: “datetime”,

    “updatedAt”: “datetime”,

    “isPublished”: “boolean”

}
```

Part 2: Data Population (15 points)

Task 2.1: Insert Sample Data

Insert at least:

- 20 users (mix of students and instructors)
- 8 courses across different categories
- 15 enrollments
- 25 lessons
- 10 assignments
- 12 assignment submissions

Task 2.2: Data Relationships

Ensure proper referential relationships between collections using appropriate field references.

Part 3: Basic CRUD Operations (25 points)

Task 3.1: Create Operations

Write Python code using PyMongo to:

1. Add a new student user
2. Create a new course
3. Enroll a student in a course
4. Add a new lesson to an existing course

Task 3.2: Read Operations

Write Python queries to:

1. Find all active students
2. Retrieve course details with instructor information
3. Get all courses in a specific category
4. Find students enrolled in a particular course
5. Search courses by title (case-insensitive, partial match)

Task 3.3: Update Operations

Write Python code to:

1. Update a user’s profile information
2. Mark a course as published
3. Update assignment grades
4. Add tags to an existing course

Task 3.4: Delete Operations

Write Python code to:

1. Remove a user (soft delete by setting isActive to false)
2. Delete an enrollment
3. Remove a lesson from a course

Part 4: Advanced Queries and Aggregation (25 points)

Task 4.1: Complex Queries

Write Python code using various PyMongo operators:

1. Find courses with price between $50 and $200
2. Get users who joined in the last 6 months
3. Find courses that have specific tags using $in operator
4. Retrieve assignments with due dates in the next week

Task 4.2: Aggregation Pipeline

Create aggregation pipelines using PyMongo for:

1. **Course Enrollment Statistics:**
   - Count total enrollments per course
   - Calculate average course rating
   - Group by course category
2. **Student Performance Analysis:**
   - Average grade per student
   - Completion rate by course

    - Top-performing students
3. **Instructor Analytics:**
    - Total students taught by each instructor
    - Average course rating per instructor
    - Revenue generated per instructor
4. **Advanced Analytics:**
    - Monthly enrollment trends
    - Most popular course categories
    - Student engagement metrics

Part 5: Indexing and Performance (10 points)

Task 5.1: Index Creation

Create appropriate indexes for:

1. User email lookup
2. Course search by title and category
3. Assignment queries by due date
4. Enrollment queries by student and course

Task 5.2: Query Optimization

1. Analyze query performance using explain() method in PyMongo
2. Optimize at least 3 slow queries
3. Document the performance improvements using Python timing functions

Part 6: Data Validation and Error Handling (5 points)

Task 6.1: Schema Validation

Implement validation rules for:

1. Required fields
2. Data type validation
3. Enum value restrictions
4. Email format validation

Task 6.2: Error Handling

Write queries that handle common errors:

1. Duplicate key errors
2. Invalid data type insertions
3. Missing required fields

Deliverables

1. Interactive Notebook (eduhub_mongodb_project.ipynb)

A Jupyter Notebook or similar interactive notebook containing:

- All MongoDB queries and operations with proper markdown explanations
- Executed code cells showing both the commands and their results
- Visual output of query results, document structures, and data samples
- Step-by-step execution flow with clear section headers

2. Code File (eduhub_queries.py)

A Python file containing all your MongoDB operations using PyMongo, properly commented and organized by task (backup/reference file).

3. Documentation (README.md)
Include:

- Project setup instructions
- Database schema documentation
- Query explanations
- Performance analysis results
- Challenges faced and solutions

4. Sample Data (sample_data.json)
Export your sample data for each collection.

5. Test Results (test_results.md)
Document the output of key queries and aggregations (if not using notebook format).

Evaluation Criteria

Component	**Points**	**Criteria**
Database Design	20	Proper schema design, relationships, data modeling
Data Population	15	Comprehensive sample data with realistic relationships
CRUD Operations	25	Correct implementation of all basic operations
Advanced Queries	25	Complex queries and aggregation pipelines
Performance	10	Proper indexing and query optimization
Documentation	5	Clear documentation and code comments

Bonus Challenges (Extra Credit)

1. Implement text search functionality for course content
2. Create a recommendation system using aggregation
3. Design a data archiving strategy for old enrollments
4. Implement geospatial queries for location-based course recommendations

Submission Guidelines

Primary Submissions:
1. **GitHub Repository**: Create a public GitHub repository containing all project files
    - Repository name format: mongodb-eduhub-project
    - Include a comprehensive README.md as the repository homepage
    - Organize files in appropriate folders (notebooks/, data/, docs/, etc.)
    - Commit history should show progressive development

    - Include a .gitignore file to exclude unnecessary files
2. **ZIP Archive**: Submit all files in a single ZIP archive as backup
    - Name your submission: LastName_FirstName_MongoDB_Project.zip
    - Should contain identical content to GitHub repository

Required Repository Structure:

```
mongodb-eduhub-project/

├── README.md

├── notebooks/

│   └── eduhub_mongodb_project.ipynb

├── src/

│   └── eduhub_queries.py

├── data/

│   ├── sample_data.json

│   └── schema_validation.json

├── docs/

│   ├── performance_analysis.md

│   └── presentation.pptx

└── .gitignore
```

GitHub Repository Requirements:

- **Repository Description**: Brief project description in repository settings
- **README.md**: Comprehensive project documentation with setup instructions
- **License**: Include an appropriate open-source license (MIT recommended)
- **Commits**: Regular commits showing project development progress
- **Branch Protection**: Work on main branch or use feature branches appropriately
- **Repository URL**: Submit the GitHub repository URL along with your ZIP file

Notebook Requirements:

- **Primary submission**: Interactive Jupyter notebook (.ipynb) showing all Python code and results
- Execute all Python code cells to show results
- Include comments within code cells explaining PyMongo operations
- Display sample outputs for data insertion and query results
- Show aggregation pipeline results with proper formatting
- Use pandas DataFrame display for better result visualization where appropriate

Documentation Requirements:

- Include a brief presentation (5-10 slides) explaining your design decisions
- Ensure notebook cells are executed and display results before submission
- Document any setup requirements or dependencies in README.md
- Include performance analysis results and optimization explanations

Submission Checklist:

- [ ] Public GitHub repository created and populated
- [ ] All code files committed and pushed to repository
- [ ] README.md includes setup instructions and project overview
- [ ] Jupyter notebook executed with visible results
- [ ] Sample data files included and properly formatted
- [ ] Presentation slides included in docs folder
- [ ] Repository URL submitted along with ZIP backup
- [ ] All deliverables organized in appropriate folders

Submission Due date: [Sunday, June 15, 2025, at 11:59 p.m. WAT]

Submission Format: GitHub repository URL + ZIP file backup