Course Project

Sunday, May 26, 2024

Table of contents

1 Quackstagram - A Social Media Platform with SQL Backend

Cheapo Software Solutions (CSS Inc) is looking to enhance their Quackstagram platform by adding a database for better data management and to provide analytics (to sell user data and monetize the platform). The current system, which stores data in text files, limits the ability to perform comprehensive analytics.

For this project, your task is to design a relational database for Quackstagram. You can either use the basic version of the platform we provided or the improved version from your Object-Oriented Modeling (OOM) project. After designing the database schema, you will implement it in a MySQL database. Additionally, you will need to create queries that can answer a list of questions from Cheapo Software Solution (complete list in Part D), aimed at maximizing the platform's profit potential (Money printer go BRRRR).

Help Cheapo Software Solutions make a quacking fortune!

Objective: Develop and implement a relational database for the "Quackstagram" social media platform, and integrate this database with your Java code.

2 Part A – Design a Relational Database Schema

2.1 Objective:

Design a relational database schema based on Quackstagram's features and functionalities as inferred from the Java source code [1] .

2.2 Tasks:

1. **Detailed Entity Analysis:**
    - Identify Entities: Perform a thorough analysis of the Java code to identify all entities such as Users, Posts, Comments, Likes, and Follows [2].
    - Identify Attributes: Define attributes for each entity based on variables and methods in the Java classes.
2. **Relationship Mapping:**
    - Determine the types of relationships (one-to-one, one-to-many, many-to-many) between entities.

[1] Do not submit rar! Only Zip are accepted.

[2] You may also wish to review all of the questions that the company wants you to answer to get a better sense of what is needed from your DB.

- Identify if many-to-many relationships exist. They may have an impact on your schema.

3. **Normalization and Functional Dependencies:**
   - [Do this after the lecture on functional dependencies]
   - Document functional dependencies for at least 4 tables (you can pick any 4 tables).
   - Provide a step-by-step breakdown of the normalization process up to 3NF or BCNF [3] (for all 4 of the picked tables).
4. **ERD:** Create a detailed ERD with clear notation, showcasing all entities, relationships, and key constraints.

3 Part B – Implement a MySQL-Compatible Relational Database Schema

3.1 Objective:

Develop a MySQL database schema based on the design, and prepare it for integration with the Java application.

3.2 Tasks:

1. **Schema.sql:**
   - Write precise `CREATE TABLE` statements for the planned schema.
   - Include any relavent column constraints and types for data integrity and accuracy. Include definitions of primary keys, any foreign keys and any unique attributes; specify any default values of attributes;
   - Populate the database with substantial `INSERT INTO` statements for realistic testing [4] .
2. **Views.sql:**
   - Create at least 3 meaningful views that provide insights into user behavior, content popularity, and system analytics (1 for each of these categroies). Each query must contain either a subquery or both GROUP BY and HAVING clauses;

[3] Note that if your tables are in Boyce-Codd Normal Form (BCNF) they are also in 3NF.

[4] I would not mind if you used a LLM to come up with dummy data to dump into your DB. You can also intert randomly generated values here, the exact values are not as relavent for us as they would evenetually be replaced by real data obtained from real users.

- Design at least 2 indexes to optimize the performance of these views and overall query speed. You need to document the performance of queries before and after the index application to demonsrate the improvement and justify the application.

3. **Triggers.sql:**
   - Create file triggers.sql which contains 1 procedure, 1 function and 2 triggers. At least one of the triggers must call your procedure and your function. (e.g., post creation, user follow, you can pick any operation, these are just examples).
   - Develop at least one stored procedure and function for frequent or complex database operations.

4 Part C – Integration and Functional Application Development

4.1 Objective:

Integrate the designed database with the Java application to create a fully functional Quackstagram platform.

4.2 Tasks:

1. **Database Connection:**
   - Set up the JDBC connection [5] between the Java application and the MySQL database.
   - Provide clear code for database connection, querying [6], and data manipulation.
2. **Feature Implementation:**
   - Handle all the data processes with the database now (you cannot use text files anymore). You will need to utilize the database for key features like user registration, post creation, and following other users.
3. **Error Handling and Security:**
   - Implement error handling for database interactions.

[5] You can use any other connector you want as long as it is MySQL or SQLite database on the other end.

[6] You do not have to integrate answers to all the questions by Cheapo Technologies into your Java source code, these answers can be supplied in a seperate file. Here the goal is to replace all the text files with a SQL database.

5 Part D – Profit Maximization SQL queries

5.1 Objective:

Data is the new gold! We need to extract as much data as possible to montise it. Write SQL queries that can answer the following questions for Cheapo Technologies.

- Questions:

1. List all users who have more than X followers where X can be any integer value.
2. Show the total number of posts made by each user. (You will have to decide how this is done, via a username or userid)
3. Find all comments made on a particular user's post.
4. Display the top X most liked posts.
5. Count the number of posts each user has liked.
6. List all users who haven't made a post yet.
7. List users who follow each other.
8. Show the user with the highest number of posts.
9. List the top X users with the most followers.
10. Find posts that have been liked by all users.
11. Display the most active user (based on posts, comments [7], and likes).
12. Find the average number of likes per post for each user.
13. Show posts that have more comments than likes.
14. List the users who have liked every post of a specific user.
15. Display the most popular post of each user (based on likes).
16. Find the user(s) with the highest ratio of followers to following.
17. Show the month with the highest number of posts made.
18. Identify users who have not interacted with a specific user's posts.
19. Display the user with the greatest increase in followers in the last X days.
20. Find users who are followed by more than X% of the platform users.

6 Part E – Report

We need a very simple report with a few things in it:

6.1 Contents:

1. The names and IDs of the students in your project group and which parts of the project each student has worked on.

[7] Only if you have implemented comments in your code, if not, just consider likes.

2. A couple of paragraphs explaining what your database is about and a description of how the Quackstagram project uses your database.
3. Entity-relationship diagram for your database.
4. An example of each table with some data and primary key attributes clearly identified.
5. The list of FDs for each table.
6. Proof that each table is in 3NF.
7. Justification for the usefulness of the views proposed in part B within a scenario for possible use of the views in the Quackstagram Project.
8. Analysis of the speed of the queries in your views and justification for the indexes proposed in part B.
9. Justification for the necessity of the triggers and stored procedure and function proposed in part B within a scenario for possible use of the triggers within Quackstagram.
10. SQL queries with answers to all the questions asked in Part D.

7 Submission Guidelines

- **Deadline:** End of Week 6.
- **Format:** Submit a .zip [8] file containing the detailed report (Part E) and all code files (Parts B, C and D) on Canvas.

8 Marking Scheme (Total 25 Points)

1. Schema Design (2 points)

- We will examine the thoughtfulness, correctness, and efficiency of the database schema.
- For this, we will look at your report, ERD diagram, and SQL file for creating the tables.

2. Normalization (4 points)

- Each table normalization is worth 1 point.
- We will examine the accuracy of the normalization process.
- This will be examined by looking at your report and normalized table structure.

3. Application Integration and Functionality (6 points)

[8] Do not submit rar! Only Zip are accepted.

- Assess the effectiveness and smoothness of integrating the database with the Java application.
- You will get 6 points if:
  - The code is well-integrated structurally
  - There is appropriate error handling
  - The written queries are efficient
- You will get 4 points if the code is integrated, but the structure of integration is not efficient (you break any of the OOP principles while implementing the DB integration).
- You will get < 4 points if the code is not well integrated or there is clear issues with the integration (components are broken or it does not work).
  - You will lose 2 points if the error handling is not appropriately managed.

4. Views, Triggers, and Stored Procedures (5 points)

- **Views (1.5 points):**
  - Each view is worth 0.5 points (1.5 points in total)
  - To get full points, your views need to be based on the 3 categories mentioned in the document and should work as intended.
- **Indexes (1 point):**
  - Each index is worth 0.5 points (1 point in total)
  - The index should be applied in an appropriate context to count
  - You can determine the appropriateness of your index application based on the performance before and after the application of the index (this needs to be documented in your report).
- **Triggers (1.5 point):**
  - Each trigger is worth 0.75 points (1.5 point in total)
  - A well-written trigger adhering to the requirements above would yield full points; otherwise, you lose partial points.
- **Stored Procedures and Functions (1 point):**
  - For each procedure and function, you get 0.5 points (1 point in total)
  - The grading is based on the appropriateness of the procedure or function and its justification.

5. Cheapo Software Solutions Profit Maximization SQL queries (8 points)

- Each SQL query is worth 0.4 points.
- If your query is appropriate and not hardcoded (e.g., if the question asks for X, you can expect the user to provide that value), you get full points for that query; otherwise, you get 0 points.

8.1 Grading Rubric

Criterion	Grading Scheme	Points
Schema Design	Thoughtful, correct, and efficient schema design	2
	Incomplete or inefficient schema design	1
	No schema design provided	0
Normalization	All tables properly normalized (1 point per table)	4
	Some tables not properly normalized	1-3
	No normalization performed	0
Application Integration and Functionality	Well-integrated code with proper error handling and efficient queries	6
	Integrated code, but violates OOP principles or inefficient integration	4
	Code not well-integrated or clear integration issues (components broken or not working)	< 4
	Lack of proper error handling	-2
Views	All 3 required views implemented correctly (0.5 points each)	1.5
	Some views implemented correctly	0.5-1
	No views implemented	0
Indexes	Appropriate indexes implemented (0.5 points each)	1
	Some appropriate indexes implemented	0.5
	No indexes implemented	0
Triggers	Well-written triggers adhering to requirements (0.75 points each)	1.5
	Partially correct triggers	0.75-1.5
	No triggers implemented	0
Stored Procedures and Functions	Appropriate procedures and functions implemented (0.5 points each)	1
	Some appropriate procedures and functions implemented	0.5
	No procedures or functions implemented	0
Cheapo Technologies Profit Maximization SQL queries	Appropriate and non-hardcoded queries (0.4 points each)	8
	Hardcoded or incorrect queries	0