DataBaseManagementSystem

ASSIGNMENT – 1

Y. Marthanda Teja

192311285

1Q) Content Management System for Blogging Platform Design a database for a blogging platform that supports users, blog posts, comments, likes, and tags. Requirements: Create tables for users, posts, comments, tags, and likes, establishing relationships between them. Write stored procedures to handle creating, updating, and deleting blog posts and comments. Write SQL queries to analyze blog popularity by views, likes, and comments. Implement role-based access control for authors, moderators, and admins, ensuring secure content management.

**A)1. Database Schema Design**

**Users Table**

Stores user information and roles.

| **user\_id** | **username** | **email** | **password** | **role** | **created\_at** | **updated\_at** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | JohnDoe | john@example.com | hashed\_pass1 | author | 2024-01-01 10:00:00 | 2024-01-01 10:00:00 |
| 2 | JaneMod | jane@example.com | hashed\_pass2 | moderator | 2024-01-02 11:00:00 | 2024-01-02 11:00:00 |
| 3 | AdminMike | admin@example.com | hashed\_pass3 | admin | 2024-01-03 12:00:00 | 2024-01-03 12:00:00 |

**Posts Table**

Stores blog posts written by users.

| **post\_id** | **title** | **content** | **author\_id** | **views** | **created\_at** | **updated\_at** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | First Blog Post | Lorem Ipsum... | 1 | 150 | 2024-01-10 10:00:00 | 2024-01-10 10:00:00 |
| 2 | Second Blog Post | Hello World... | 1 | 120 | 2024-01-12 10:00:00 | 2024-01-12 10:00:00 |

**Comments Table**

Stores comments on blog posts.

| **comment\_i** | **post\_i** | **user\_i** |  |  | **updated\_at** |
| --- | --- | --- | --- | --- | --- |
| **d** | **d** |  |  |  |  |
| 1 | 1 | 2 | Great post! | 2024-01-11 11:00:00 | 2024-01-11 11:00:00 |
| 2 | 1 | 3 | Needs improvement. | 2024-01-12 12:00:00 | 2024-01-12 12:00:00 |

**Tags Table**

Stores tags for categorizing posts.

| **tag\_id** | **name** |
| --- | --- |
| 1 | Technology |
| 2 | Programming |

**PostTags Table**

Links posts with tags (many-to-many relationship).

| **post\_id** | **tag\_id** |
| --- | --- |
| 1 | 1 |
| 1 | 2 |
| 2 | 1 |

**Likes Table**

Stores likes given by users to posts.

| **like\_id** | **post\_id** | **user\_id** | **created\_at** |
| --- | --- | --- | --- |
| 1 | 1 | 2 | 2024-01-11 11:30:00 |
| 2 | 1 | 3 | 2024-01-12 12:30:00 |

**Stored Procedures**

1. **Creating a Blog Post**

sql

CREATE PROCEDURE create\_post(IN p\_user\_id INT, IN p\_title VARCHAR(255), IN p\_content TEXT)

BEGIN

INSERT INTO posts (user\_id, title, content) VALUES (p\_user\_id, p\_title, p\_content);

END;

1. **Updating a Blog Post**

sql

CREATE PROCEDURE update\_post(IN p\_post\_id INT, IN p\_title VARCHAR(255), IN p\_content TEXT)

BEGIN

UPDATE posts SET title = p\_title, content = p\_content, updated\_at = CURRENT\_TIMESTAMP WHERE post\_id = p\_post\_id;

END;

1. **Deleting a Blog Post**

sql

CREATE PROCEDURE delete\_post(IN p\_post\_id INT)

BEGIN

DELETE FROM posts WHERE post\_id = p\_post\_id;

END;

1. **Creating a Comment**

sql

CREATE PROCEDURE create\_comment(IN p\_post\_id INT, IN p\_user\_id INT, IN p\_content TEXT)

BEGIN

INSERT INTO comments (post\_id, user\_id, content) VALUES (p\_post\_id, p\_user\_id, p\_content);

END;

1. **Updating a Comment**

sql

CREATE PROCEDURE update\_comment(IN p\_comment\_id INT, IN p\_content TEXT)

BEGIN

UPDATE comments SET content = p\_content WHERE comment\_id = p\_comment\_id;

END;

1. **Deleting a Comment**

sql

CREATE PROCEDURE delete\_comment(IN p\_comment\_id INT)

BEGIN

DELETE FROM comments WHERE comment\_id = p\_comment\_id;

END;

**SQL Queries to Analyze Blog Popularity**

1. **Popularity by Views**

sql

SELECT post\_id, title, views

FROM posts

ORDER BY views DESC

LIMIT 10;

1. **Popularity by Likes**

sql

SELECT p.post\_id, p.title, COUNT(l.like\_id) AS like\_count

FROM posts p

LEFT JOIN likes l ON p.post\_id = l.post\_id

GROUP BY p.post\_id, p.title

ORDER BY like\_count DESC

LIMIT 10;

1. **Popularity by Comments**

sql

SELECT p.post\_id, p.title, COUNT(c.comment\_id) AS comment\_count

FROM posts p

LEFT JOIN comments c ON p.post\_id = c.post\_id

GROUP BY p.post\_id, p.title

ORDER BY comment\_count DESC

LIMIT 10;

**Role-Based Access Control**

To implement role-based access control, you can use stored procedures and SQL to enforce permissions based on the user's role.

1. **Example Stored Procedure for Creating Posts with Role Check**

sql

CREATE PROCEDURE create\_post\_with\_role\_check(IN p\_user\_id INT, IN p\_title VARCHAR(255), IN p\_content TEXT)

BEGIN

DECLARE v\_role ENUM('author', 'moderator', 'admin');

SELECT role INTO v\_role FROM users WHERE user\_id = p\_user\_id;

IF v\_role IN ('author', 'admin') THEN

INSERT INTO posts (user\_id, title, content) VALUES (p\_user\_id, p\_title, p\_content);

ELSE

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'User does not have permission to create posts.';

END IF;

END;

1. **Example Stored Procedure for Deleting Comments with Role Check**

sql

CREATE PROCEDURE delete\_comment\_with\_role\_check(IN p\_comment\_id INT, IN p\_user\_id INT)

BEGIN

DECLARE v\_role ENUM('author', 'moderator', 'admin');

DECLARE v\_comment\_user\_id INT;

SELECT role INTO v\_role FROM users WHERE user\_id = p\_user\_id;

SELECT user\_id INTO v\_comment\_user\_id FROM comments WHERE comment\_id = p\_comment\_id;

IF v\_role = 'admin' OR (v\_role = 'moderator' AND v\_comment\_user\_id = p\_user\_id) THEN

DELETE FROM comments WHERE comment\_id = p\_comment\_id;

ELSE

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'User does not have permission to delete this comment.';

END IF;

END;