Case Study Title: UrbanShadows User Login System

**SCHEMA DESIGN**

|  |
| --- |
| CREATE TABLE users (  user\_id INT PRIMARY KEY AUTO\_INCREMENT,  username VARCHAR(50) UNIQUE NOT NULL,  email VARCHAR(100) UNIQUE NOT NULL,  password VARCHAR(255) NOT NULL,  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  ); |

**Sample Data Insertion**

|  |
| --- |
| INSERT INTO users (username, email, password)  VALUES  ('ghostwalker', 'ghostwalker@urbanshadows.com', 'ghost123'),  ('phantomqueen', 'phantomqueen@urbanshadows.com', 'phantom321'),  ('shadowseeker', 'shadowseeker@urbanshadows.com', 'seek@dark'); |

**SQL SCENARIOS & QUERIES**

1. **Check if a User Exists (Login Validation)**

|  |
| --- |
| SELECT \* FROM users  WHERE username = 'ghostwalker' AND password = 'ghost123'; |

1. **Register New User (From New\_user.html)**

|  |
| --- |
| INSERT INTO users (username, email, password)  VALUES ('darkrider', 'darkrider@urbanshadows.com', 'r1d3Dark!'); |

1. **Forget Password – Update Password (From change\_password.html)**

|  |
| --- |
| UPDATE users  SET password = 'newSecurePass123'  WHERE email = 'phantomqueen@urbanshadows.com'; |

1. **List All Registered Users**

|  |
| --- |
| SELECT user\_id, username, email, created\_at  FROM users  ORDER BY created\_at DESC; |

1. **Delete an Inactive User (Admin-level operation)**

|  |
| --- |
| DELETE FROM users  WHERE username = 'darkrider'; |

1. **Search for Users with Specific Domain**

|  |
| --- |
| SELECT username  FROM users  WHERE email LIKE '%@urbanshadows.com'; |

1. **Count Total Users**

|  |
| --- |
| SELECT COUNT(\*) AS total\_users FROM users; |

1. **Stored Procedure for New User Registration**

**Use Case: Register a new user only if their email doesn’t already exist.**

|  |
| --- |
| DELIMITER //  CREATE PROCEDURE register\_user (  IN p\_username VARCHAR(50),  IN p\_email VARCHAR(100),  IN p\_password VARCHAR(255)  )  BEGIN  IF EXISTS (SELECT 1 FROM users WHERE email = p\_email) THEN  SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Email already registered.';  ELSE  INSERT INTO users (username, email, password)  VALUES (p\_username, p\_email, p\_password);  END IF;  END;  //  DELIMITER ; |
| CALL register\_user('haunter42', 'haunter42@urbanshadows.com', 'safeHaunt123'); |

1. **Login Function with Return Status**

**Use Case: Return login status as success/failure from a function.**

|  |
| --- |
| DELIMITER //  CREATE PROCEDURE login\_user (  IN p\_username VARCHAR(50),  IN p\_password VARCHAR(255),  OUT p\_status VARCHAR(20)  )  BEGIN  IF EXISTS (  SELECT 1 FROM users  WHERE username = p\_username AND password = p\_password  ) THEN  SET p\_status = 'Login Success';  ELSE  SET p\_status = 'Login Failed';  END IF;  END;  //  DELIMITER ; |

|  |
| --- |
| CALL login\_user('ghostwalker', 'ghost123', @login\_status);  SELECT @login\_status; |

**DATABASE NAME: UrbanShadowsDB**

1. **Tables Structure -** users

|  |
| --- |
| CREATE TABLE users (  user\_id INT PRIMARY KEY AUTO\_INCREMENT,  username VARCHAR(50) NOT NULL UNIQUE,  email VARCHAR(100),  password VARCHAR(255),  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  ); |

**Blogs**

|  |
| --- |
| CREATE TABLE blogs (  blog\_id INT PRIMARY KEY AUTO\_INCREMENT,  user\_id INT,  title VARCHAR(255),  content TEXT,  image\_url VARCHAR(255),  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  FOREIGN KEY (user\_id) REFERENCES users(user\_id)  ); |

**Likes**

|  |
| --- |
| **CREATE TABLE likes (**  **like\_id INT PRIMARY KEY AUTO\_INCREMENT,**  **blog\_id INT,**  **user\_id INT,**  **liked\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,**  **FOREIGN KEY (blog\_id) REFERENCES blogs(blog\_id),**  **FOREIGN KEY (user\_id) REFERENCES users(user\_id)**  **);** |

**Comments**

|  |
| --- |
| CREATE TABLE comments (  comment\_id INT PRIMARY KEY AUTO\_INCREMENT,  blog\_id INT,  user\_id INT,  comment TEXT,  commented\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  FOREIGN KEY (blog\_id) REFERENCES blogs(blog\_id),  FOREIGN KEY (user\_id) REFERENCES users(user\_id)  ); |

|  |
| --- |
| -- Insert Users  INSERT INTO users (username, email, password) VALUES  ('anjali\_sharma', 'anjali@example.com', 'pass123'),  ('priya\_sen', 'priya@example.com', 'pass456'),  ('ravi\_kumar', 'ravi@example.com', 'pass789'),  ('sneha\_malik', 'sneha@example.com', 'pass321');  -- Insert Blogs  INSERT INTO blogs (user\_id, title, content, image\_url) VALUES  (1, 'The Woman in the Mirror', 'Every night at 3 AM, a pale woman appears in the mirror...', 'mirror.jpg'),  (2, 'Footsteps on the Roof', 'On full moon nights, footsteps are heard on the roof but no one is ever found...', 'footsteps.jpg'),  (1, 'The Haunted Library', 'Books fly off the shelves and whispers fill the halls after midnight.', 'library.jpg');  -- Insert Likes  INSERT INTO likes (blog\_id, user\_id) VALUES  (1, 3), (1, 4), (1, 2), (1, 1), (2, 1), (2, 3), (3, 4);  -- Insert Comments  INSERT INTO comments (blog\_id, user\_id, comment) VALUES  (1, 3, 'Creepy! I’m never looking in a mirror at night again!'),  (1, 4, 'That gave me chills!'),  (2, 4, 'That story reminds me of my grandma’s tales 😱'),  (3, 2, 'I felt a cold breeze just reading this!'),  (3, 3, 'The library is the scariest place after hours.'); |

1. **Add a blog by user**

|  |
| --- |
| **INSERT INTO blogs (user\_id, title, content, image\_url)**  **VALUES (1, 'The Woman in the Mirror', 'Every night at 3 AM...', 'mirror.jpg');** |

1. **Retrieve all blogs with author**

|  |
| --- |
| **SELECT b.blog\_id, b.title, b.content, u.username AS author**  **FROM blogs b**  **JOIN users u ON b.user\_id = u.user\_id;** |

1. **Show all blog titles with like counts – JOINS**

|  |
| --- |
| **SELECT b.title, COUNT(l.like\_id) AS total\_likes**  **FROM blogs b**  **LEFT JOIN likes l ON b.blog\_id = l.blog\_id**  **GROUP BY b.blog\_id;** |

1. **Create a view to show blog with comment counts – VIEW**

|  |
| --- |
| **CREATE VIEW blog\_summary AS**  **SELECT b.blog\_id, b.title, u.username,**  **COUNT(c.comment\_id) AS total\_comments**  **FROM blogs b**  **JOIN users u ON b.user\_id = u.user\_id**  **LEFT JOIN comments c ON b.blog\_id = c.blog\_id**  **GROUP BY b.blog\_id, u.username;** |

1. **Get users who posted more than one blog – subquery**

|  |
| --- |
| **SELECT username**  **FROM users**  **WHERE user\_id IN (**  **SELECT user\_id**  **FROM blogs**  **GROUP BY user\_id**  **HAVING COUNT(blog\_id) > 1**  **);** |

1. **Show blog titles with popularity label based on likes – CASE**

|  |
| --- |
| **SELECT b.title,**  **COUNT(l.like\_id) AS like\_count,**  **CASE**  **WHEN COUNT(l.like\_id) >= 5 THEN '🔥 Popular'**  **WHEN COUNT(l.like\_id) BETWEEN 1 AND 4 THEN '👍 Average'**  **ELSE '🧊 Unnoticed'**  **END AS popularity\_status**  **FROM blogs b**  **LEFT JOIN likes l ON b.blog\_id = l.blog\_id**  **GROUP BY b.blog\_id;** |

1. **Create trigger to auto-insert "Welcome!" comment when new blog is posted – TRIGGER**

|  |
| --- |
| **CREATE TRIGGER welcome\_comment**  **AFTER INSERT ON blogs**  **FOR EACH ROW**  **BEGIN**  **INSERT INTO comments (blog\_id, user\_id, comment)**  **VALUES (NEW.blog\_id, NEW.user\_id, 'Welcome to UrbanShadows blog!');**  **END;** |

1. **Cursor to loop through blogs and count comments – CURSOR**

|  |
| --- |
| **DELIMITER //**  **CREATE PROCEDURE CountCommentsPerBlog()**  **BEGIN**  **DECLARE done INT DEFAULT FALSE;**  **DECLARE blogId INT;**  **DECLARE blogTitle VARCHAR(255);**  **DECLARE cur CURSOR FOR SELECT blog\_id, title FROM blogs;**  **DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;**  **OPEN cur;**  **read\_loop: LOOP**  **FETCH cur INTO blogId, blogTitle;**  **IF done THEN**  **LEAVE read\_loop;**  **END IF;**  **SELECT blogTitle AS 'Blog Title',**  **(SELECT COUNT(\*) FROM comments WHERE blog\_id = blogId) AS 'Total Comments';**  **END LOOP;**  **CLOSE cur;**  **END //**  **DELIMITER ;** |

1. **Procedure to add a new comment - Stored Procedure**

|  |
| --- |
| **DELIMITER //**  **CREATE PROCEDURE AddComment(**  **IN p\_blog\_id INT,**  **IN p\_user\_id INT,**  **IN p\_comment TEXT**  **)**  **BEGIN**  **INSERT INTO comments (blog\_id, user\_id, comment)**  **VALUES (p\_blog\_id, p\_user\_id, p\_comment);**  **END //**  **DELIMITER ;** |

**SQL Case Study for UrbanShadows Scary Stories**

**Tables**

|  |
| --- |
| **CREATE TABLE users (**  **user\_id INT PRIMARY KEY AUTO\_INCREMENT,**  **username VARCHAR(50),**  **email VARCHAR(100),**  **password VARCHAR(100)**  **);**  **CREATE TABLE stories (**  **story\_id INT PRIMARY KEY AUTO\_INCREMENT,**  **user\_id INT,**  **title VARCHAR(255),**  **content TEXT,**  **image\_url VARCHAR(255),**  **created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,**  **FOREIGN KEY (user\_id) REFERENCES users(user\_id)**  **);**  **CREATE TABLE comments (**  **comment\_id INT PRIMARY KEY AUTO\_INCREMENT,**  **story\_id INT,**  **user\_id INT,**  **comment TEXT,**  **commented\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,**  **FOREIGN KEY (story\_id) REFERENCES stories(story\_id),**  **FOREIGN KEY (user\_id) REFERENCES users(user\_id)**  **);**  **CREATE TABLE likes (**  **like\_id INT PRIMARY KEY AUTO\_INCREMENT,**  **story\_id INT,**  **user\_id INT,**  **liked\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,**  **FOREIGN KEY (story\_id) REFERENCES stories(story\_id),**  **FOREIGN KEY (user\_id) REFERENCES users(user\_id)**  **);** |

**Insert Dummy Data**

|  |
| --- |
| **-- Users**  **INSERT INTO users (username, email, password) VALUES**  **('anjali\_sharma', 'anjali@example.com', 'pass123'),**  **('priya\_sen', 'priya@example.com', 'pass456'),**  **('ravi\_kumar', 'ravi@example.com', 'pass789'),**  **('sneha\_malik', 'sneha@example.com', 'pass321');**  **-- Stories**  **INSERT INTO stories (user\_id, title, content, image\_url) VALUES**  **(1, 'The Umbrella Man', 'A rainy walk home turns chilling when Arpit meets a man no one else can see...', 'umbrella.jpg'),**  **(2, 'Not an Earthquake', 'A solo night in a deserted Dharchula guest house turns eerie when Rohit experiences mysterious bed shaking...', 'earthquake.jpg'),**  **(3, 'The Sea Has Hands', 'A chilling Christmas eve at Dumas Beach turns terrifying when a mysterious push pulls a photographer into the sea...', 'sea.jpg');**  **-- Comments**  **INSERT INTO comments (story\_id, user\_id, comment) VALUES**  **(1, 3, 'That sounds spooky!'),**  **(1, 4, 'I would never walk alone in rain anymore.'),**  **(2, 1, 'Could it be something supernatural?'),**  **(3, 2, 'Beach stories always give me the creeps.');**  **-- Likes**  **INSERT INTO likes (story\_id, user\_id) VALUES**  **(1, 2),**  **(1, 3),**  **(2, 1),**  **(3, 4),**  **(3, 1);** |

1. **Get all stories titles and their authors:**

|  |
| --- |
| **SELECT s.title, u.username**  **FROM stories s**  **JOIN users u ON s.user\_id = u.user\_id;** |

1. **Get all comments for story\_id = 1:**

|  |
| --- |
| **SELECT comment, commented\_at FROM comments WHERE story\_id = 1;** |

1. **List all stories with number of likes: - JOIN**

|  |
| --- |
| **SELECT s.title, COUNT(l.like\_id) AS total\_likes**  **FROM stories s**  **LEFT JOIN likes l ON s.story\_id = l.story\_id**  **GROUP BY s.story\_id;** |

1. **List comments with commenter usernames and story titles: - JOIN**

|  |
| --- |
| **SELECT c.comment, u.username, s.title**  **FROM comments c**  **JOIN users u ON c.user\_id = u.user\_id**  **JOIN stories s ON c.story\_id = s.story\_id;** |

1. **Trigger to automatically update story created\_at on update**

|  |
| --- |
| **DELIMITER //**  **CREATE TRIGGER update\_story\_timestamp BEFORE UPDATE ON stories**  **FOR EACH ROW**  **BEGIN**  **SET NEW.created\_at = NOW();**  **END //**  **DELIMITER ;** |

1. **Trigger to prevent duplicate likes by same user on a story**

|  |
| --- |
| **DELIMITER //**  **CREATE TRIGGER prevent\_duplicate\_likes BEFORE INSERT ON likes**  **FOR EACH ROW**  **BEGIN**  **IF EXISTS (**  **SELECT 1 FROM likes WHERE story\_id = NEW.story\_id AND user\_id = NEW.user\_id**  **) THEN**  **SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'User already liked this story';**  **END IF;**  **END //**  **DELIMITER ;** |

1. **View for stories with author names and like count:**

|  |
| --- |
| **CREATE VIEW story\_likes\_view AS**  **SELECT s.story\_id, s.title, u.username AS author, COUNT(l.like\_id) AS likes**  **FROM stories s**  **JOIN users u ON s.user\_id = u.user\_id**  **LEFT JOIN likes l ON s.story\_id = l.story\_id**  **GROUP BY s.story\_id;** |

1. **View for all comments with story title and commenter name:**

|  |
| --- |
| **CREATE VIEW comment\_details\_view AS**  **SELECT c.comment\_id, c.comment, s.title AS story\_title, u.username AS commenter**  **FROM comments c**  **JOIN stories s ON c.story\_id = s.story\_id**  **JOIN users u ON c.user\_id = u.user\_id;** |

1. **Show story popularity based on likes count:CASE**

|  |
| --- |
| **SELECT s.title,**  **COUNT(l.like\_id) AS total\_likes,**  **CASE**  **WHEN COUNT(l.like\_id) > 3 THEN 'Very Popular'**  **WHEN COUNT(l.like\_id) BETWEEN 1 AND 3 THEN 'Popular'**  **ELSE 'Less Popular'**  **END AS popularity**  **FROM stories s**  **LEFT JOIN likes l ON s.story\_id = l.story\_id**  **GROUP BY s.story\_id;** |

1. **Show user status based on number of stories posted:CASE**

|  |
| --- |
| **SELECT u.username,**  **COUNT(s.story\_id) AS stories\_posted,**  **CASE**  **WHEN COUNT(s.story\_id) = 0 THEN 'New User'**  **WHEN COUNT(s.story\_id) BETWEEN 1 AND 2 THEN 'Contributor'**  **ELSE 'Prolific Author'**  **END AS status**  **FROM users u**  **LEFT JOIN stories s ON u.user\_id = s.user\_id**  **GROUP BY u.user\_id;** |

1. **Cursor to print titles of all stories (example in MySQL stored procedure):CURSOR**

|  |
| --- |
| **DELIMITER //**  **CREATE PROCEDURE print\_all\_story\_titles()**  **BEGIN**  **DECLARE done INT DEFAULT FALSE;**  **DECLARE story\_title VARCHAR(255);**  **DECLARE cur CURSOR FOR SELECT title FROM stories;**  **DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;**  **OPEN cur;**  **read\_loop: LOOP**  **FETCH cur INTO story\_title;**  **IF done THEN**  **LEAVE read\_loop;**  **END IF;**  **SELECT story\_title; -- or do something with the title**  **END LOOP;**  **CLOSE cur;**  **END //**  **DELIMITER ;** |

1. **Cursor to iterate over stories and update a dummy field (example):CURSOR**

|  |
| --- |
| **-- Suppose we add a dummy column 'is\_featured' (BOOLEAN) to stories:**  **ALTER TABLE stories ADD COLUMN is\_featured BOOLEAN DEFAULT FALSE;**  **DELIMITER //**  **CREATE PROCEDURE feature\_popular\_stories()**  **BEGIN**  **DECLARE done INT DEFAULT FALSE;**  **DECLARE s\_id INT;**  **DECLARE cur CURSOR FOR**  **SELECT story\_id FROM stories WHERE story\_id IN (SELECT story\_id FROM likes GROUP BY story\_id HAVING COUNT(\*) > 1);**  **DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;**  **OPEN cur;**  **read\_loop: LOOP**  **FETCH cur INTO s\_id;**  **IF done THEN**  **LEAVE read\_loop;**  **END IF;**  **UPDATE stories SET is\_featured = TRUE WHERE story\_id = s\_id;**  **END LOOP;**  **CLOSE cur;**  **END //**  **DELIMITER ;** |

1. **Stories that have more likes than the average number of likes:SUBQUERY**

|  |
| --- |
| **SELECT title FROM stories WHERE story\_id IN (**  **SELECT story\_id FROM likes GROUP BY story\_id**  **HAVING COUNT(\*) > (**  **SELECT AVG(like\_count) FROM (**  **SELECT COUNT(\*) AS like\_count FROM likes GROUP BY story\_id**  **) AS like\_subquery**  **)**  **);** |

1. **Users who commented on more than one story:SUBQUERY**

|  |
| --- |
| **SELECT username FROM users WHERE user\_id IN (**  **SELECT user\_id FROM comments GROUP BY user\_id HAVING COUNT(DISTINCT story\_id) > 1**  **);** |

1. **Procedure to add a new comment:**

|  |
| --- |
| **DELIMITER //**  **CREATE PROCEDURE add\_comment(IN p\_story\_id INT, IN p\_user\_id INT, IN p\_comment TEXT)**  **BEGIN**  **INSERT INTO comments (story\_id, user\_id, comment) VALUES (p\_story\_id, p\_user\_id, p\_comment);**  **END //**  **DELIMITER ;** |

1. **Procedure to get total likes for a story:**

|  |
| --- |
| **DELIMITER //**  **CREATE PROCEDURE get\_story\_likes(IN p\_story\_id INT, OUT total\_likes INT)**  **BEGIN**  **SELECT COUNT(\*) INTO total\_likes FROM likes WHERE story\_id = p\_story\_id;**  **END //**  **DELIMITER ;** |