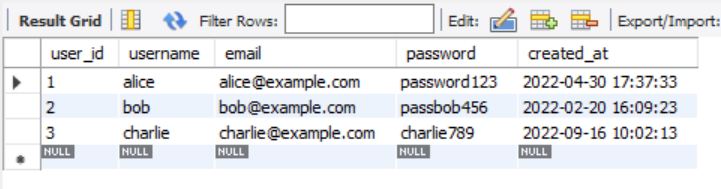
**UrbanShadows (SQL Case Study)**

**Q1. Write an SQL query to retrieve all records from the users table, including all columns and all users.**

|  |
| --- |
| SELECT \* FROM users; |

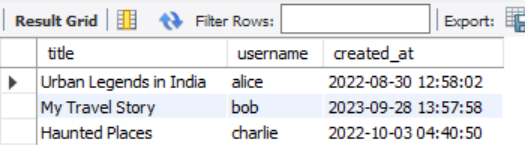
****

**Explanation:**

* SELECT \*: Selects all columns.
* FROM users: Specifies the table to retrieve data from.
* **Purpose**: Get complete data (all fields) for all users.

**Q2.** **Write an SQL query to list all blogs along with the corresponding user names who posted them. Assume there is a blogs table and a user’s table with a foreign key relationship**

|  |
| --- |
| SELECT b.title, u.username, b.created\_at FROM blogs b  INNER JOIN users u ON b.user\_id = u. user\_id; |

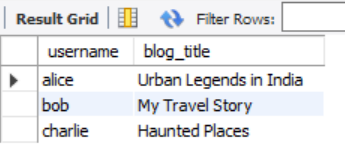
****

**Explanation:**

* blogs b: Alias b for the blogs table.
* INNER JOIN: Combines rows only when there's a match in both tables.
* ON b.user\_id = u.user\_id: Join condition; matches blog authors with users.
* **Purpose**: Show which user posted which blog and when.

**Q3.** **Write an SQL query to list all users and their blogs, if any. Include users even if they have not written any blogs.**

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

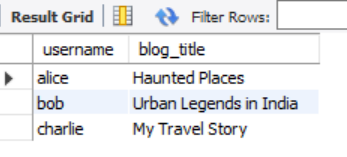
****

**Explanation:**

* LEFT JOIN: Includes all users even if they didn’t write a blog.
* b.title AS blog\_title: Alias to give a clear column name.
* **Purpose**: Find users with or without blogs.

**Q4.** **Write an SQL query to retrieve the names of users who liked blogs, along with the titles of the blogs they liked.**

|  |
| --- |
| SELECT u.username, b.title AS blog\_title FROM likes l  RIGHT JOIN blogs b ON l.blog\_id = b. blog\_id  RIGHT JOIN users u ON l.user\_id = u. user\_id; |

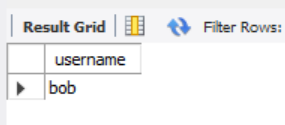
****

**Explanation:**

* likes l: Likes table with alias.
* JOIN blogs/users: Match liked blog with blog title and username.
* **Purpose**: See who liked what blog.

**Q5. Write an SQL query to retrieve the names of users who have commented on a specific blog, identified by its title.**

|  |
| --- |
| SELECT username FROM users  WHERE user\_id IN (  SELECT user\_id FROM comments  WHERE blog\_id = (SELECT blog\_id FROM blogs WHERE title = 'Urban Legends in India')  ); |

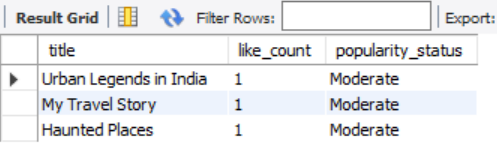
****

**Explanation:**

* Nested query:
  + Inner-most SELECT blog\_id: Get ID of the blog with given title.
  + Middle SELECT user\_id: Find users who commented on that blog.
  + Outer SELECT username: Fetch usernames.
* **Purpose**: Get names of users who commented on a particular blog.

**Q6.** **Write an SQL query to show the popularity of each blog based on the number of likes it has received. Include the blog title and the total number of likes.**

|  |
| --- |
| SELECT b.title, COUNT(l. like\_id) AS like\_count, CASE  WHEN COUNT(l. like\_id) >= 2 THEN 'Popular'  WHEN COUNT(l. like\_id) = 1 THEN 'Moderate'  ELSE 'Unnoticed'  END AS popularity\_status  FROM blogs b  LEFT JOIN likes l ON b.id = l.blog\_id  GROUP BY b. blog\_id; |

****

**Explanation:**

* LEFT JOIN: Include blogs with 0 likes.
* COUNT(l.like\_id): Total likes per blog.
* CASE: Conditions to label popularity.
* GROUP BY b.blog\_id: Aggregate likes per blog.
* **Purpose**: Determine how liked each blog is.

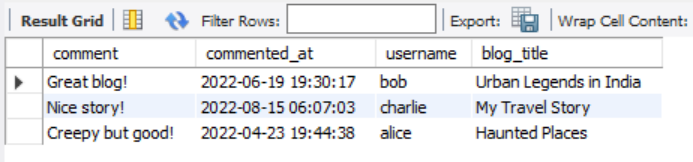
**Q7. Write an SQL statement to create a view that displays all comments along with the corresponding blog titles and the usernames of the commenters.**

|  |
| --- |
| CREATE VIEW comment\_details AS  SELECT c.comment, c.commented\_at, u.username, b.title AS blog\_title  FROM comments c  JOIN users u ON c.user\_id = u user\_id  JOIN blogs b ON c.blog\_id = b.blog\_id; |

|  |
| --- |
| SELECT \* FROM comment\_details; |

**Explanation:**

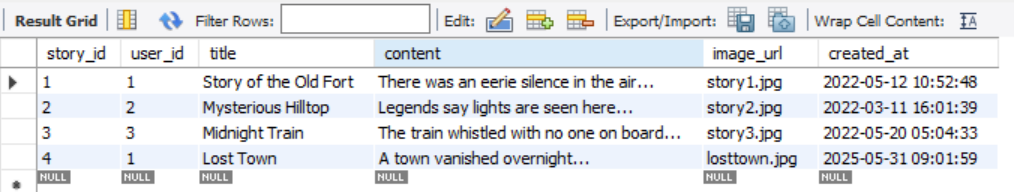
* CREATE VIEW: Defines a reusable virtual table.
* JOINs: Connect comments to blog and user.
* **Purpose**: Quick access to rich comment data for UI or reports.

****

**Q8.** **Write a stored procedure to insert a new story into the stories table. The procedure should accept parameters such as title, content, user ID, and date.**

|  |
| --- |
| DELIMITER $  CREATE PROCEDURE AddStory (  IN p\_user\_id INT,  IN p\_title VARCHAR(255),  IN p\_content TEXT,  IN p\_image\_url VARCHAR(255)  )  BEGIN  INSERT INTO stories (user\_id, title, content, image\_url, created\_at)  VALUES (p\_user\_id, p\_title, p\_content, p\_image\_url, NOW());  END $  DELIMITER ; |

|  |
| --- |
| CALL AddStory(1, 'Lost Town', 'A town vanished overnight...', 'losttown.jpg'); |

****

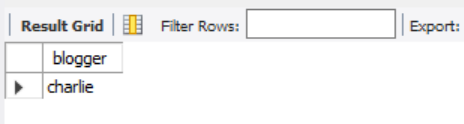
**Explanation:**

* CREATE PROCEDURE: Defines a routine.
* IN: Input parameters to pass values.
* NOW(): Inserts current timestamp.
* **Purpose**: Simplify adding stories by running one command.

**Q9.** **Write a cursor in SQL to iterate through all users who have posted blogs and display their usernames one by one. Simulate the logic using appropriate cursor declarations and control flow.**

|  |
| --- |
| DELIMITER $  CREATE PROCEDURE ShowBloggers()  BEGIN  DECLARE done INT DEFAULT FALSE;  DECLARE v\_name VARCHAR(255);  DECLARE cur CURSOR FOR  SELECT DISTINCT u.username  FROM users u  JOIN blogs b ON u.user\_id = b.user\_id;  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;  OPEN cur;  read\_loop: LOOP  FETCH cur INTO v\_name;  IF done THEN  LEAVE read\_loop;  END IF;  SELECT v\_name AS blogger;  END LOOP;  CLOSE cur;  END $  DELIMITER ; |

|  |
| --- |
| CALL ShowBloggers(); |

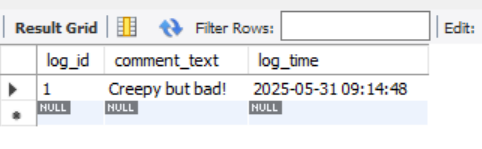
****

**Explanation:**

* CURSOR: Iterates through query results row-by-row.
* HANDLER: Exits loop on last row.
* LOOP: Fetches and displays one username at a time.
* **Purpose**: Simulate record-by-record processing in SQL.

**Q10.** **Write a trigger that logs an entry whenever a new comment is added to the comments table. The log should record details such as the comment ID, user ID, blog ID, and timestamp.**

|  |
| --- |
| CREATE TABLE comment\_log (  log\_id INT AUTO\_INCREMENT PRIMARY KEY,  comment\_text TEXT,  log\_time DATETIME  );  DELIMITER $  CREATE TRIGGER after\_comment\_insert  AFTER INSERT ON comments  FOR EACH ROW  BEGIN  INSERT INTO comment\_log (comment\_text, log\_time)  VALUES (NEW.comment, NOW());  END $  DELIMITER ; |

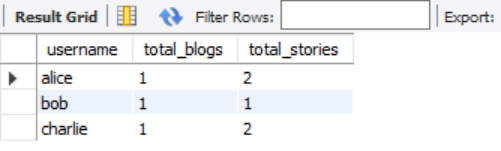
****

**Explanation:**

* TRIGGER: Executes automatically on INSERT.
* NEW.comment: Refers to the inserted row.
* AFTER INSERT: Executes after a new comment is saved.
* **Purpose**: Log every comment in a separate table for auditing.

**Q.11. Write an SQL query using aggregate functions and GROUP BY to show the total number of blogs and stories posted by each user.**

|  |
| --- |
| SELECT u.username,  COUNT(DISTINCT b.blog\_id) AS total\_blogs,  COUNT(DISTINCT s.story\_id) AS total\_stories  FROM users u  LEFT JOIN blogs b ON u.user\_id = b.user\_id  LEFT JOIN stories s ON u.user\_id = s.user\_id  GROUP BY u.user\_id; |

****

**Explanation:**

* LEFT JOIN: Include users even if they have no blogs/stories.
* COUNT(DISTINCT ...): Count only unique entries.
* GROUP BY u.user\_id: One row per user.
* **Purpose**: See each user’s total contributions.