Trigger, Procedure

Triggers

- **update_novel_rating**: This will be an **onEdit()** function in Google Apps Script that is automatically triggered whenever a new review is added to the "Reviews" sheet. The function will then read the relevant ratings, calculate the new average, and update the corresponding cell in the "Novels" sheet.

```
CREATE TRIGGER update_novel_rating

AFTER INSERT ON Reviews

FOR EACH ROW

BEGIN

    UPDATE Novels

    SET rating = (SELECT AVG(rating) FROM Reviews WHERE novel_id = NEW.novel_id)

    WHERE novel_id = NEW.novel_id;

END;
```

- **log_user_deletions**: This can be a manually triggered function or part of a user management function in your script that copies a user's data to an archive sheet before deleting it from the main "Users" sheet.

`log_user_deletions`: Before a user is deleted, this trigger archives their basic information into a `Deleted_Users_Log` table for historical purposes. First, you need to create the log table:

```
CREATE TABLE Deleted_Users_Log (

    user_id INT,

    username VARCHAR(255),

    email VARCHAR(255),

    role ENUM('Reader', 'Writer', 'Admin', 'Developer'),

    deleted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP

);
```

Then, create the trigger:

```
CREATE TRIGGER log_user_deletions
BEFORE DELETE ON Users
FOR EACH ROW
BEGIN
    INSERT INTO Deleted_Users_Log (user_id, username, email, role)
    VALUES (OLD.user_id, OLD.username, OLD.email, OLD.role);
END;
```

Can use (ledger table) to store log

Procedures become Google Apps Script Functions:

- **GetTrendingNovels(time_period)**: This will be a custom function in your script that queries the "Novels" sheet based on views, likes, and recent reviews.

```
CREATE PROCEDURE GetTrendingNovels(IN time_period VARCHAR(10))
BEGIN
    SELECT N.*
    FROM Novels N
    LEFT JOIN Reviews R ON N.novel_id = R.novel_id
    WHERE
        CASE
            WHEN time_period = 'daily' THEN R.created_at >= NOW() - INTERVAL 1 DAY
            WHEN time_period = 'weekly' THEN R.created_at >= NOW() - INTERVAL 1 WEEK
            WHEN time_period = 'monthly' THEN R.created_at >= NOW() - INTERVAL 1 MONTH
```

```
            ELSE 1=1
        END
    GROUP BY N.novel_id
    ORDER BY (N.views + N.likes + COUNT(R.review_id)) DESC;
END;
```

- **ToggleWishlist(user_id, novel_id)**: This function will search the "User_Wishlist" sheet for a matching user_id and novel_id pair and either add or remove the row.

Adds a novel to a user's wishlist:

```
CREATE PROCEDURE AddToWishlist(IN p_user_id INT, IN p_novel_id INT)
BEGIN
    INSERT INTO User_Wishlist (user_id, novel_id)
    VALUES (p_user_id, p_novel_id);
END;
```

Removes a novel from a user's wishlist:

```
CREATE PROCEDURE RemoveFromWishlist(IN p_user_id INT, IN p_novel_id INT)
BEGIN
    DELETE FROM User_Wishlist
    WHERE user_id = p_user_id AND novel_id = p_novel_id;
END;
```

- **ToggleWishlist(user_id, novel_id)**: Adds or removes a novel from a user's wishlist by inserting or deleting a record in the User_Wishlist table. (need to separate into two procedure)

- **ToggleFollow(current_user_id, author_user_id)**: Follows or unfollows an author by inserting or deleting a record in the User_Follows table.

Allows a user to follow an author:

```
CREATE PROCEDURE FollowAuthor(IN p_follower_id INT, IN p_following_id INT)

BEGIN

    INSERT INTO User_Follows (follower_id, following_id)

    VALUES (p_follower_id, p_following_id);

END;
```

Allows a user to unfollow an author:

```
CREATE PROCEDURE UnfollowAuthor(IN p_follower_id INT, IN p_following_id INT)

BEGIN

    DELETE FROM User_Follows

    WHERE follower_id = p_follower_id AND following_id = p_following_id;

END;
```

- **UpdateReadingProgress(user_id, novel_id, episode_id)**: Creates or updates a record in User_Reading_Progress to save the user's last read chapter. (optional, if want to keep)

```
CREATE PROCEDURE UpdateReadingProgress(IN p_user_id INT, IN p_novel_id INT, IN p_episode_id INT)

BEGIN

    INSERT INTO User_Reading_Progress (user_id, novel_id, last_read_episode_id)

    VALUES (p_user_id, p_novel_id, p_episode_id)

    ON DUPLICATE KEY UPDATE

    last_read_episode_id = p_episode_id,

    updated_at = CURRENT_TIMESTAMP;
```

```
END;
```