Exercise: Writing Custom Queries

In this exercise, you'll practice techniques like writing derived queries, using **@EntityGraph** to optimize fetching, leveraging the **@Query** annotation for custom queries, and working with projections to retrieve partial data efficiently.

Steps

- Populate the database with a few users and profiles. Set the loyalty points of these profiles to 5, 10, and 20.
- Add a derived query method in **ProfileRepository** to find all profiles where loyalty points is greater than a given value.
- Call this method to find profiles with more than 2 loyalty points.
- Print the **ID** of each profile.
- Modify your code to also print the **email** of the user associated with each profile.
- Run the application and check the console. Notice the **N+1 problem.**
- Fix the issue, run the application again and confirm that the issue is fixed.
- Modify the derived query method to sort results by the user's email.
- The method name is now too long and unreadable. Make the necessary changes to improve the code.
- Change the return type of the method to **List<UserSummary>**, where UserSummary is a projection interface that includes only **id** and **email.**
- Notice that this method no longer belongs to ProfileRepository since it now represents user data rather than profile data. Move it to UserRepository. Make the necessary changes and make sure the application still works.