LAB 7 part C Answer

Why the Application Still Works with Lazy Loading

1. **Transactional Context:**
   - The @Transactional annotation on the AccountService class ensures that all methods within the service are executed within a transactional context. This means that the database session remains open throughout the transaction, allowing lazy-loaded entities to be accessed without encountering a **LazyInitializationException.**
2. **Deferred Entity Loading:**
   - Lazy loading defers the loading of related entities until they are explicitly accessed. In a transactional context, this deferred loading happens seamlessly because the persistence context (Hibernate session) is still active.
3. **Efficient Data Handling:**
   - Since the application operations are typically encapsulated within service methods that are transactional, lazy loading ensures that related entities are fetched only when required, leading to more efficient data handling and improved application performance.

Testing the Changes

- **Without @Transactional Annotation:**
  - Removing the @Transactional annotation would result in a LazyInitializationException when accessing lazy-loaded entities outside the transaction scope. This is because the persistence context would be closed, and the entities would no longer be available for loading.
- **With @Transactional Annotation:**
  - Re-adding the @Transactional annotation ensures that the persistence context remains active, and lazy-loaded entities can be fetched as needed without any issues.

By transitioning from eager loading to lazy loading and ensuring that the service methods are transactional, the Bank application achieves optimized performance and resource management. The application continues to work seamlessly with lazy loading due to the active transactional context provided by the @Transactional annotation, ensuring efficient and on-demand data retrieval.