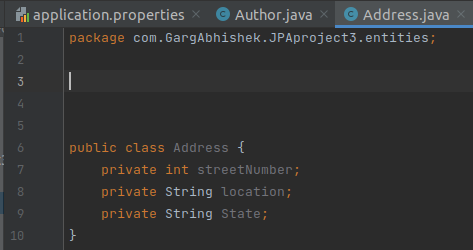
Exercise: Spring Data JPA with Hibernate Part 3

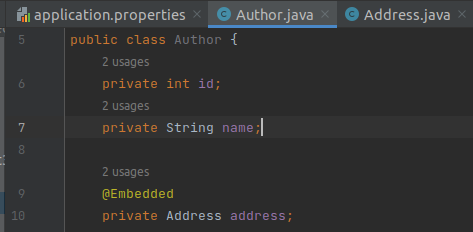
1. Create a class Address for Author with instance variables streetNumber, location, State.

Answer:



1. Create instance variable of Address class inside Author class and save it as embedded object.

Answer:



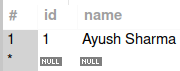
1. Introduce a List of subjects for author.

Answer:

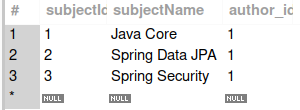


1. Persist 3 subjects for each author.

Answer:



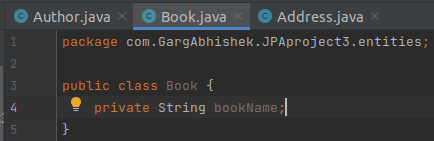
Author Table



Subject Table

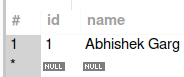
1. Create an Entity book with an instance variable bookName.

Answer:

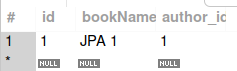


1. Implement One to One mapping between Author and Book.

Answer:



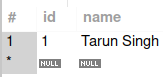
Author Table



Book Table

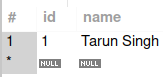
1. Implement One to Many Mapping between Author and Book(Unidirectional, BiDirectional and without additional table ) and implement cascade save.

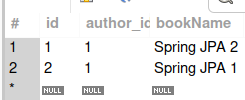
Answer:





UniDirectional

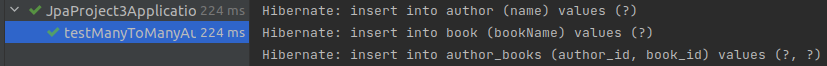




BiDirectional

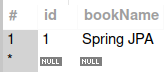
1. Implement Many to Many Mapping between Author and Book.

Answer:





Author Table



Book Table



Bridge Table (author\_book)

1. Which method on the session object can be used to remove an object from the cache?

Answer:

We can use session evict() method to remove a single object from the hibernate first level cache. We can use session clear() method to clear the cache i.e delete all the objects from the cache.

1. What does @transactional annotation do?

Answer:

When a class declares @Transactional on itself or its members, Spring creates a proxy that implements the same interface(s) as the class you’re annotating. In other words, Spring wraps the bean in the proxy and the bean itself has no knowledge of it. A proxy provides a way for Spring to inject behaviors before, after, or around method calls into the object being proxied. The generated proxy object is supplied with a TransactionInterceptor, which is created by Spring. So when the @Transactional method is called from client code, the TransactionInterceptor gets invoked first from the proxy object, which begins the transaction and eventually invokes the method on the target bean. When the invocation finishes, the TransactionInterceptor commits/rolls back the transaction accordingly.