**Task 04 || Spring Boot, ORM (JPA, Hibernate)**

1. **Duration:** 3 working days
2. **Description:**

* This task has 2 parts.
  1. Connect with a Relational Database (MySQL) in place of H2 DB.
  2. In Task 03, Book DTO and Book Entity has the same properties. In this task we will add some properties to the BookEntity to trace and version our data.
* create feature branch for task 04 from release branch:

go to release branch terminal and execute,

*git checkout -b feature/bim/atique-202015/task-04*

*[Note: use your nick name and emp\_id in place of “atique-202015”]*

* Part 01: Connect with MySQL DB
* Add below properties in *application.properties* file to configure MySQL DB

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/bookinfo  
spring.datasource.username=root  
spring.datasource.password=password  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true

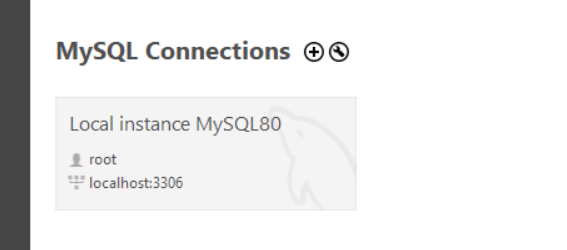
*[Note: we have already used Some of above properties in previous task, just replace their values. Some are new, simply add them, and notice* spring.datasource.url, *our database name will be* bookinfo*]*

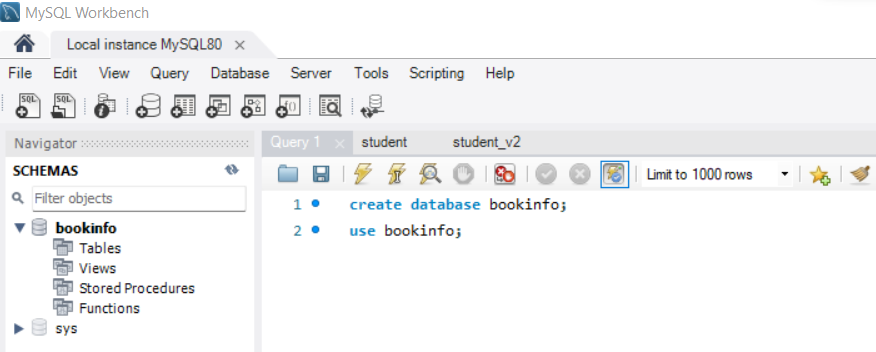
* Remove H2 DB driver dependency from *build.gradle → dependencies*

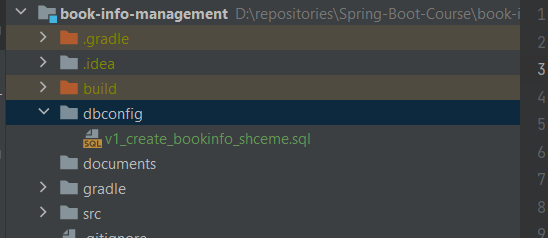
runtimeOnly 'com.h2database:h2'

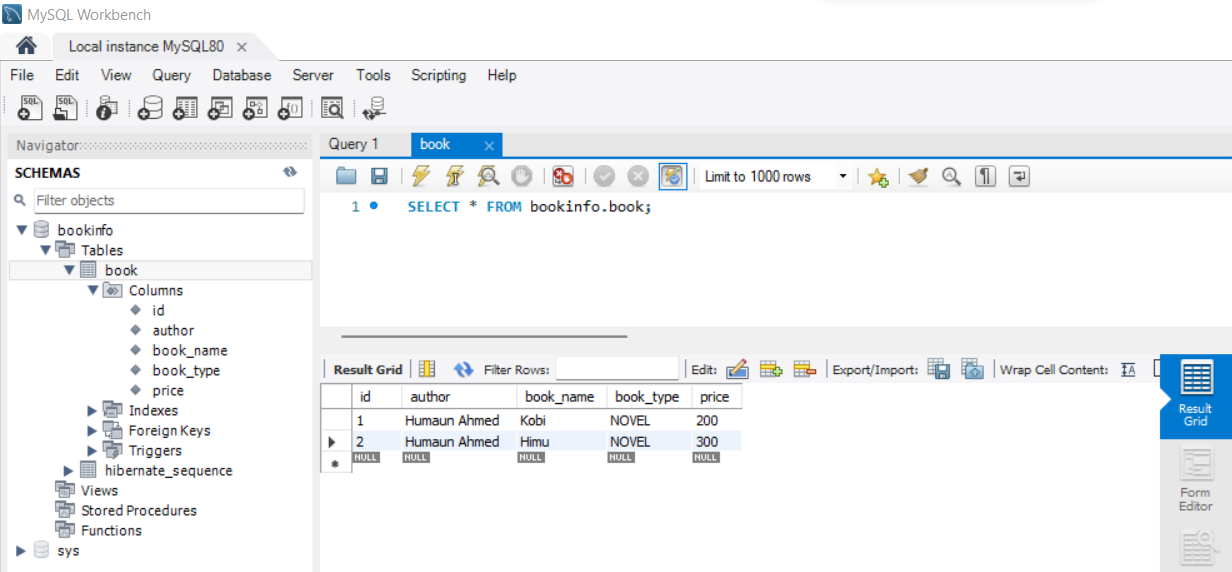
* Add MySQL DB driver dependency to *build.gradle → dependencies*

runtimeOnly 'mysql:mysql-connector-java'

* You need to create the database manually, to do so first Log in to the ***"root"*** user in MySQL Workbench.
* Create bookinfo database/scheme in MySQL Workbench query console by following the screenshot bellow. bookinfo database will appear in left side navigation panel. Press refresh button in case database not appear.



* Create *“dbconfig”* directoryunder *src/main/java/com.example.bookinfomanagement* and keep database creation query there in a .sql file like
* Run the Application, it will automatically create required tables in bookinfo database.
* Hit POST localhost:8084/book-info-manager-1.0/api/v1/book API with proper request, see data in *book* table.



* Done. Did you notice the beauty of JPA? we have changed our entire DBMS from H2 to MySQL with some configuration changes in *application.properties* and MySQL driver dependency in *build.gradle*, without any changes in codebase and everything just worked out of the box.
* We have used 2 new properties

1. spring.jpa.hibernate.ddl-auto=update
2. spring.jpa.show-sql=true

for first one, we have set the value to “*update*” so, hibernate will update DB based on our annotated entity classes. That is why our table “*book*” created automatically our application startup.

What are the other options for this property? see: <https://stackoverflow.com/questions/42135114/how-does-spring-jpa-hibernate-ddl-auto-property-exactly-work-in-spring>

for second one, the SQL generated by JPA will be logged in console, thus we can see what JPA is doing under the hood. See the console log

* open terminal, add & commit your changes for part 01 in your task 04 feature branch

*git add .*

*git commit -m "[bim] task 04 part 01"*

* Part 02: Changes in BookEntity.
* Save and query book info using random id is not a good idea, specially from user perspective. In addition, there is a unique ID (ISBN) to identify a book that is already known to everyone or at least available on Google. See: <https://en.wikipedia.org/wiki/ISBN>
* Use existing ID column as DB primary key, use @GeneratedValue annotation on id column in book entity so MySQL will automatically generate the value. The primary key is more resides in the domain context rather than a way to communicate with a third party via DTO.
* To interact with user, use a unique property named “ISBN” in book entity and book DTO.
* Add 3 more columns to book entity, CREATED (Date), UPDATED(Date), VERSION (Long).
* Use @PrePersist on a method in your entity to auto update your created and updated column. Use @Version on your version column, this will update your version column after each update. See learning metarials.
* So, columns of BOOK table will be:

*ID, ISBN, BOOK\_NAME, BOOK\_TYPE, AUTHOR, CREATED, UPDATED, VERSION*

* Name your table in book entity as *“BOOK\_V2”.* Hibernate will create new table on application startup.
* New API Doc.

1. Create Book

|  |  |
| --- | --- |
| URI | /api/v1/book |
| Desc | POST 'localhost:8084/book-info-manager-1.0/api/v1/book'  {  "isbn" : "9789844590014",  "bookName" : "Himu",  "bookType" : "NOVEL",  "author" : "Humaun Ahmed",  "price" : 200  }  Res:  {  "isbn" : "9789844590014",  "bookName" : "Himu",  "bookType" : "NOVEL",  "author" : "Humaun Ahmed",  "price" : 300  }  *[Note: id (Long), bookName (String), bookType (Enum), author (String), price (Long)]* |

1. Get Book

|  |  |
| --- | --- |
| URI | /api/v1/book/{isbn} |
| Desc | GET 'localhost:8084/book-info-manager-1.0/api/v1/book/9789844590014'  Res:  {  "isbn" : "9789844590014",  "bookName" : "Himu",  "bookType" : "NOVEL",  "author" : "Humaun Ahmed",  "price" : 200  } |

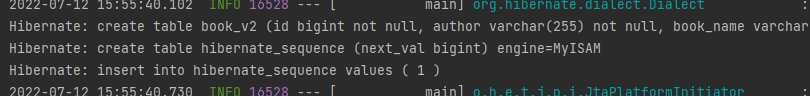
1. Update Book

|  |  |
| --- | --- |
| URI | /api/v1/book/{isbn} |
| Desc | PUT 'localhost:8084/book-info-manager-1.0/api/v1/book/9789844590014'  Req:  {  "isbn" : "9789844590014",  "bookName" : "Himu update",  "bookType" : "NOVEL",  "author" : "Humaun Ahmed",  "price" : 300  }  Res:  {  "isbn" : "9789844590014",  "bookName" : "Himu update",  "bookType" : "NOVEL",  "author" : "Humaun Ahmed",  "price" : 300  } |

1. Delete Book

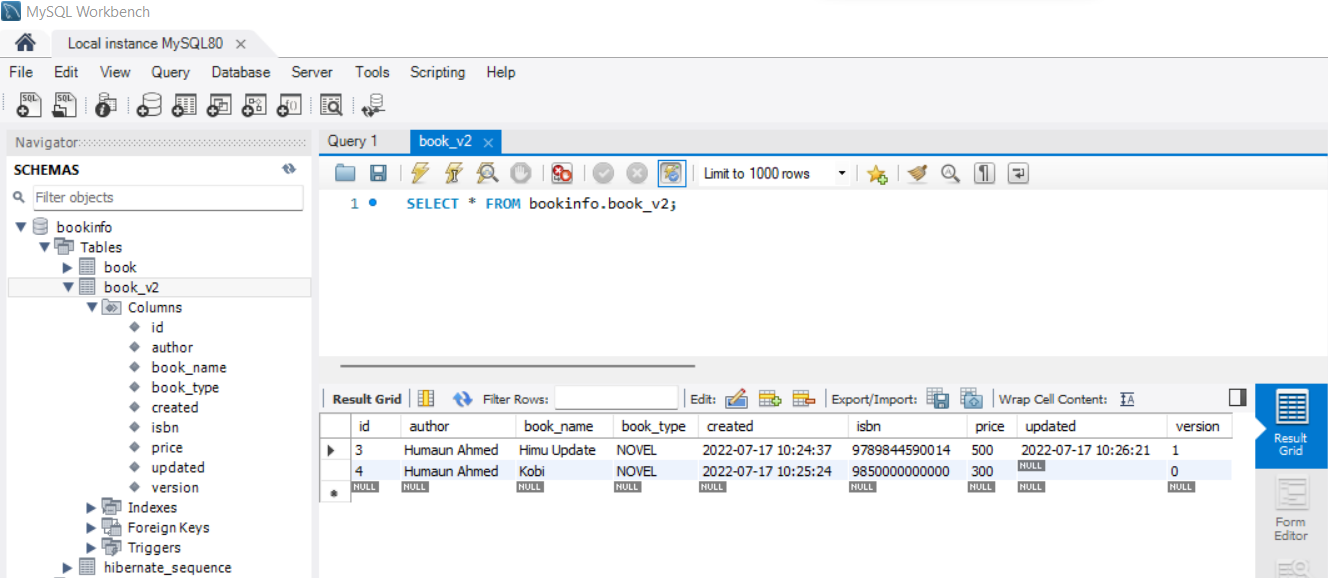
|  |  |
| --- | --- |
| URI | /api/v1/book/{isbn} |
| Desc | DELETE 'localhost:8084/book-info-manager-1.0/api/v1/book/9789844590014' |

* You need to use derived queries in JPA Repository to access book info using ISBN. See learning metarials.
* Run the application after implementation, it will automatically create required tables in bookinfo database.



*Table created “book\_v2”. Another one “hibernate\_sequence” for primary key*

* Create and Update some book info using API and see data in *book* table.



* open terminal, add, commit & push your changes for part 02 in your task 04 feature branch

*git add .*

*git commit -m "[bim] task 04 part 02"*

*git push*

1. **Learning metatrails**

* <https://www.baeldung.com/spring-data-derived-queries>
* <https://www.baeldung.com/jpa-entity-lifecycle-events>

1. **Helper Project**

* Simple CRUD operations to manage student info is done in this project using MySQL Database and JPA in 4 layers.
* You have already cloned this project in task 02
* fetch branches: *git fetch*
* Migration from H2 DB to MySQL is done in branch “hp/student-info-manage-mysql-jpa-part-01”

To see checkout branch: *git checkout hp/student-info-manage-mysql-jpa-part-01*

* Separate DB primary key and created, updated, version column related work done in branch “hp/student-info-manage-mysql-jpa-part-02”

To see checkout branch: *git checkout hp/student-info-manage-mysql-jpa-part-02*

* Description can be found in README.md file.
* Postman collection kept in *documents* package.
* You can see the project structure and browse the code.

1. **What to submit**

Prepare a doc with the answer of bellow questions:

* In part 02 we have created a new table “BOOK\_V2” for book info to keep things simple. Imagine your book-info-manager-1.0 application already on live and BOOK table has huge amount data.

So, what will be the problem if we create new table?

What problems we have to deal with if we want to proceed with existing “BOOK” table?

Suggest a solution to this migration problem.

* If you work with a live application, which option would you choose for the "spring.jpa.hibernate

.ddl-auto" property? and Why?

* How derived queries in JPA repository works?
* Write about the newly added dependencies in *build.gradle* file.
* Write about the newly added properties in *application.properties* file.
* Write about new annotations we used:

@GeneratedValue, @PrePersist, @Version

* Submit your code for task 04 in GitHub. See how to submit section.

1. **How to submit**

* Name your doc as: <nick name>\_<emp\_id>\_<task id>.

Ex: atique\_202015\_task\_04.docx

* Upload your doc in documents folder
* commit your changes in your task 04 feature branch:

*git commit -m "[bim] task 04"*

* Push the changes to your feature branch: *git push*
* Create a merge req in GitHub [your feature branch→ your release branch]

Ex: *feature/bim/atique-202015/task-04 → bim/atique-202015/release-1.0*

* Follow up instructors’ feedback on GitHub merge request.