**Creating Hibernate Code from Database Schema**

While it is not time consuming to create Hibernate Entities for small database schemas, it may take some effort to write something up for a database scheme that has 100’s of tables. Effort to create the entities include

* Under the database tables, columns, data types
* Create and IntelliJ project with pom.xml
* Write the java code for each entity. Should match the database schema definition
* Create the hibernate configuration file

We can eliminate this effort by creating Hibernate entities using LLM. In this post, I have explained by attempt to create the needed Hibernate java code using

* OpenAI as the LLM that anchors an Agent workflow
* Mistral AI’s codestral model to generate java code and XML configuration files
* LangGraph as the framework to execute agents and define the agent workflow
* SQLAlchemy to access the database and create a text representation of the schema

**Results**

I had my database running on Neon Serverless Postgres platform. The schema for the DB I created is shown in the following image.

A diagram of a computer

Description automatically generated

Snapshot of the project structure and one entity class is shown below.

A screenshot of a computer

Description automatically generated

The result of running the App.java class that accesses employee, customers and orders table through Employee, Customer, Order hibernate entities is given below.

A screenshot of a computer

Description automatically generated

**Approach – Step 1: Create a maven project with pom.xml**

First step in the process of to create a maven project with a pom.xml will needed dependencies and plugins. I used a simple prompt and tool to pass to a LangGraph graph to do this work for me.

<<insert prompt used>>

<<insert the tool code>>

**Approach – Step 2: Write Code, Write Config, Maven Build**

This is the core step where all the code generation is done. I used a chain of thought prompt here to build the code in multiple steps. While Mistral AI Codestral was use for code generation, I used OpenAI to hold it all together as the LLM for graph’s reasoning.

<<insert prompt here>>

The flow to create the hibernate code and configuration is as below.



Tool – get\_schema: Called by the LLM to get the definition for the objects in the database schema.

<<insert code here>>

Tool – create\_hibernate\_code: Called by the LLM to write the hibernate code for e very entity in the database as identified by the LLM.

<<insert code here>>

Tool – create\_hibernate\_config: Called by the LLM to write the hibernate configuration file based on the entities created in the previous step

<<insert code here>>

Tool – save-file: Called to save the generated java and XML file to the file system

<<insert code here>>

Tool – run\_maven\_command: Called to run the maven build, compile commands once the code is ready. Needs you to pass your maven install path.

<<insert code here>>

**LangGraph Graph that ties it all together**

The entity that hold this all together is a Langgraph agent given below.

<<code here>>

**Output Parsers to get just Java and XML code from LLM calls.**

Use the pydantic classes given below to ensure Codestral gives back only Java and XML code in its response in the create\_hibernate\_code and create\_hibernate\_config tools.

<<insert code here>>

**Generated Java Code**

<https://github.com/dheerajrhegde/HibernateCodeGenerator/tree/main/HibernateProject>

**Python Code used for generation**

<https://github.com/dheerajrhegde/HibernateCodeGenerator/tree/main/ERDiagram2Hibernate>

**Note**

The App.java code was modified to get the data from few entities for testing purposes. Code generated by Codestral was just a barebone code to get all customers and print the Customer objects.