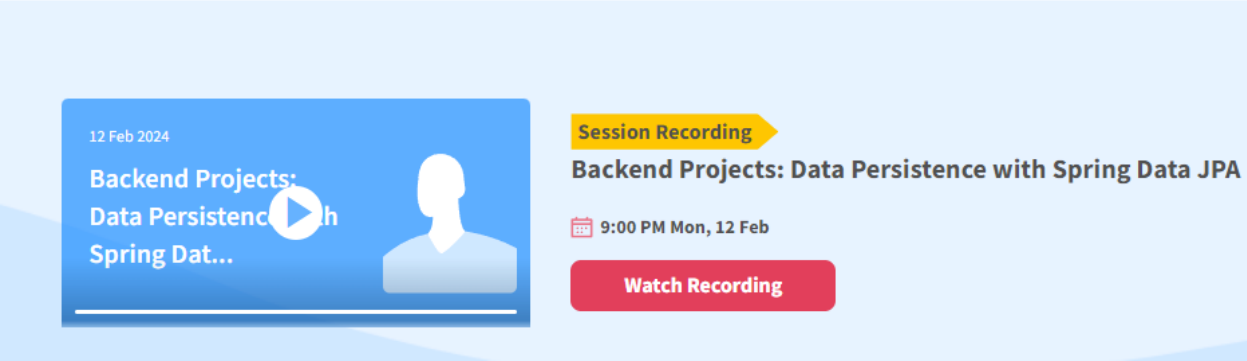


Day 197 - **Backend Projects: Data Persistence with Spring Data JPA** Mandatory

Mon, 12 Feb 2024

Session 0% | Assignment 0/0 | Additional Problems 0/0 New | Help Requ

ttps://www.baeldung.com/hibernate-identifiers

https://en.wikipedia.org/wiki/Universally_unique_identifier

https://spring.io/guides/gs/accessing-data-jpa

to read other generation logic: https://www.baeldung.com/hibernate-identifiers

add JPA Buddy extension in intellij...

Agenda

1) Introduction to ORM & JDBC
2) Introduction to JPA & Spring JPA
3) Repository pattern
4) Setting up DB in Our Product Service
5) UUID
6) Inheriting Relations in DB

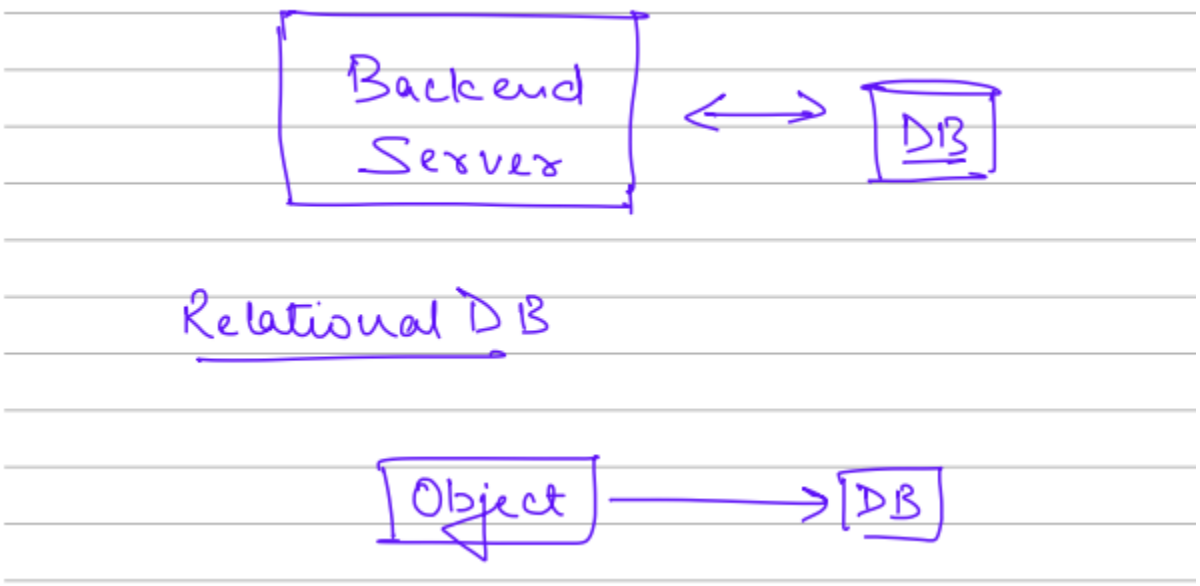
Total 3 classes on DB..

Intro to ORM & JDBC:

If you have a BE server or any services, needs to connect to a database. That's what we will talk about. When we get the products. Who was providing? FAKESTORE. FAKESTORE api keeps all data in DB. When I need to build an ecom I will keep data in DB. For our DB we will atlk about RDBMS..

SQL vs noSql we learnt in HLD.

are we gonna see NoSQL in project module? In separate class.



In harry poter there are books restricted to teacher only. In a lib if some people can read the books. So the lib eco system has rules/ guidelines.

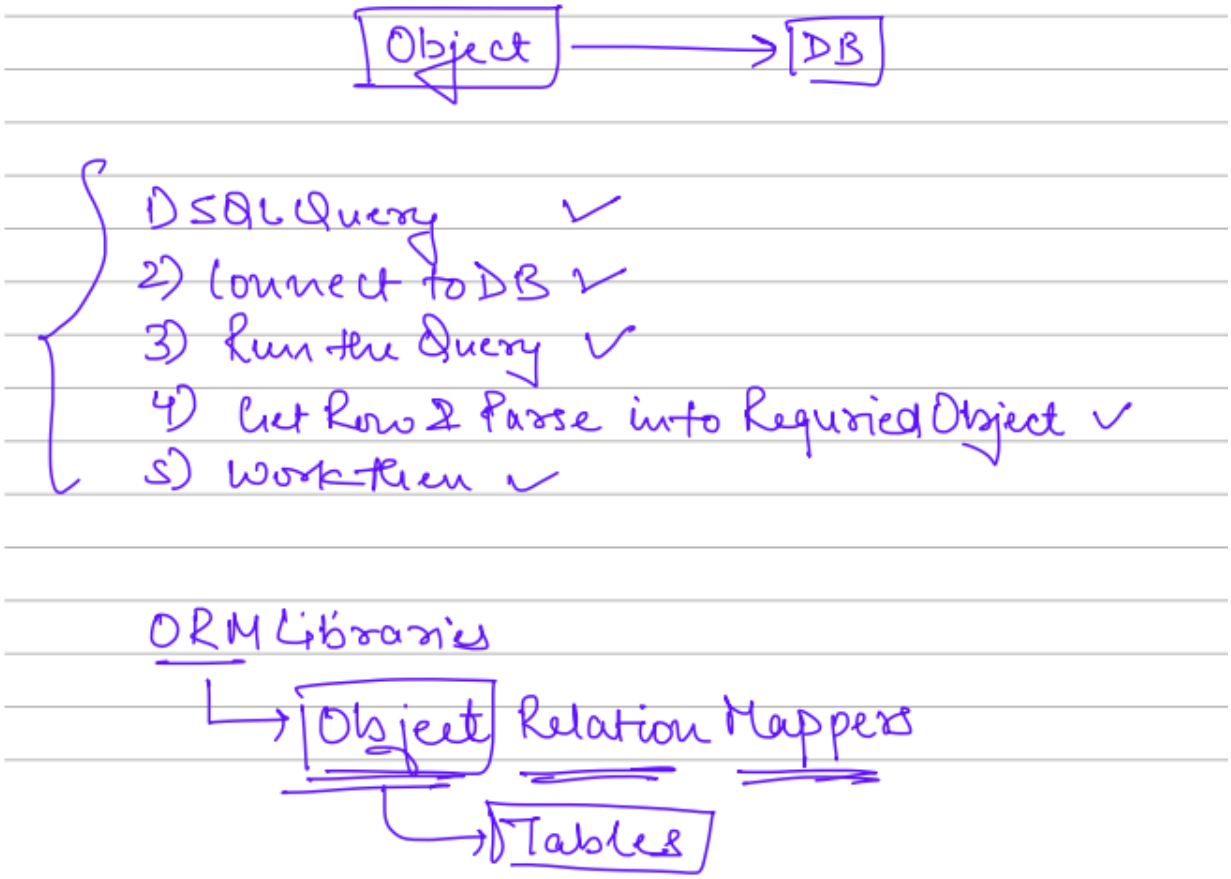
JPA is nothing but a set of rules. How you can borrow book and return book to library. In Library I ask librarian to issue me a book. So I need to make a connection with him ask for "I want x book" also "I want to return the book". JDBC is driver/ library to get data and return data to DB.

In harry porter instead of librarian. I say "x book come here" like a magic.. I get x book come flying to me.. much more easier if this magic exist. That magic is ORM. JPA is not ORM

What is ORM, JDBC, JPA?

When you send and get an obj from DB. What steps you need?

1. Write SQL query for what kinda data you need. = What book you need or you wanna return..
2. I need to connect to DB.. I reach to library
3. Run the query = go and identify the book
4. Get the Row and Parse into Required Object… = get the book
5. Work on them = read the book.



This is a very good amount of overhead. For every object we need to do these. But we don't wanna do this. Here comes the ORM library. Object Relation Mapping. It maps obj to your Tables.. in the data we have tables. ORM provides you an easy way to Work with Database and convert data into object/ Class. It can help you to write queries, convert results to an object and joins..

Provides you an easy way to work with DB and convert data into objects (classes)

→ Write Queries
→ Convert results to objects
→ Joins

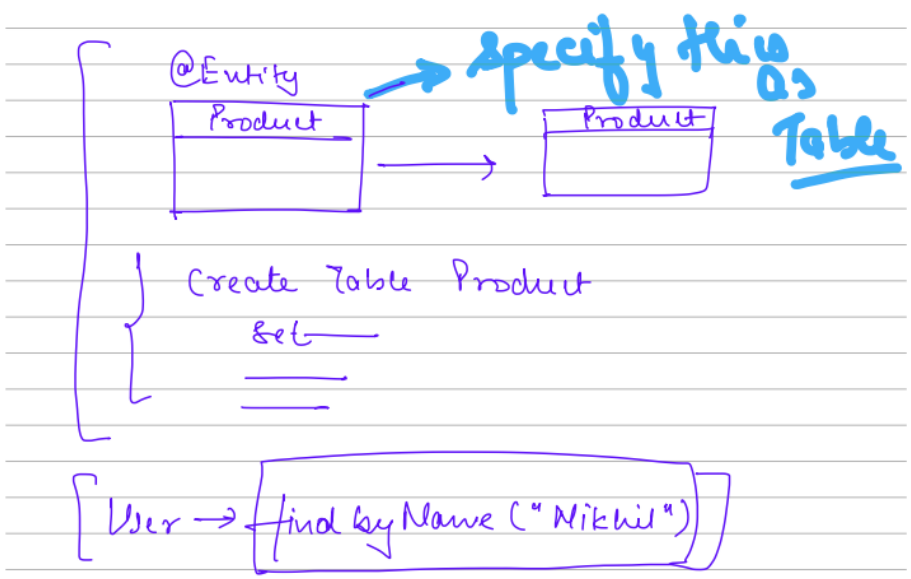
Example fo MySql: I have a ENTITY called Products and I need to convert to a TABLE.. to do this I will have to do .. CREATE Table PRODUCT..

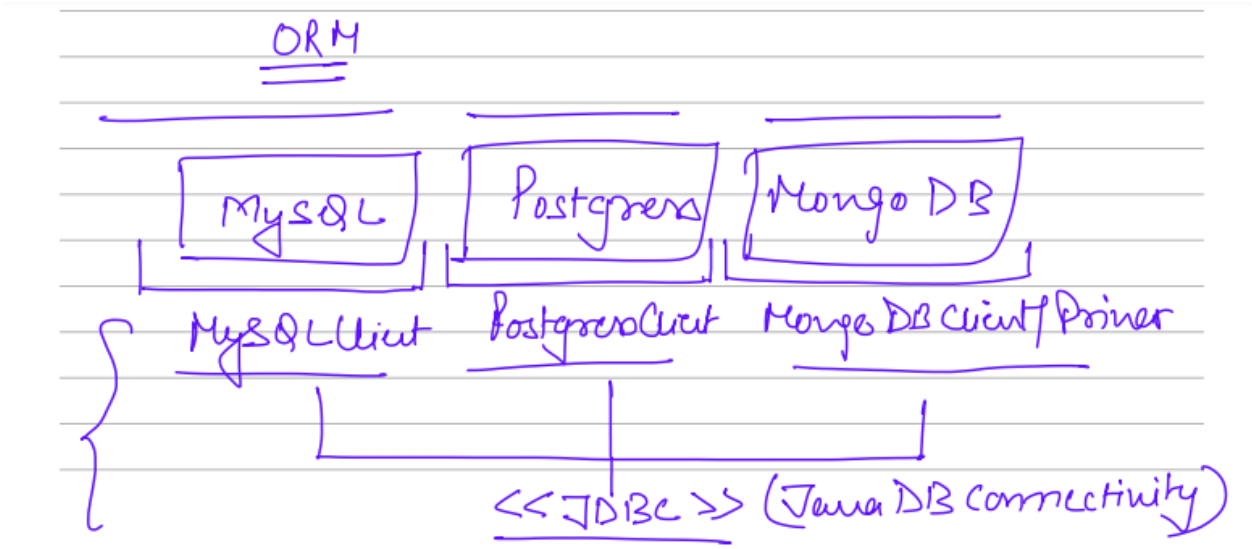
And lot of manual work…

I need to specify this as a table and let someone to do this. ORM will do this..i want to say simple thing… and pass arg..

Find by Name("Nikhil") I write this function and everything else will be taken care by system. Lets see how this gets achieved. ORM helps us with this.

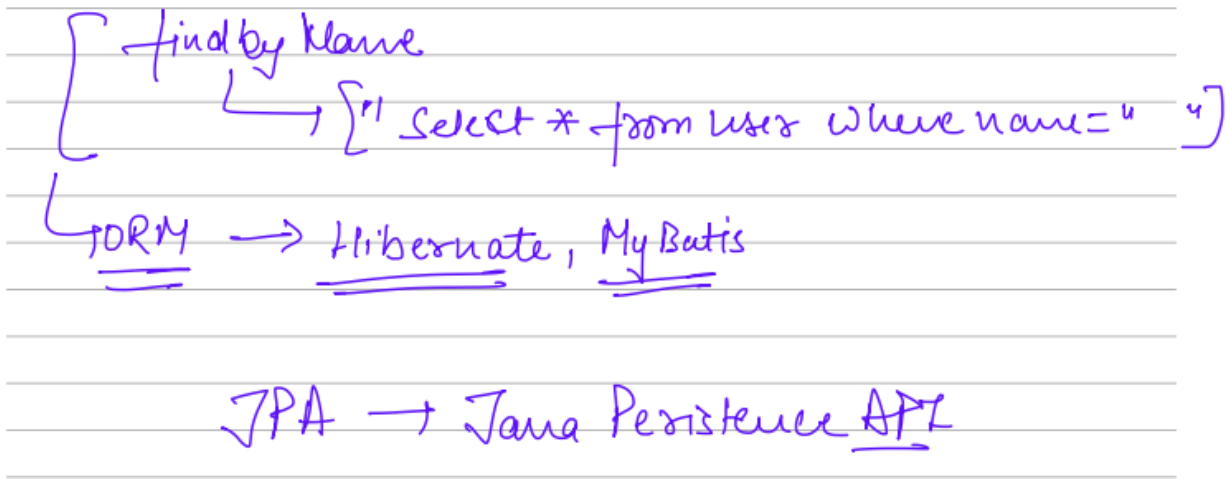
In MySql or PostGres.. to connecting all of these DB is going to be same? The way they gets connected or details they required will not be same. Then I need client for all of them client = Drivers..





To use them in our system, how can we make sure they are loosely coupled? **Interface**. This IF need to follow common convention.. similar to Bank API in Phonepe. **That connection nothing but JDBC**.

JDBC is interface or standard for java database connectivity.

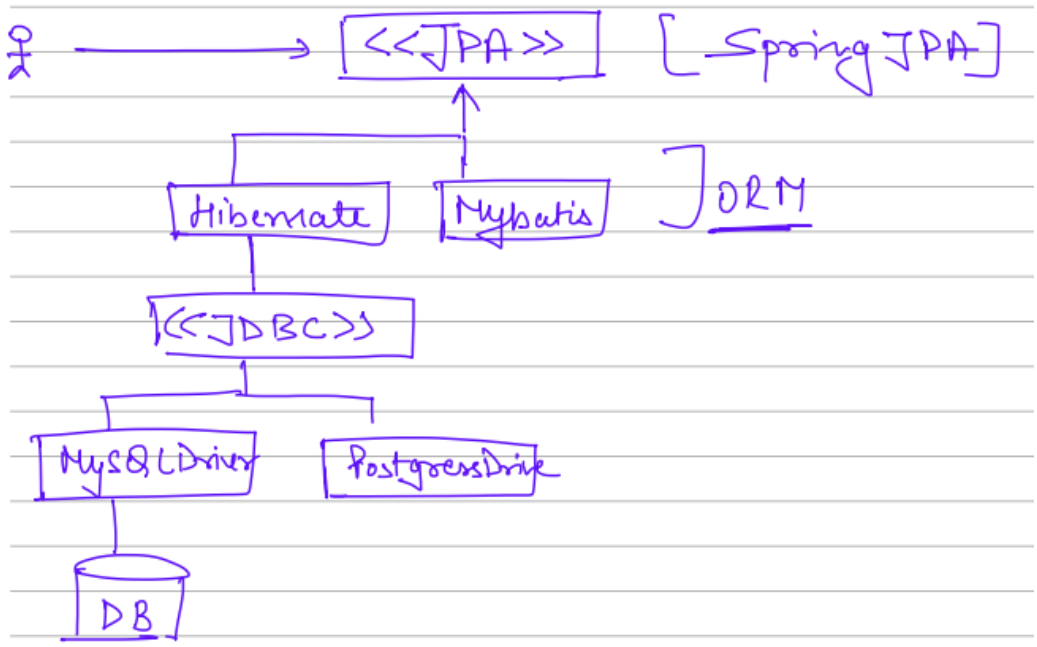


We need a platform to create query, connecting and execution. So we need another Layer to convert this to a query.. that's taken care by ORM.

Some ORMs are HiberNate, MyBetis,

We can switch to another ORM .. we use Inteface

ORM are external to app, we make it loosely coupled by IF. That IF/ standard for ORM is called JPA.



In JPA client gets conneted. In JPA we have diff ORMs, hibernate, MyBetis.. each of the ORMs will have JDBC, work with JDBC. Depending on how many database they will support they will have mysql, postgres driver etc.. These drivers will connect to the database.

We will always connect to JPA. JPA connects to ORM. Orm will connent to JDBC. JDBC will make sure withever DB you wanna connect.

We will Use Spring data JPA. Spring uses Hibernate internally…

JPA is nothing but guidelines… JPA is Interface..
Hibernate is implementation over JPA. Spring JPA uses Hibernate. Hibernate implements JPA. JPA is nothing but standards..
Spring JPA is library. Jpa = standards.. we will use spring JPA.
Hibernate uses JDBC standards to connect to DB.

Library

Spring JPA → uses Hibernate

Hibernate implements JPA (Standards)

→ uses JDBC Standards to connect to DB

What is Hibernate?

Hibernate is a Java framework that facilitates the growth of Java applications to interact with the database. It is an open-source, portable, ORM (Object Relational Mapping) tool.

It enforces the specifications of JPA full form is the Java Persistence API for information endurance. Hibernate delivers a reference performance of the Java Persistence API that makes it a tremendous choice as an ORM tool with the advantages of flexible coupling. Now let's look at the tools of ORM.

ORM Tool

An ORM tool streamlines data production, information manipulation, and data access. It is a programming strategy that maps the object to the information saved in the database. The ORM tool internally consumes the JDBC API to interact with the database.

What is JPA?

A Java Persistence API i.e. JPA is itself a specification of Java that is utilized to manage, access, and persist data between the Java objects and the relational databases. It is contemplated as a common approach for Object Relational Mapping i.e. ORM. JPA can be recognized as a bridge between object-oriented domain prototypes and relational database systems. JPA doesn't conduct any operation by itself, being a specification. Therefore, it expects execution.

So, ORM tools like Hibernate, iBatis, and TopLink execute JPA specifications for data endurance. Now you might be curious to know if there is any need for JPA or not? Well, don't worry because we will discuss the need for JPA in today's technological world.

Need of JPA

As we have observed so far, JPA is a specification. It delivers common models and functionality to ORM tools. By executing the same specification, the whole ORM tools, for instance, Hibernate, TopLink, iBatis, etc. follow the established standards. In the future, if we like to switch our application from one ORM tool to another, we can do it effortlessly.

What Is Java Persistence API?

The Java Persistence ***API*** gives a specification for prevailing, reading, and managing information from the object of your Java to the relational tables in the database. Now let's look at the framework of hibernate.

What Is Hibernate Framework?

Hibernate is an object-relational mapping treatment for Java environments. Object-relational mapping or let's say ORM is the programming method to map application domain prototype objects to the relational database tables. It is a Java established ORM tool that gives a framework for mapping application domain objects to the relational database tables and also vice versa. It provides a source for implementation of the Java Persistence API that gives rise to it as an extraordinary choice as an ORM tool with advantages of loose coupling. You should note that JPA is a specification and on the other hand Hibernate is a JPA provider or let's say execution.

What Is Spring Data JPA?

Spring Data is a component of the Spring Framework. The purpose of the Spring Data repository abstraction is to significantly lessen the amount of boilerplate code needed to execute data access layers for multiple persistence stores. Spring Data JPA is not a JPA provider as it is a library or framework that enhances as an extra layer of abstraction on the top of our JPA provider is added, such as Hibernate. As of now, you are well aware of the definition of JPA, Hibernate, and Spring Data JPA. So, finally, let's discuss the differences between hibernate and spring data JPA.

The Disparity Between Spring Data JPA and Hibernate

Hibernate is a JPA implementation, while Spring Data JPA is a JPA Data Access Abstraction as we have discussed above. Spring Data proposes a solution to GenericDao custom implementations. It can further generate JPA queries on your behalf through the methodology or method name conventions. With Spring Data, you probably utilize Hibernate, Eclipse Link, or any other JPA provider.

A very interesting advantage is that you can control transaction boundaries declaratively. Spring Data JPA is not an implementation or a JPA provider; it's hardly an abstraction used to significantly decrease the amount of boilerplate code required to enforce information or data access layers for many persistence stores. We hope this article is useful to you and if you need more information or any kind of help, we are here to help you as we have skilled data engineers. So come and contact us today!!!

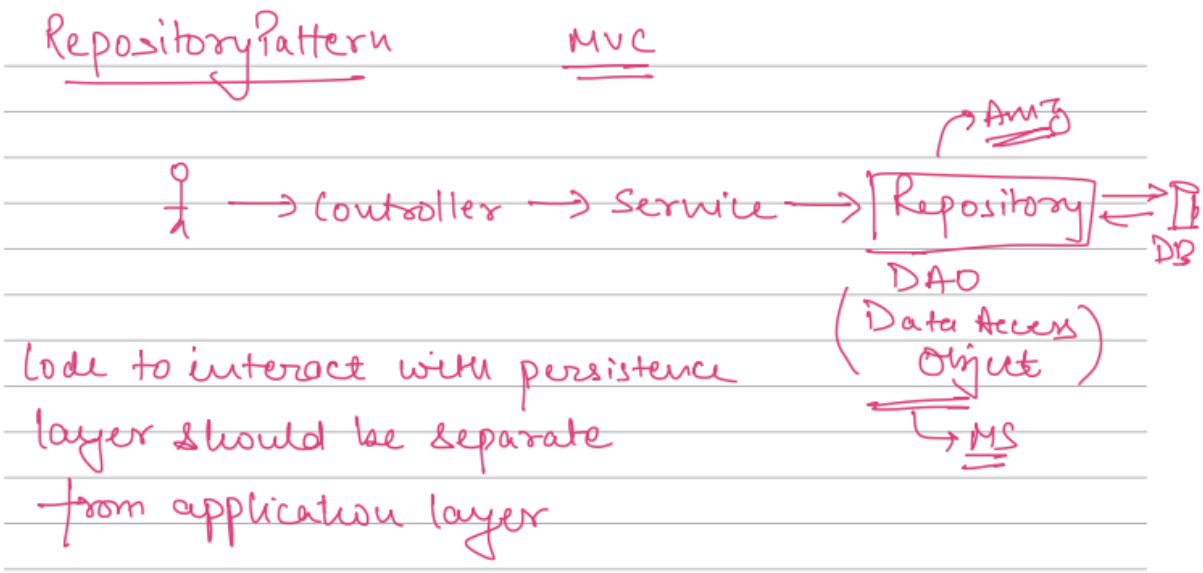
Spring boot project me you can use spring data JPA. When you use IPhone you prefer apple watch… if you use springboot spring JOA makes more sense.

Repository Pattern:

In MVC pattern we talked about it. The flow between controller to service..

Request goes from controller → service → repository (aka DAO) → DB.

Repository is nothing but DAO. Another name is DAO. Data access obj. the obj which can help you to access data from Database. Fun fact: Microsoft uses naming convention as DAO. Amazon uses Repository. This is how they use package name.



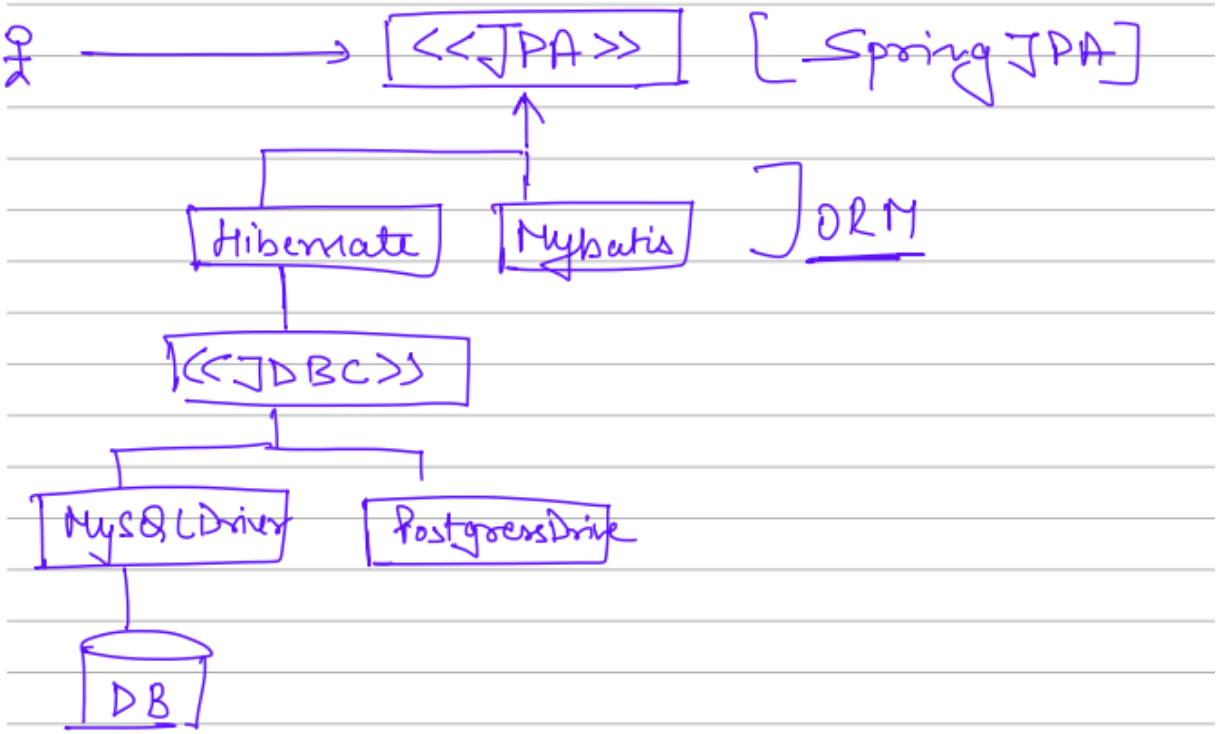
What is Repository Pattern??

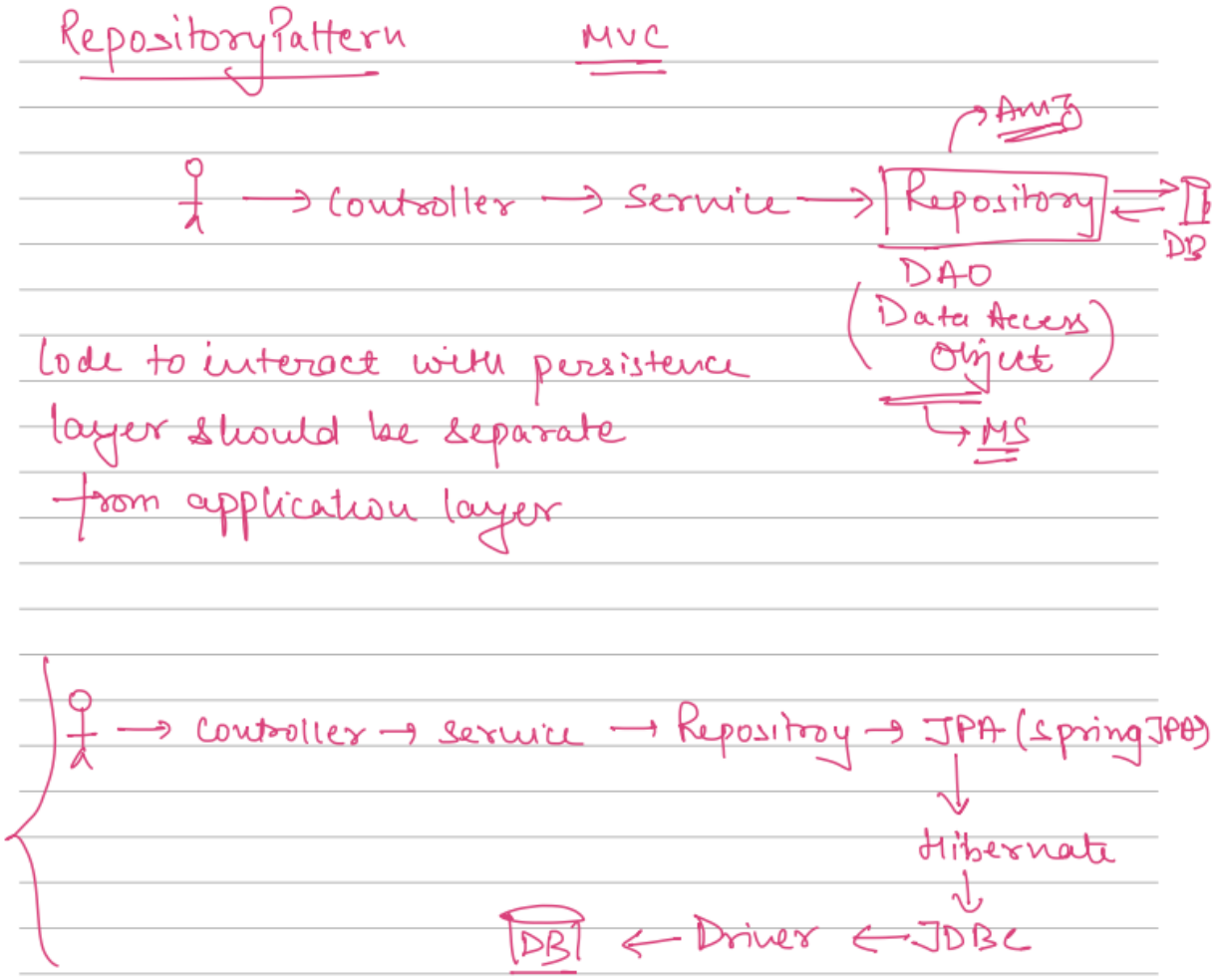
Code to interact with persistence Layer should be separate from Application Layer.

Should it be a part of service layer? No, it should have separate layer..

Another flow:

User→ controller→ service → Repository → JPA (spring data JPA) → Hibernate → JDBC → Driver → DB





Repository is nothing but DAO. The object which can help youi to access the data from DB (data access object).

MS uses DAO. Amazon uses Repository.

Whats Repository pattern:

Code to interact with Persistence Layer should be separated from application Layer. It should not be part of service Layer. Deserves a own layer.

Flow:

User connects with Controller → service → Repository → JPA (Spring data JPA) → Hibernate → JDBC → specific Driver which DB you want to connect → that's connects to your DB.

Lets see how can we connect to DB??

Mysql workbench basics – on onenote

Now we need spring data.. “getting started with spring data JPA” – google search

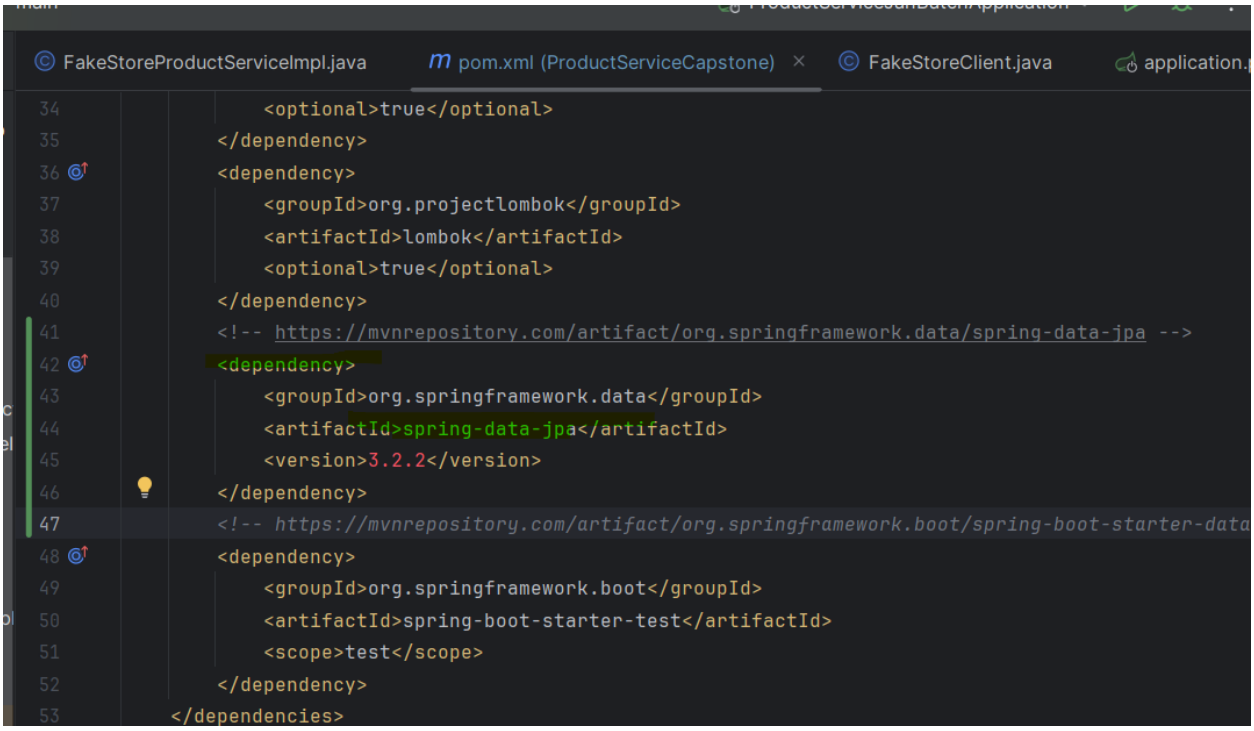
https://spring.io/guides/gs/accessing-data-jpa

first thing I need to do if I want to spring data JPA… first thing is adds a dependency.

Search: “spring data jpa mvn repository”

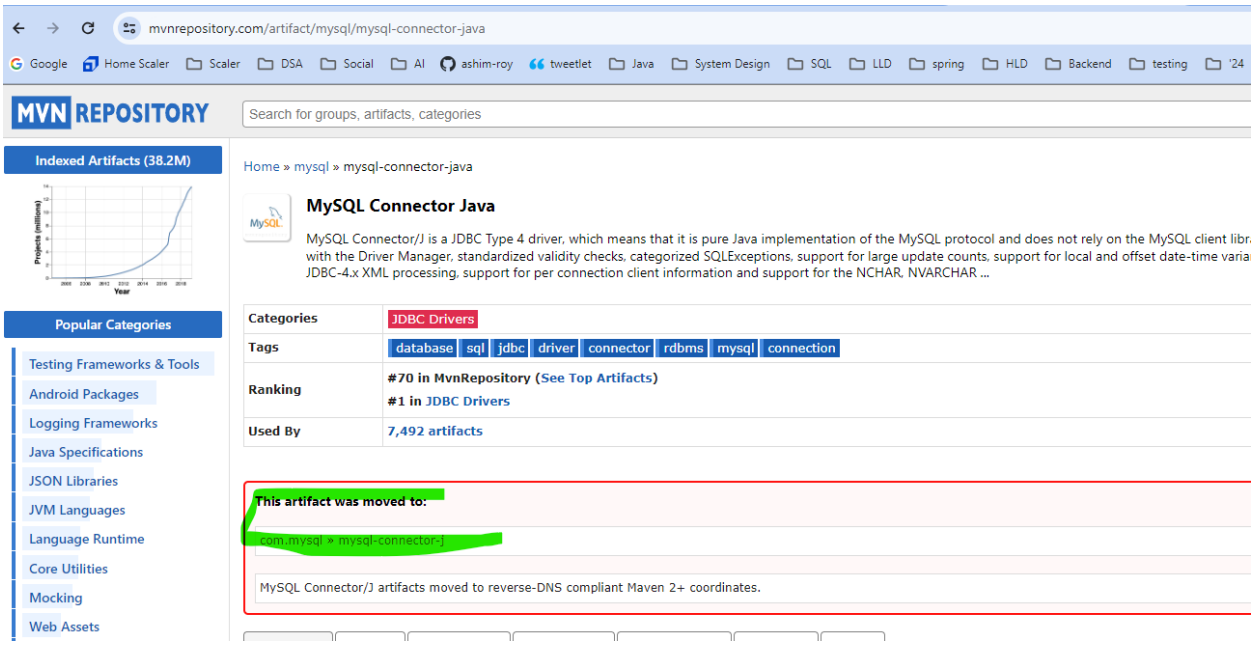
https://mvnrepository.com/artifact/org.springframework.data/spring-data-jpa/3.2.2

```
<!-- https://mvnrepository.com/artifact/org.springframework.data/spring-data-jpa -->
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>3.2.2</version>
</dependency>
```



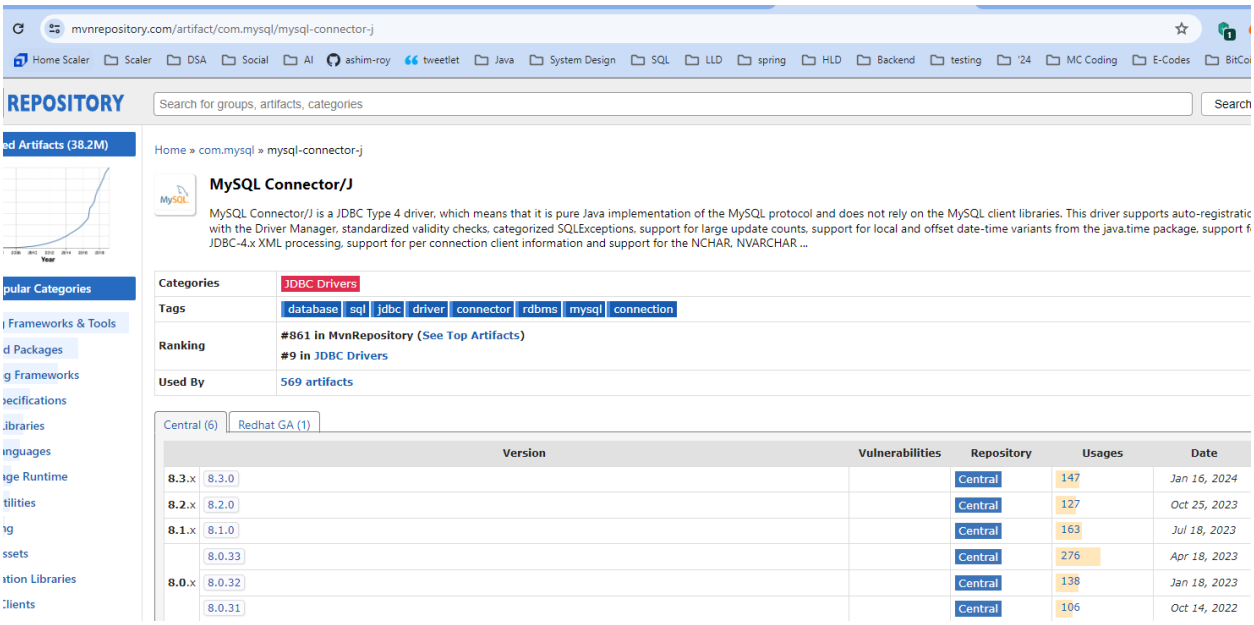
But Nikhil didn’t paste the 3..2.2 version here… lest see what happens if I don’t remove it.

Also we are using MySQL… database so we need mysql connector.. “mysql connector mvn”



Its moved here so click on the link..

https://mvnrepository.com/artifact/com.mysql/mysql-connector-j



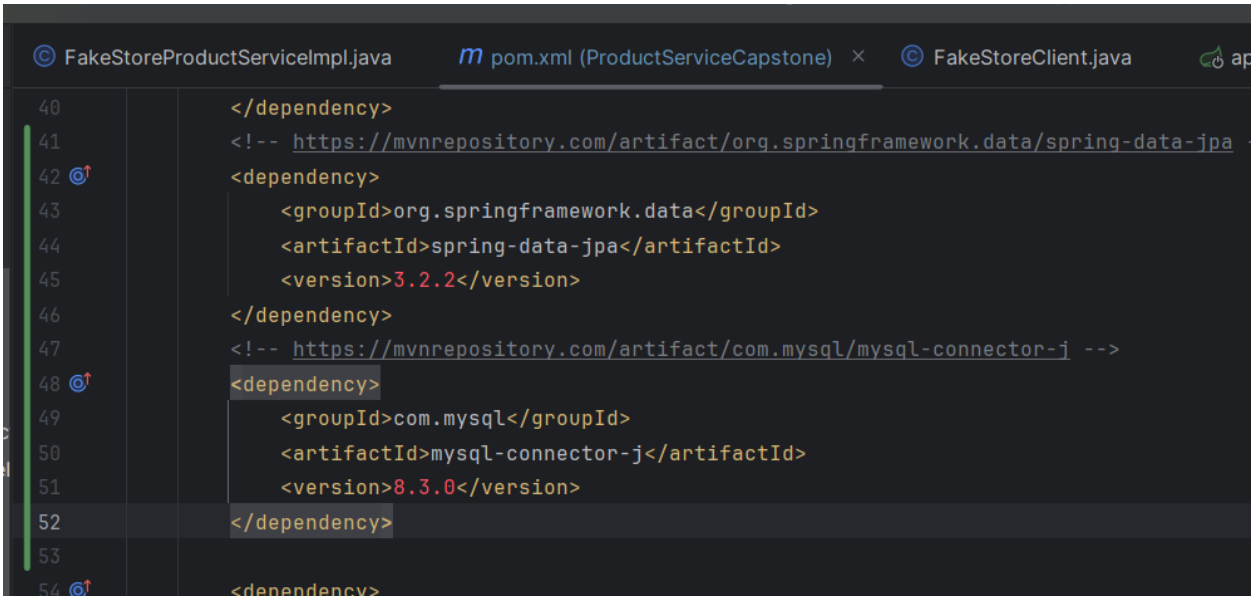
We go for 8.3.0:

https://mvnrepository.com/artifact/com.mysql/mysql-connector-j/8.3.0

<!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->

<dependency>

```
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>8.3.0</version>
</dependency>
```



Can we add dependency without version?

Declaring a dependency without version

A recommended practice for larger projects is to declare dependencies without versions and use dependency constraints for version declaration.

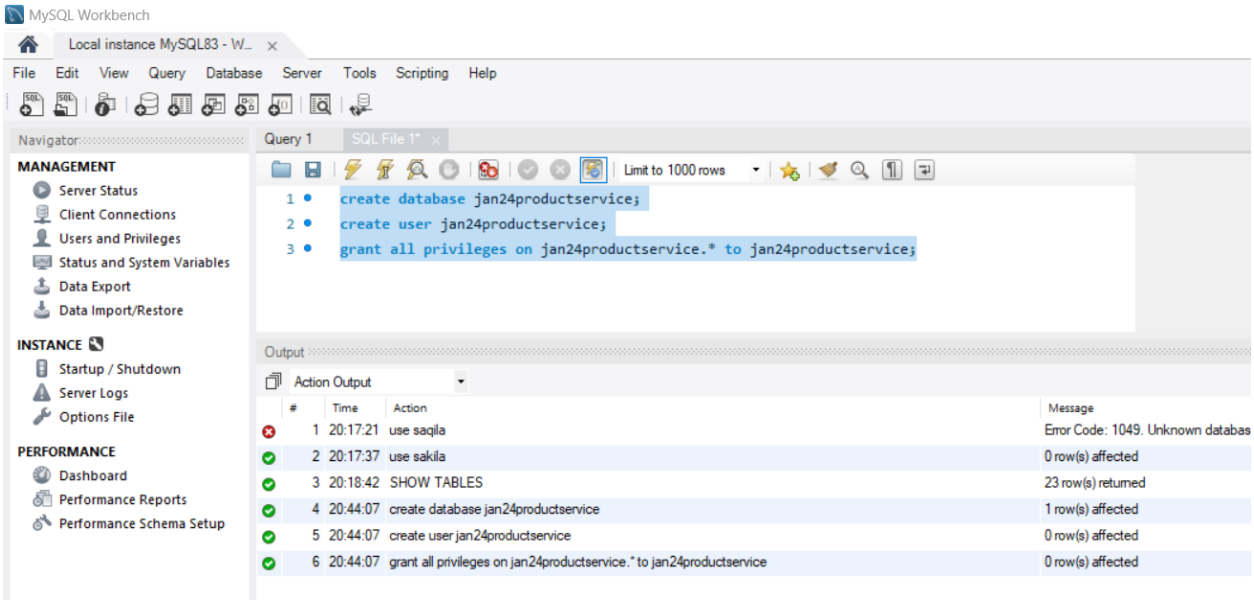
As of now we havnt created the database not we created the TABLE. If we use the connector and JPA by default it will start try to connect to a Database.. but to connect to a DB it need some BASIC information, credential, which DB etc..

SO we first create a database..

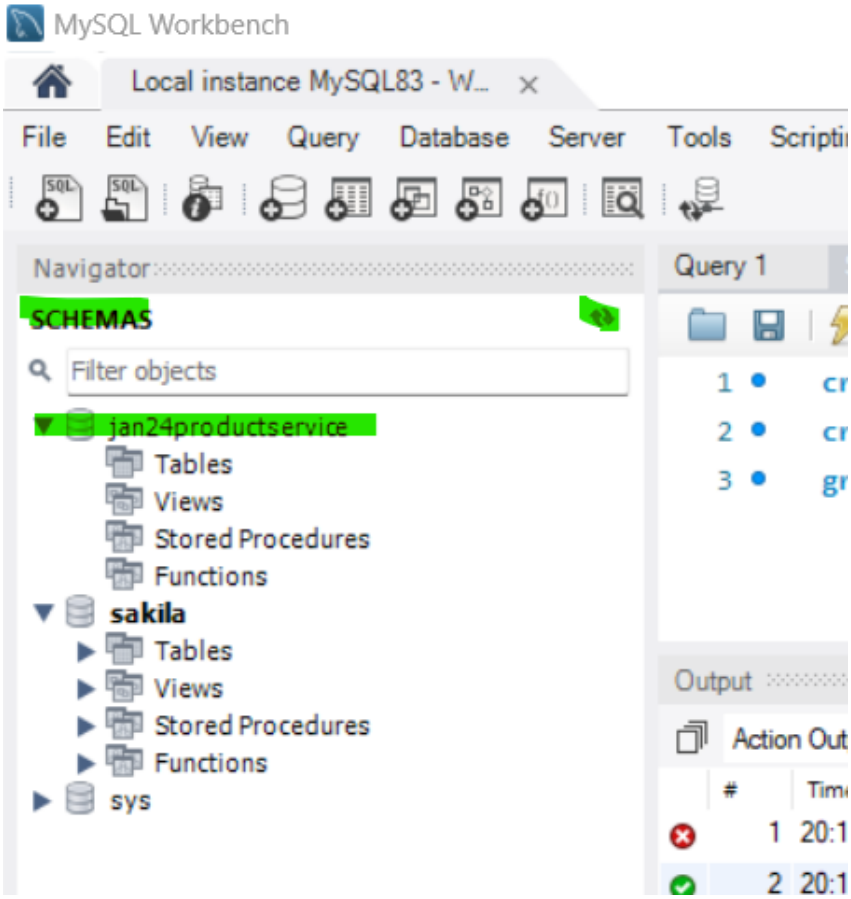
create database jan24productservice;

create user jan24productservice;

grant all privileges on jan24productservice.* to jan24productservice;



Refresh schema to see:



Things we need to connect to a DB: URL, UserNamne, Password, Database..

Lets see how JPA does it: "spring jpa database connection" : https://spring.io/guides/gs/accessing-data-mysql

Its asking me add few details, to auto configure the db its saying me to add some details...

Create the Database

Open a terminal (command prompt in Microsoft Windows) and open a MySQL client as a user who can create new users.
For example, on a Linux system, use the following command;

```
COPY$ sudo mysql --password
```

This connects to MySQL as `root` and allows access to the user from all hosts. This is **not the recommended way** for a production server.
To create a new database, run the following commands at the `mysql` prompt:

```
COPYmysql> create database db_example; -- Creates the new database
mysql> create user 'springuser'@'%' identified by 'ThePassword'; --
Creates the user
mysql> grant all on db_example.* to 'springuser'@'%'; -- Gives all
privileges to the new user on the newly created database
```

Create the `application.properties` File

Spring Boot gives you defaults on all things. For example, the default database is `H2`. Consequently, when you want to use any other database, you must define the connection attributes in the `application.properties` file.
Create a resource file called `src/main/resources/application.properties`, as the following listing shows:

```
COPYspring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/db_
example
spring.datasource.username=springuser
spring.datasource.password=ThePassword
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

```
#spring.jpa.show-sql: true
```

Here, `spring.jpa.hibernate.ddl-auto` can be `none`, `update`, `create`, or `create-drop`. See the [Hibernate documentation]() for details.

Create the `application.properties` File

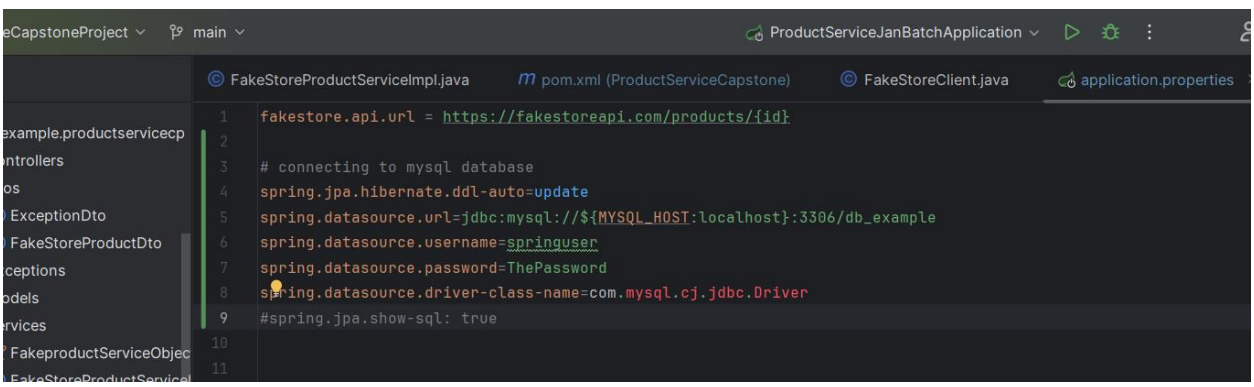
Spring Boot gives you defaults on all things. For example, the default database is `H2`. Consequently, when you want to use any other database, you must define the connection attributes in the `application.properties` file.

Create a resource file called `src/main/resources/application.properties`, as the following listing shows:

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/db_example
spring.datasource.username=springuser
spring.datasource.password=ThePassword
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
#spring.jpa.show-sql: true
```

Here, `spring.jpa.hibernate.ddl-auto` can be `none`, `update`, `create`, or `create-drop`. See the Hibernate documentation for details.

Added all details in app.ppty file.



Update the database name here..

```
FakeStoreProductServiceImpl.java    pom.xml (ProductServiceCapstone)    FakeStoreClient.java
fakestore.api.url = https://fakestoreapi.com/products/{id}

# connecting to mysql database
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost}:3306/jan24productservice
spring.datasource.username=jan24productservice

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
#spring.jpa.show-sql: true
```

Nikhil's file

```
ProductServiceJa...lication
Product.java    JpaBaseConfiguration.class    BaseModel.java    application.properties
fakestore.api.url=https://fakestoreapi.com/products/{id}

#Connection to Mysql DB,
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/jan24productservice
spring.datasource.username=jan24productservice
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

Driver: if we don't specify it will use default driver which is H2. We want mysql

Mysql jdbc driver. If we don't specify and let spring identify service will be slower as it will have all drivers. HB has to take care a lot. That's why we specify the connector and driver.

Dialect: a value which is reqd when hibernate is going to convert the objects. It need the conversion library.. if you don't give driver then which driver it will use…?

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
```

```
spring.jpa.hibernate.ddl-auto=update
```

this wil cover in 2nd class when we upgrade the schema.

```
fakestore.api.url = https://fakestoreapi.com/products/{id}

# connecting to mysql database
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost}:3306/jan24productservice
spring.datasource.username=jan24productservice
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.show-sql= true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
```

Refresh the MVN icon to load everything.. red line goes away..

Now lest run the service..

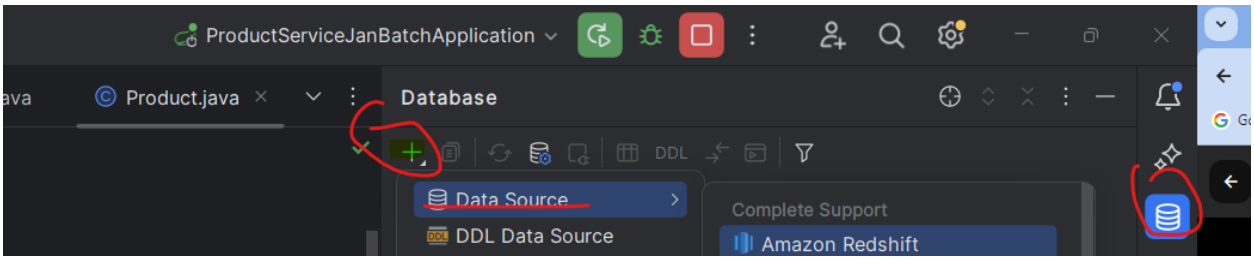
Should see something like this:

```
: Initializing Spring embedded WebApplicationContext
: Root WebApplicationContext: initialization completed in 570 ms
: HHH000204: Processing PersistenceUnitInfo [name: default]
: HHH000412: Hibernate ORM core version 6.4.1.Final
: HHH000026: Second-level cache disabled
: No LoadTimeWeaver setup: ignoring JPA class transformer
: HikariPool-1 - Starting...
: HikariPool-1 - Added connection com.mysql.cj.jdbc.ConnectionImpl@286aa749
: HikariPool-1 - Start completed.
: HHH90000025: MySQL8Dialect does not need to be specified explicitly using 'hibernate.dialect
: HHH90000026: MySQL8Dialect has been deprecated; use org.hibernate.dialect.MySQLDialect inste
: HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA
: Initialized JPA EntityManagerFactory for persistence unit 'default'
: spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed
: LiveReload server is running on port 35729
```

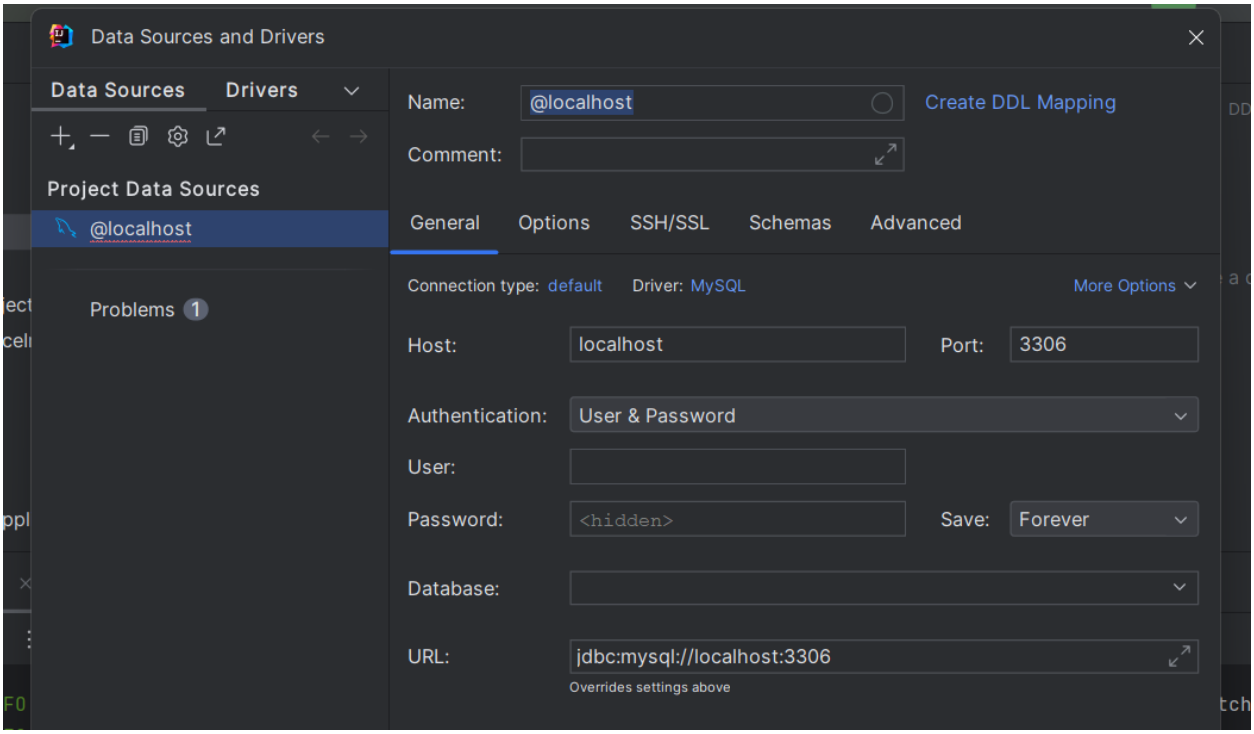
Added a connection to mySql…

Now lets try to add here in DB in intellij..

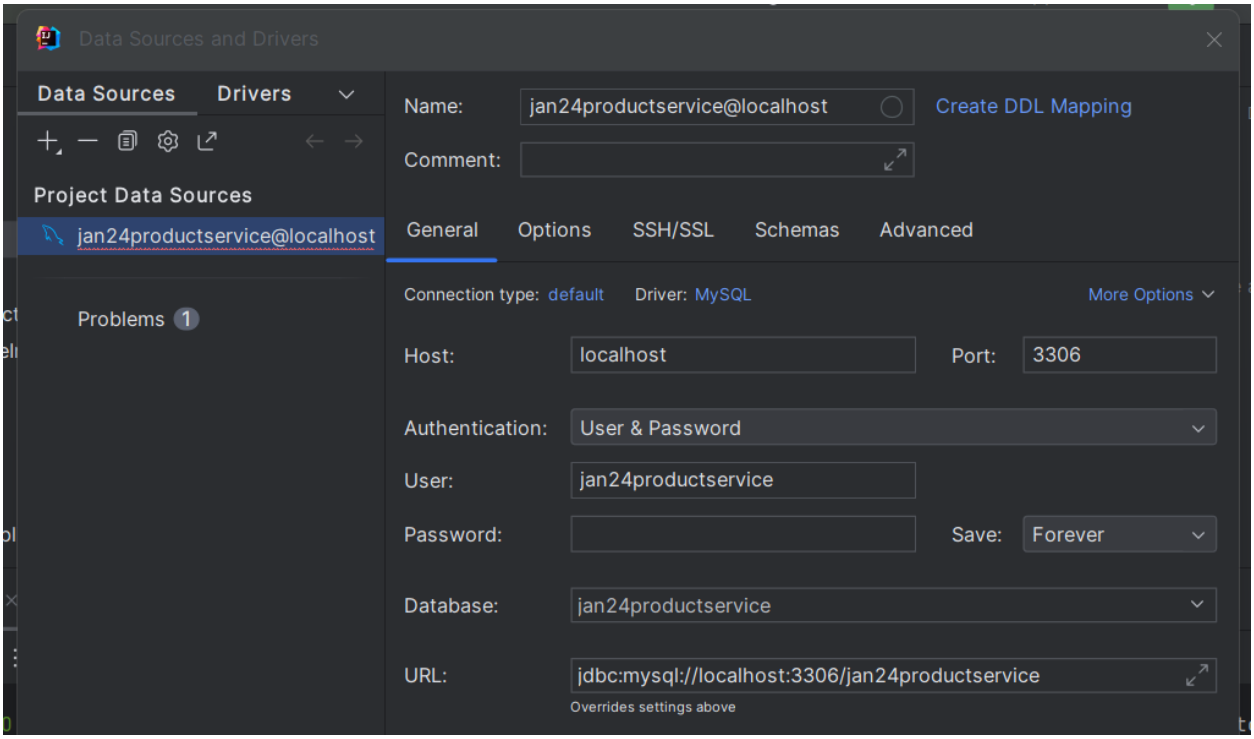
How to connect to a database: scroll down and click on mysql.

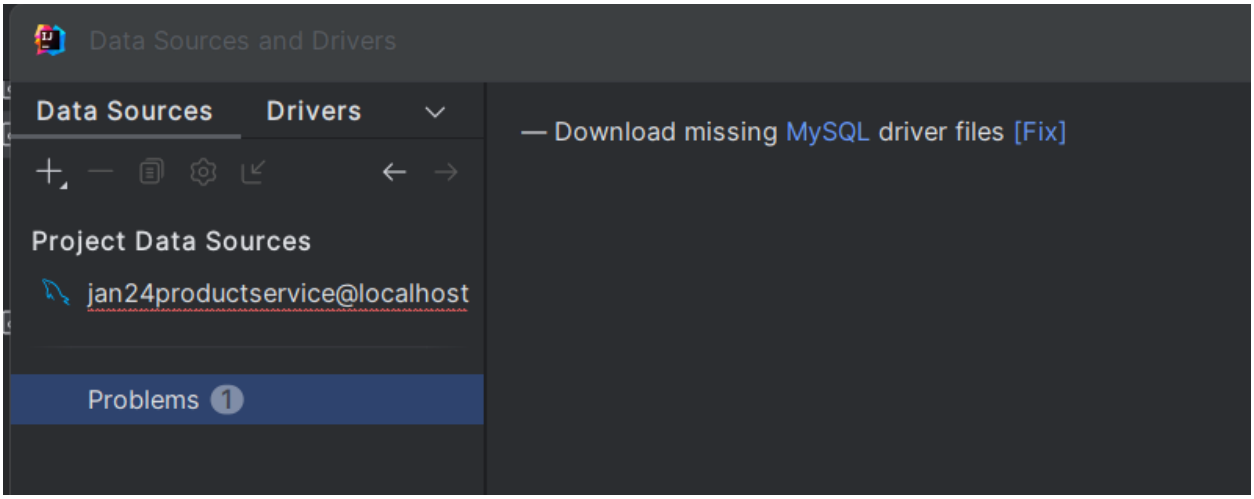


You will get this:

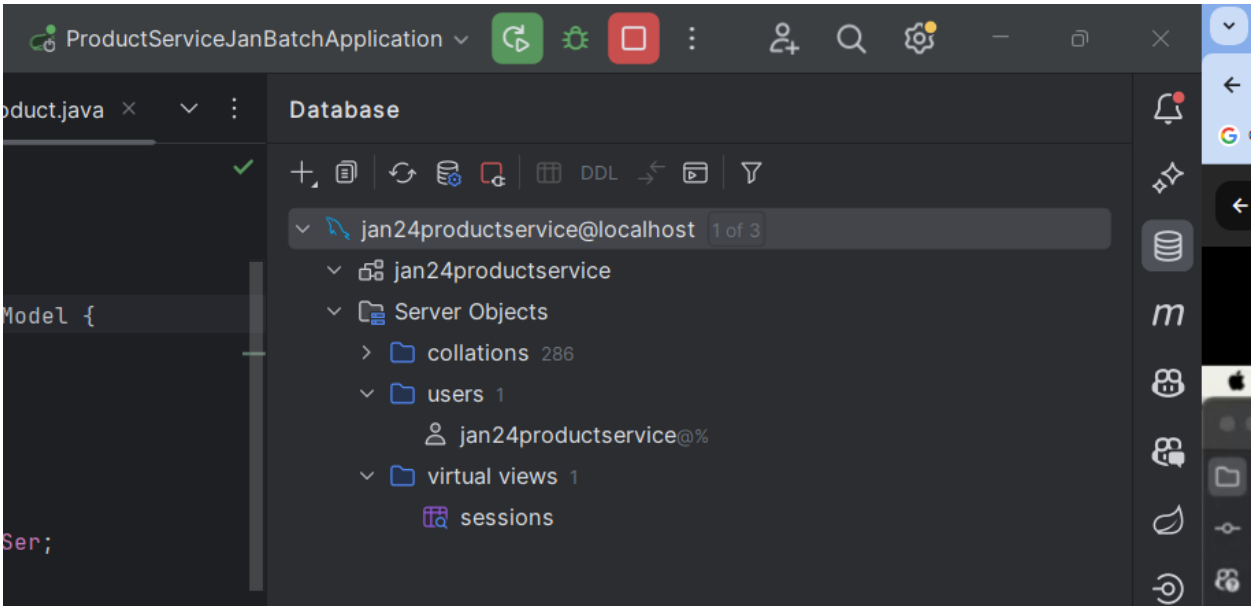


Add user and Database…





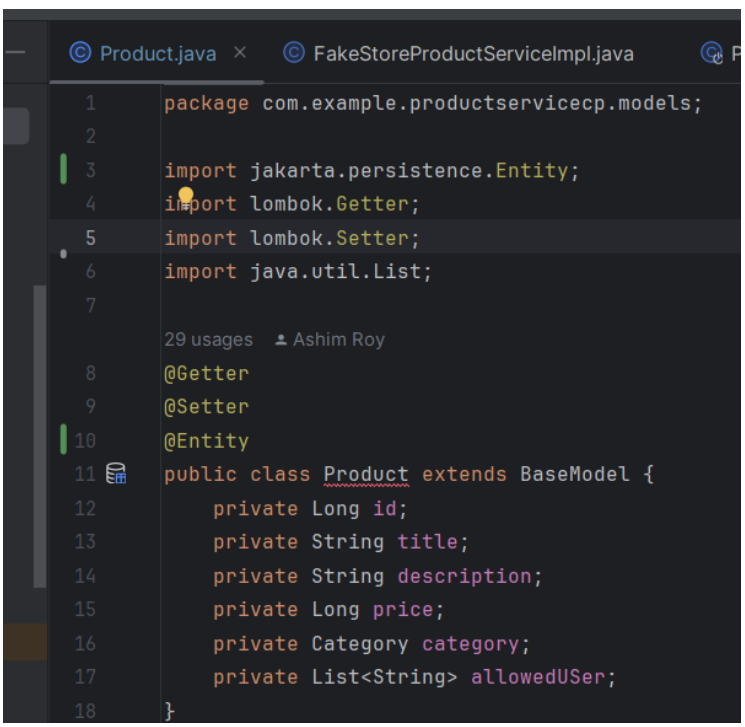
It downloads mysql connector.



In this project we will create for models… product and category. We want to convert the models to Tables.. I should not write the schema. I let JPA to do it…

How?

We add @Entity.. import jakarta.persistence.Entity;



```
package com.example.productservicecp.models;

import jakarta.persistence.Entity;
import lombok.Getter;
import lombok.Setter;
import java.util.List;

29 usages  Ashim Roy
@Getter
@Setter
@Entity
public class Product extends BaseModel {
    private Long id;
    private String title;
    private String description;
    private Long price;
    private Category category;
    private List<String> allowedUSer;
}
```

Persistence: whatever data you are persisting in DB.

```java
package com.example.productservicecp.models;

import jakarta.persistence.Entity;
import lombok.Getter;
import lombok.Setter;

4 usages  Ashim Roy
@Getter
@Setter
@Entity
public class Category extends BaseModel {
    private String name;
}
```

Same for category.

There is an error for product and category.. which says.. doesn't have primary key…

```java
@Entity
public class Category extends BaseModel {
    private Stri
}
```

Persistent entity 'Category' should have primary key

Add Id attribute Alt+Shift+Enter More actions... Alt+Enter

com.example.productservicecp.models

```java
@Entity
public class Category
extends BaseModel
```

ProductServiceCapstone

Id will be the PK.

```
Product.java    Category.java    BaseModel.java ×    FakeSto

package com.example.productservicecp.models;

import ...

2 usages   2 inheritors   Ashim Roy
@Getter
@Setter
public class BaseModel {
    private Long id;
}
```

We don’t want a table to be created for basemodel ..

Id should be part of product table… so we mark base as mapped superclass.

```
Product.java    Category.java    BaseModel.java ×    FakeStoreProduct

import jakarta.persistence.Id;
import jakarta.persistence.MappedSuperclass;
import lombok.Getter;
import lombok.Setter;

2 usages   2 inheritors   Ashim Roy *
@Getter
@Setter
@MappedSuperclass
public class BaseModel {
    @Id
    private Long id;
}
```

In product there is another error:

```
import jakarta.persistence.Entity;
import lombok.Getter;
import lombok.Setter;

import java.util.List;

30 usages  apple *
@Getter
@Setter
@Entity
public class Product extends BaseModel {
    private Long id;
    private String title;
    private String desc;
    private Long price;
    private Category category;
    private List
}
```

Entity attribute is not marked with association annotation

Make category Many to One association More actions...

com.example.productservicejanbatch24.models

Entity attribute is not marked with association annoatation. Product and category both are tables. Whats the relation between product and category???

One product can have 1 category

One category can have many products..

So its M: 1 relation..

Read it like:

Many product has one category… hence in product class we have manytoOne just above category…

```java
@Getter
@Setter
@Entity
public class Product extends BaseModel {
    private Long id;
    private String title;
    private String description;
    private Long price;
    @ManyToOne
    private Category category;
    private List<String> allowedUSer;
}

/*
the relation between product and category???
One product can have 1 category
One category can have many products.
 */
```

Product many one category…

//private List<String> allowedUSer; // we cant have a list here, remove this.

```java
import jakarta.persistence.ManyToOne;
import lombok.Getter;
import lombok.Setter;
import java.util.List;

29 usages  Ashim Roy *
@Getter
@Setter
@Entity
public class Product extends BaseModel {
    private String title;
    private String description;
    private Long price;
    @ManyToOne
    private Category category;
    //private List<String> allowedUSer; // we cant have a list here, remove this.
}
```

Now run it.. as we had connected to DB we get below:

```
Run   ProductServiceJanBatchApplication
Console   Actuator
  :: Spring Boot ::                (v3.2.3)

2024-03-17T22:21:59.666+05:30  INFO 17096 --- [  restartedMain] c.e.p.ProductServiceJanBatchApplication  : Starting ProductServiceJanBatchApplication using Java 17.0.1 with PID
2024-03-17T22:21:59.669+05:30  INFO 17096 --- [  restartedMain] c.e.p.ProductServiceJanBatchApplication  : No active profile set, falling back to 1 default profile: "default"
2024-03-17T22:21:59.712+05:30  INFO 17096 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-propertie
2024-03-17T22:21:59.712+05:30  INFO 17096 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.leve
2024-03-17T22:22:00.489+05:30  INFO 17096 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2024-03-17T22:22:00.507+05:30  INFO 17096 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 11 ms. Found 0 JPA reposi
2024-03-17T22:22:00.994+05:30  INFO 17096 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-03-17T22:22:01.005+05:30  INFO 17096 --- [  restartedMain] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-03-17T22:22:01.005+05:30  INFO 17096 --- [  restartedMain] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-03-17T22:22:01.052+05:30  INFO 17096 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-03-17T22:22:01.052+05:30  INFO 17096 --- [  restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1340 ms
2024-03-17T22:22:01.181+05:30  INFO 17096 --- [  restartedMain] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2024-03-17T22:22:01.257+05:30  INFO 17096 --- [  restartedMain] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.4.4.Final
```

There is an error:

```
Hibernate: alter table product add constraint FK1mtsbur82frn64de7balymq9s foreign key (category_id) references category (id)
2024-02-12T21:58:52.691+05:30  WARN 7311 --- [  restartedMain] o.h.t.s.i.ExceptionHandlerLoggedImpl     : GenerationTarget encountered exception

org.hibernate.tool.schema.spi.CommandAcceptanceException Create breakpoint : Error executing DDL "alter table product add constraint FK1mtsbur82frn64
    at java.base/java.util.HashMap.forEach(HashMap.java:1429) ~[na:na] <6 internal lines>
    at org.springframework.orm.jpa.vendor.SpringHibernateJpaPersistenceProvider.createContainerEntityManagerFactory(SpringHibernateJpaPersistenc
    at org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean.createNativeEntityManagerFactory(LocalContainerEntityManagerFactoryBea
    at org.springframework.orm.jpa.AbstractEntityManagerFactoryBean.buildNativeEntityManagerFactory(AbstractEntityManagerFactoryBean.java:409)
    at org.springframework.orm.jpa.AbstractEntityManagerFactoryBean.afterPropertiesSet(AbstractEntityManagerFactoryBean.java:396) ~[spring-orm-6
```

Sometimes from reading the error you wont figure out..

```
ngApplication.java:1343) ~[spring-boot-3.2.2.jar:3.2.2]
JanBatch24Application.main(ProductServiceJanBatch24Application.java:10) ~[classes/:na] <2 internal lines>
uncher.run(RestartLauncher.java:50) ~[spring-boot-devtools-3.2.2.jar:3.2.2]
: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'desc va
ption(SQLError.java:121) ~[mysql-connector-j-8.3.0.jar:8.3.0]
ranslateException(SQLExceptionsMapping.java:122) ~[mysql-connector-j-8.3.0.jar:8.3.0]
```

Desc is a keyword in mysql.. he added description as desc.. so error.. desc is a reserved keyword for mysql..

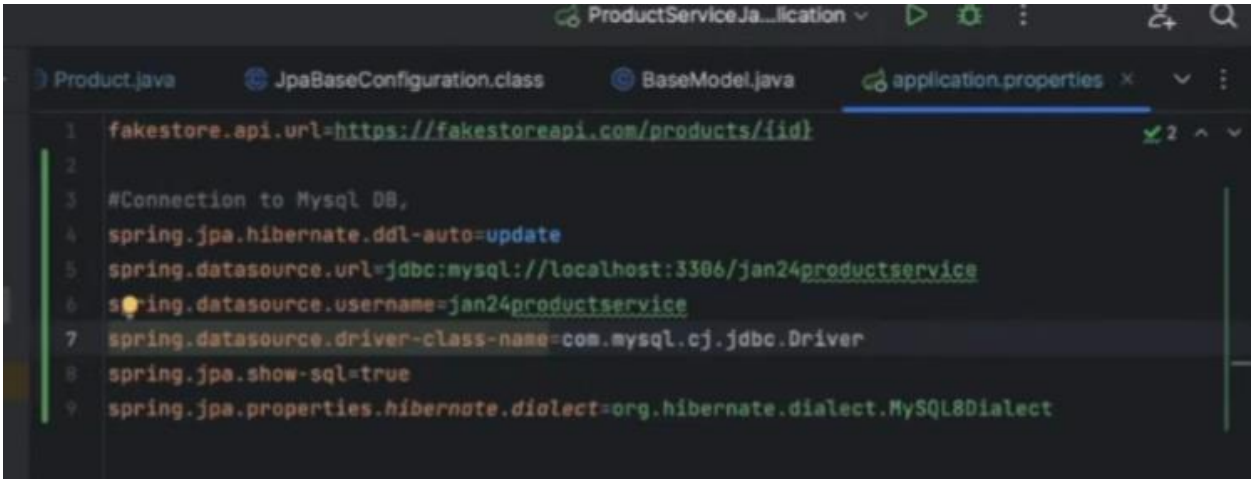
```
@Entity
public class Product extends BaseModel {
    private String title;
    private String desc;
    private Long price;
    @ManyToOne
    private Category category;

    /*
    1 Product - 1 Category
    1 Category - M Product
```

But im good:

```
public class Product extends BaseModel {
    private String title;
    private String description;
    private Long price;
    @ManyToOne
    private Category category;
    //private List<String> allowedUSer; // we cant have a list here, remove
this.
}
```

as I run I get few errors:

2024-03-17T22:32:27.698+05:30 ERROR 7828 --- [restartedMain] j.LocalContainerEntityManagerFactoryBean : Failed to initialize JPA EntityManagerFactory: Unable to create requested service [org.hibernate.engine.jdbc.env.spi.JdbcEnvironment] due to: Unable to resolve name [org.hibernate.dialect.MySQL5Dialect] as strategy [org.hibernate.dialect.Dialect]



Above is Nikhil file...

I have to make change to 8..

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

```
: HHH90000025: MySQL8Dialect does not need to be specified explicitly using 'hibernate.dialect' (remove the property setting and it will be selected by default)
: HHH90000026: MySQL8Dialect has been deprecated; use org.hibernate.dialect.MySQLDialect instead
: HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
: HikariPool-1 - Starting...
: HikariPool-1 - Exception during pool initialization.
```

The message you've received indicates that in your Hibernate configuration, you have a property **`hibernate.dialect`** set to **`MySQL8Dialect`**. However, Hibernate can auto-detect the dialect for MySQL 8, so you don't need to explicitly specify it.

You can remove the **`hibernate.dialect`** property from your configuration, and Hibernate will automatically use the appropriate dialect for MySQL 8 without needing it to be explicitly specified. This is because Hibernate has built-in support for various database dialects, including MySQL 8. Removing this property can simplify your Hibernate configuration.

Failed to initialize JPA EntityManagerFactory: Unable to create requested service [org.hibernate.engine.jdbc.env.spi.JdbcEnvironment] due to: Unable to determine Dialect without JDBC metadata (please set 'jakarta.persistence.jdbc.url' for common cases or 'hibernate.dialect' when a custom Dialect implementation must be provided)

```
spring.datasource.url=jdbc:mysql://localhost:3306/jan24productservice
```

I have corrected the URL now run it and its error free:

Product.java | Category.java | BaseModel.java | application.properties | FakeStor

```
fakestore.api.url = https://fakestoreapi.com/products/{id}

# connecting to mysql database
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/jan24productservice
spring.datasource.username=jan24productservice
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.show-sql= true
#spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

ProductServiceJanBatchApplication

Console | Actuator

```
2024-03-17T22:43:34.556+05:30  INFO 17324 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-propertie
2024-03-17T22:43:34.556+05:30  INFO 17324 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.leve
2024-03-17T22:43:35.067+05:30  INFO 17324 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2024-03-17T22:43:35.083+05:30  INFO 17324 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 11 ms. Found 0 JPA reposi
2024-03-17T22:43:35.498+05:30  INFO 17324 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-03-17T22:43:35.506+05:30  INFO 17324 --- [  restartedMain] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-03-17T22:43:35.506+05:30  INFO 17324 --- [  restartedMain] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-03-17T22:43:35.545+05:30  INFO 17324 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-03-17T22:43:35.546+05:30  INFO 17324 --- [  restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 989 ms
2024-03-17T22:43:35.646+05:30  INFO 17324 --- [  restartedMain] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2024-03-17T22:43:35.682+05:30  INFO 17324 --- [  restartedMain] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.4.4.Final
2024-03-17T22:43:35.706+05:30  INFO 17324 --- [  restartedMain] o.h.c.internal.RegionFactoryInitiator    : HHH000026: Second-level cache disabled
2024-03-17T22:43:35.888+05:30  INFO 17324 --- [  restartedMain] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
2024-03-17T22:43:35.909+05:30  INFO 17324 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2024-03-17T22:43:36.280+05:30  INFO 17324 --- [  restartedMain] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection com.mysql.cj.jdbc.ConnectionImpl@e0e4
2024-03-17T22:43:36.282+05:30  INFO 17324 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2024-03-17T22:43:36.990+05:30  INFO 17324 --- [  restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.
Hibernate: create table category (id bigint not null, name varchar(255), primary key (id)) engine=InnoDB
Hibernate: create table product (id bigint not null, description varchar(255), price bigint, title varchar(255), category_id bigint, primary key (id)) engine=InnoDB
Hibernate: alter table product add constraint FK1mtsbur82frn64de7balymq9s foreign key (category_id) references category (id)
2024-03-17T22:43:37.188+05:30  INFO 17324 --- [  restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2024-03-17T22:43:37.298+05:30  WARN 17324 --- [  restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database qu
```

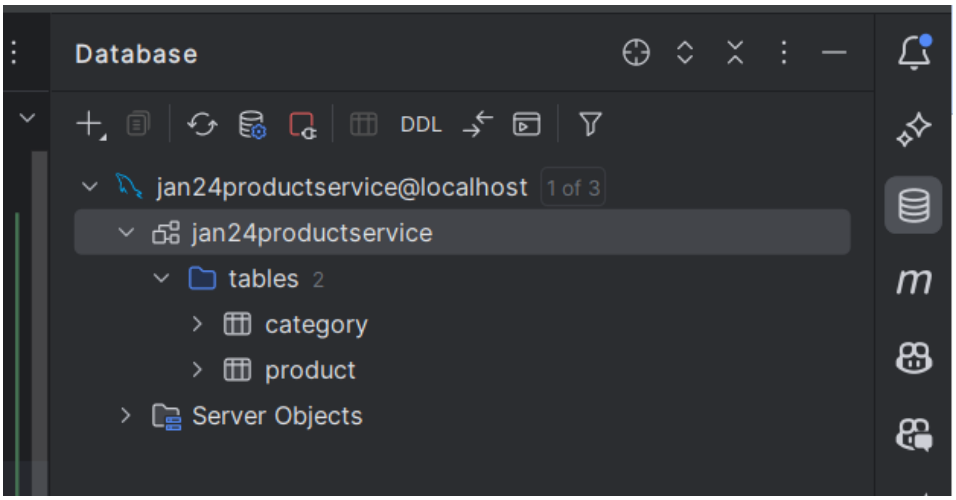
Now in log:

```
2024-03-17T22:43:36.282+05:30  INFO 17324 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2024-03-17T22:43:36.990+05:30  INFO 17324 --- [  restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available (set 'hibernate.transac
Hibernate: create table category (id bigint not null, name varchar(255), primary key (id)) engine=InnoDB
Hibernate: create table product (id bigint not null, description varchar(255), price bigint, title varchar(255), category_id bigint, primary key (id)) engine=InnoDB
Hibernate: alter table product add constraint FK1mtsbur82frn64de7balymq9s foreign key (category_id) references category (id)
2024-03-17T22:43:37.188+05:30  INFO 17324 --- [  restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'd
```

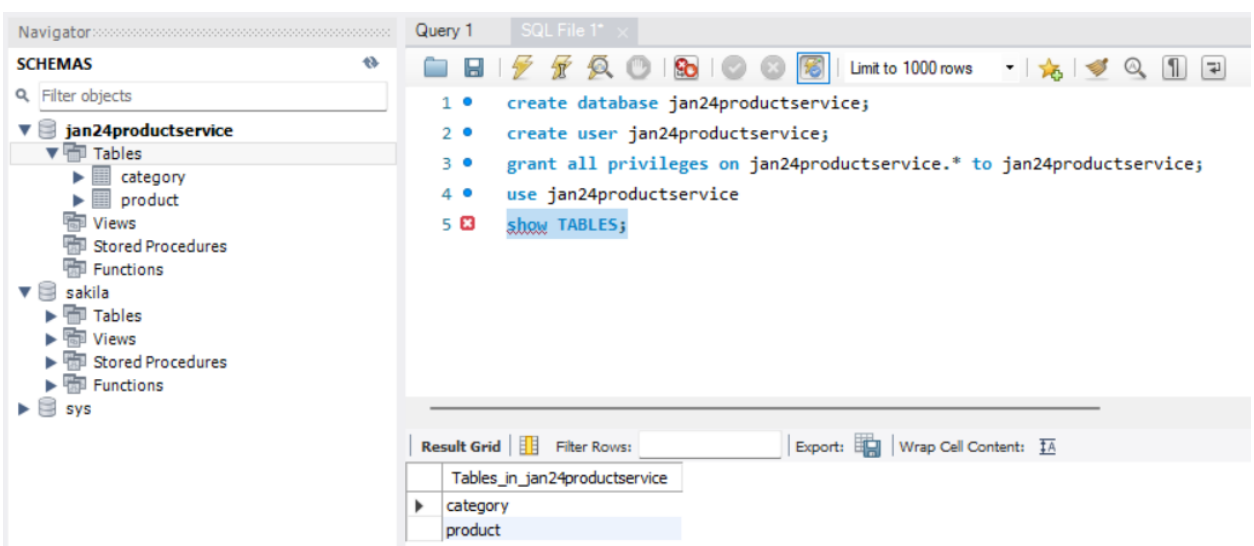
Hibernate: create table category (id bigint not null, name varchar(255), primary key (id)) engine=InnoDB

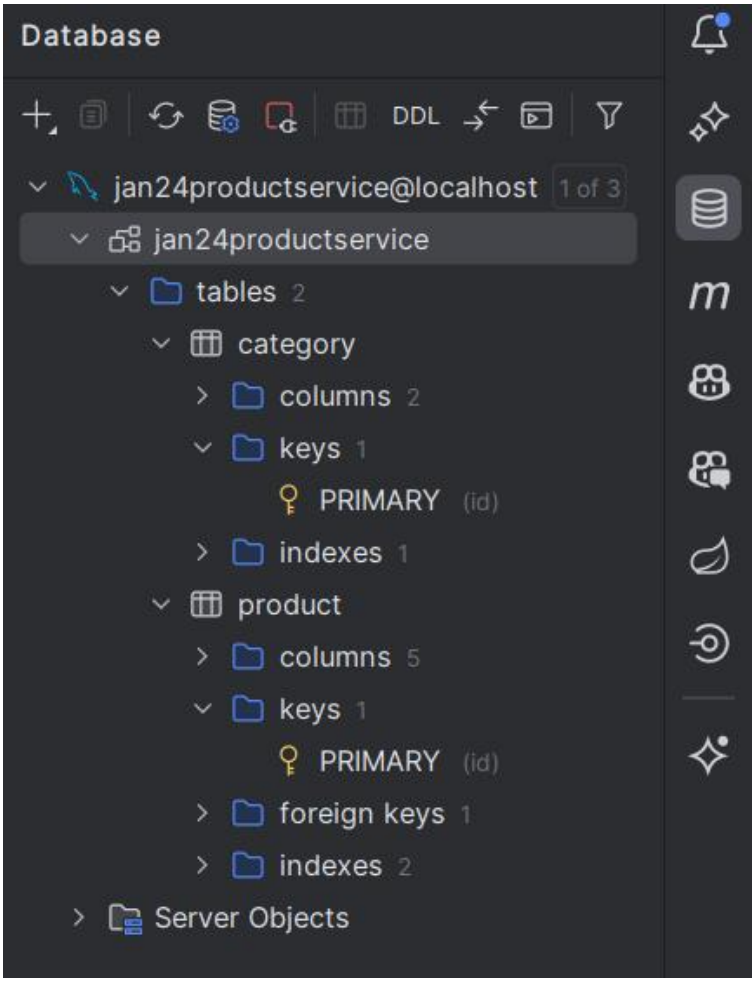
Hibernate: create table product (id bigint not null, description varchar(255), price bigint, title varchar(255), category_id bigint, primary key (id)) engine=InnoDB

Hibernate: alter table product add constraint FK1mtsbur82frn64de7balymq9s foreign key (category_id) references category (id)

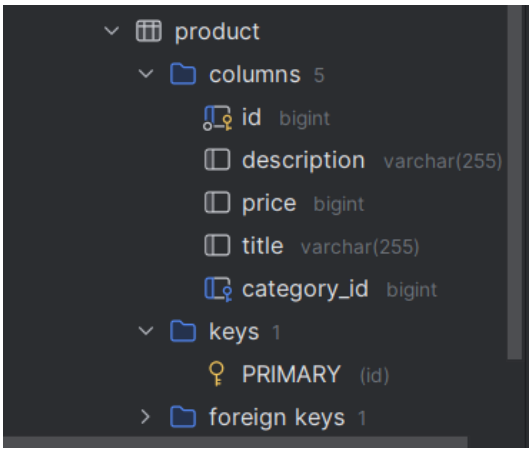


So 2 tables are created…

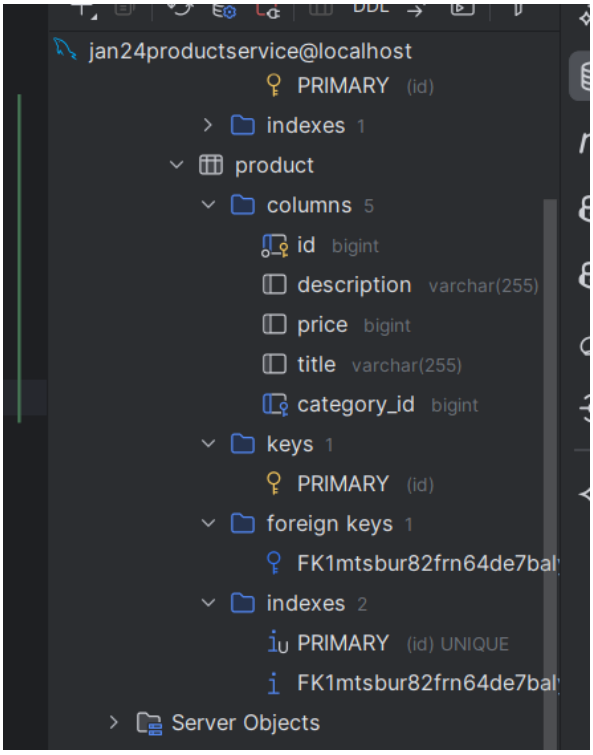




Tables are created with columns and foreign keys..



Magic of ORM. You have asked, I no worry how connected to DB or creating DB..


jan24productservice@localhost
PRIMARY (id)
indexes 1
product
columns 5
id bigint
description varchar(255)
price bigint
title varchar(255)
category_id bigint
keys 1
PRIMARY (id)
foreign keys 1
FK1mtsbur82frn64de7bal
indexes 2
PRIMARY (id) UNIQUE
FK1mtsbur82frn64de7bal
Server Objects

BREAK…….

UUID: universally Unique Identifier...

read this: https://en.wikipedia.org/wiki/Universally_unique_identifier

https://www.cockroachlabs.com/blog/what-is-a-uuid/

When working with a database, it's common practice to use some kind of `id` field to provide a unique identifier for each row in a table.

Imagine, for example, a `customers` table. We wouldn't want to use fields such as `name` or `address` as unique identifiers because it's possible more than one customer could have the same name, or share the same address, or in some cases even both!

Instead, it's a good idea to assign each row some kind of *truly* unique identifier. One option we have is to use a UUID.

What is a UUID?

A UUID – that's short for Universally Unique IDentifier, by the way – **is a 36-character alphanumeric** string that can be used to identify information. They are often used, for example, to identify rows of data within a database table, with each row assigned a specific UUID.

Here is one example of a UUID: `acde070d-8c4c-4f0d-9d8a-162843c10333`

UUIDs are widely used in part because they are highly likely to be unique *globally*, meaning that not only is our row's UUID unique in our database table, it's probably the only row with that UUID in any system *anywhere*.

(Technically, it's not *impossible* that the same UUID we generate could be used somewhere else, but with 340,282,366,920,938,463,463,374,607,431,768,211,456 different possible UUIDs out there, the chances are *very* slim).

What are UUIDs used for?

To answer this question, let's imagine we're operating an ecommerce bookshop. As orders come in, we want to assign them an id number and store them in our `orders` table using that number.

We could set up sequential IDs such that the first order to come in is 1, the second is 2, and so on, like so:

id	item	buyer	price
1	The Years of Rice and Salt	Sue	$14
2	A Darkling Sea	Al	$20
3	Too Like the Lightning	Mei	$25

And this approach might work well, at least for a while, if our scale is small. However, it has some major downsides:

First, it can easily create confusion when we're doing things like joining tables or importing new data, because the id values above aren't unique. This can create problems even internally if we use the same ID system for multiple tables, and it really gets messy when we start working with any kind of outside data.

Imagine, for example, that our little bookshop grows, and we acquire another online bookshop. When we go to integrate our order tables, we find that they've used the same system. Now we've got two order 1s, two order 2s, etc., and to resolve the issue, we'll have to update *every single ID* in at least one of the two databases we're integrating. Even in a best case scenario, that's going to be a tremendous hassle.

Second, the sequential approach often doesn't work well in distributed systems, because using sequential IDs means that INSERT commands must be executed one by one. This restriction can cause major performance issues, as your database nodes have to wait around as one node at a time writes data, rather than having all nodes be able to write simultaneously. Even if your application requires strict ID ordering, using a feature such as CockroachDB's Change Data Capture may allow you to meet those requirements while still using UUIDs and not taking the performance hit that comes with sequentially-ordered IDs.

Other traditional approaches to unique IDs, such as generating random IDs with `SERIAL`, can also lead to hotspots in distributed systems, because values generated around the same time have will often be similar and thus may be located close to each other in the table's storage. In CockroachDB, for example, this can lead to hotspots where one node gets overworked because it's handling most or all of the writes while other nodes sit idle.

UUIDs solve all of these problems because:

- They're globally unique, so the chances of encountering a duplicate ID even in external data are very, very small.
- They can be generated without the need to check against a central node, so in a distributed system, each node can generate UUIDs autonomously without fear of duplication or consistency issues.

Reason #1 alone is a good argument for using UUIDs in almost any database system. As a business that aspires to operate at scale, reason #2 is also very relevant to our bookshop, because distributed databases offer the best scalability and resilience.

Disadvantages of UUIDs

The only significant disadvantage of UUIDs is that they take up 128 bits in memory (and often a bit more when we include metadata). If minimizing storage space is absolutely mission-critical, clearly storing a sequential ID (which will probably range somewhere between 1-10 numeric characters) is going to be more efficient than storing a 36-character alphanumeric.

However, in most cases the disadvantages of using something like a sequential identifier significantly outweigh the minimal increase in storage costs that comes from using UUIDs.

UUIDs are extremely popular and widely used for a variety of different identification purposes. We've focused on database examples in this article because we make a pretty awesome database, but UUIDs are also used in analytics systems, web and mobile applications, etc.

Examples of UUIDs

There are several different types of UUIDs:

Version 1 and version 2. Sometimes called time-based UUIDs, these IDs are generated using a combination of datetime values (reflecting the time the UUID is being generated), a random value, and a part of the MAC address of the device generating the UUID.

Here's how it breaks down visually:

Generating UUIDs in this way makes having identical UUIDs almost impossible – they would have to be generated by the same device at the exact same time *and* have generated the exact same random 16-bit sequence.

Because they contain a part of the generating device's MAC address, UUID v1 and UUID v2 IDs can be used to identify (for example) which database node generated the ID. This is generally not a problem, and in distributed systems, it can be an advantage.

(The difference between v1 and v2 UUIDs is that UUID v2s also contain a segment of a local domain number. For a number of reasons, this makes them less optimal for most applications, so UUID v2s are not widely used.)

Version 3 and version 5. These two versions of UUIDs are generated by hashing a namespace identifier and name. They're similar to time-based UUIDs in that they are generated using existing data rather than being entirely random, but rather than using datetime data and the device MAC address, they use namespace data and name data.

The namespace data is itself a UUID, and the name data could really be any arbitrary string, although in practice it typically relates to how the UUID will be used – it might be an account name, for example, or a product ID. But whatever the two values used are, they're hashed to generate a 36-character alphanumeric string that is the final UUID.

UUIDs versions 3 and 5 differ primarily in that they use different hashing algorithms. UUID v3 uses MD5, and UUID v5 uses SHA-1.

https://www.cockroachlabs.com/blog/what-is-a-uuid/

UUID

↳ Universally Unique Identifier

Scaler

Student → Table DB

↳ Primary key

Email → PK

when you register for scaler. Student have a table in DB. For this DB we need a Primary Key. Being armature or naïve what will be the primary key will you keep? Email as PK we can keep as it will be unique.

If I keep email as PK.. choosing that what will be disadvantage: space, can be as long as possible.

Storing a string takes time.. even indexing will take time and formatting also.

Email id can change. Might change. As it is a User Attribute. So it is not a good identifier..

Disadv

1) Space
2) Storage of string takes time
3) It might change as it is User Attribute

Integer | Long as PK
↳ Auto increment

1) Overflow
2) Distributed System

Then we can take Integer or Long something numerical as PK. Then I will have to auto increment that.. what will be an issue… there will be overflow.

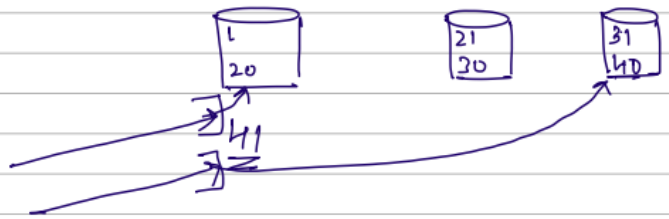
Another issue: In a distributed system, if I have 3 Databases…

[1….20] [21…..30] [31…..40]

If 2 request have come mountainously next digit I will take 41,, if they both at 41 it will be issue. Inconsistency issue.. UUID was not discussed in HLD…

If database is distributed across multiple machines then the primary key should be in sync.. across the machines… that is a overhead.. if that's a big overhead. We need to work in a sequential order. If its not a distributed then weasier to take care. In DS hard.

Then what I can do to do unique identifier. If the ID I create with a combo of unique eto me… what all can be nique: timestamp of server.. if server goes down new server comes up.. problem..



If DB is distributed across multiple machines then the PK should be in sync accross the machines.

Sequential Order

id = { Current time + mac id + ip address + User Id + Serial NO + - - - - }

I can do curr time+server id (mac id)+userID+ serial no.. using all these I can create a unique id. the more parameter I have the less time of Collison.

More parameter are there lesser the chance of collesion..

UUID [128 bit]

[0110] [1001] [0001] [1111] [1010]

Hexadecimal

Char => 4 bits $\frac{128}{4} = 32B$

Binary => 8 bits $\frac{128}{8} = 16B$

UUID = 128 bit number and denoted with Hexa decimal format... in a combination of 4.

0110, 1001, 0001, 1111, 1010.

Denoted in hexa decimal.

What is the size of character? 1 byte… int = 4 byte, binary ka size = 8 byte

To 128 bit of string

As binary: 128/8 = 16B

Binaries are cost effective… so this is how they are stored in DB.

<u>Lets see how you will create UUID and spring will create it:</u>

Basemodel as of now:

```
ackage com.example.productservicecp.models;

import jakarta.persistence.Id;
import jakarta.persistence.MappedSuperclass;
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
@MappedSuperclass
public class BaseModel {
    @Id
    private Long id;
}
```

change the long to UUID.. uuid provided by java.

```
import java.util.UUID;

@Getter
@Setter
@MappedSuperclass
public class BaseModel {
    @Id
    private UUID id;
}
```

we want to use UUID. We havnt specified what kind of generator we need…?

```
2 usages  2 inheritors  Ashim Roy *
@Getter
@Setter
@MappedSuperclass
public class BaseModel {
    @Id
    @GeneratedValue(strategy = Gen)
    private UUID id;
}
```

GenerationType.AUTO (jakarta.persistence) Gene
GenerationType jakarta.persistence
GenerationType.IDENTITY (jakarta.persistence) Gene
GenerationType.UUID (jakarta.persistence) Gene
GenerationType.SEQUENCE (jakarta.persistence) Gene
GenerationType.TABLE (jakarta.persistence) Gene
GeneratedValue jakarta.persistence
Generated lombok
Generated javax.annotation.processing

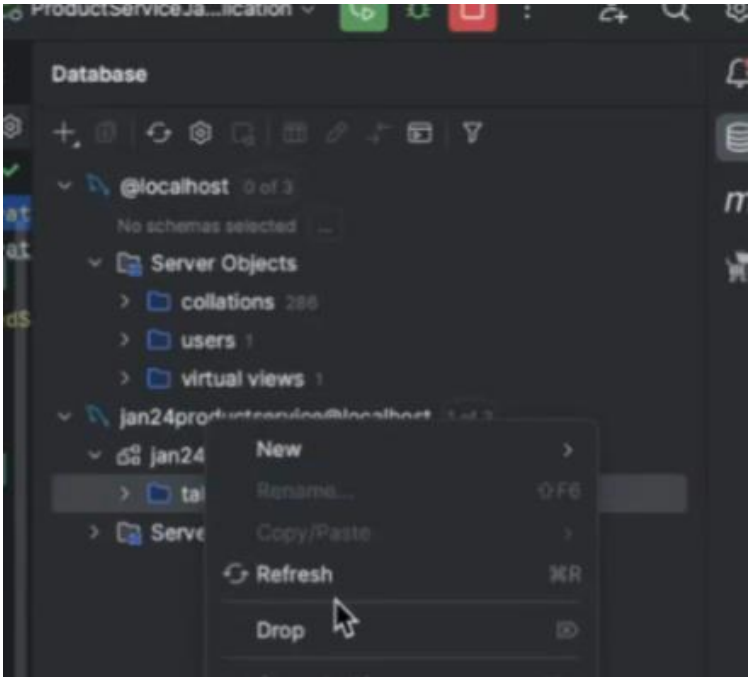
```
import java.util.UUID;

@Getter
@Setter
@MappedSuperclass
public class BaseModel {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;
}
```

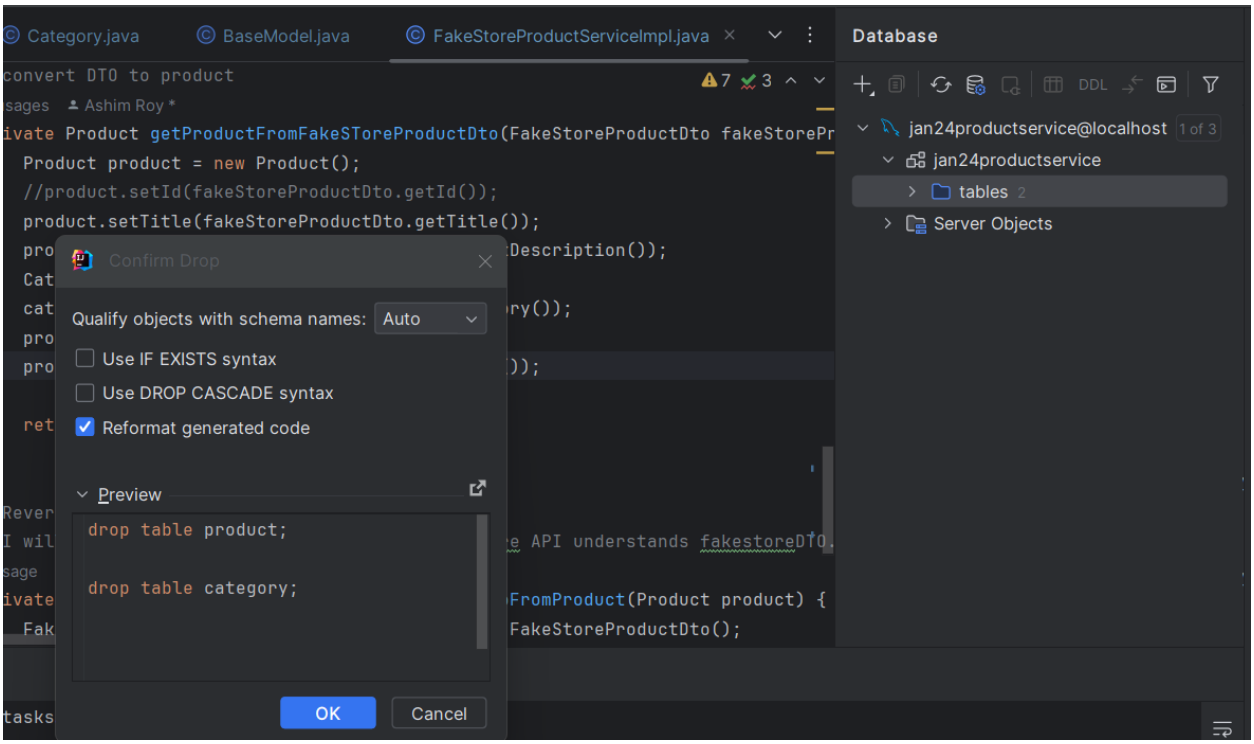
as we change it we get sdome error.. lets comment it for now.

```
4 usages  Ashim Roy
private Product getProductFromFakeSToreProductDto(FakeStoreProductDto fakeStoreProductDto) {
    Product product = new Product();
    product.setId(fakeStoreProductDto.getId());
    product.setTitle(fakeStoreProductDto.getTitle());
    product.setDescription(fakeStoreProductDto.getDescription());
    Category category = new Category();
    category.setName(fakeStoreProductDto.getCategory());
    product.setCategory(category);
    product.setPrice(fakeStoreProductDto.getPrice());
```

```
areer-Study\SCALER ALL\Scaler-Class notes\6 - Backend Capstone Project- LLD4\Project\ProductServiceCapstone
ncompatible types: java.lang.Long cannot be converted to java.util.UUID
```

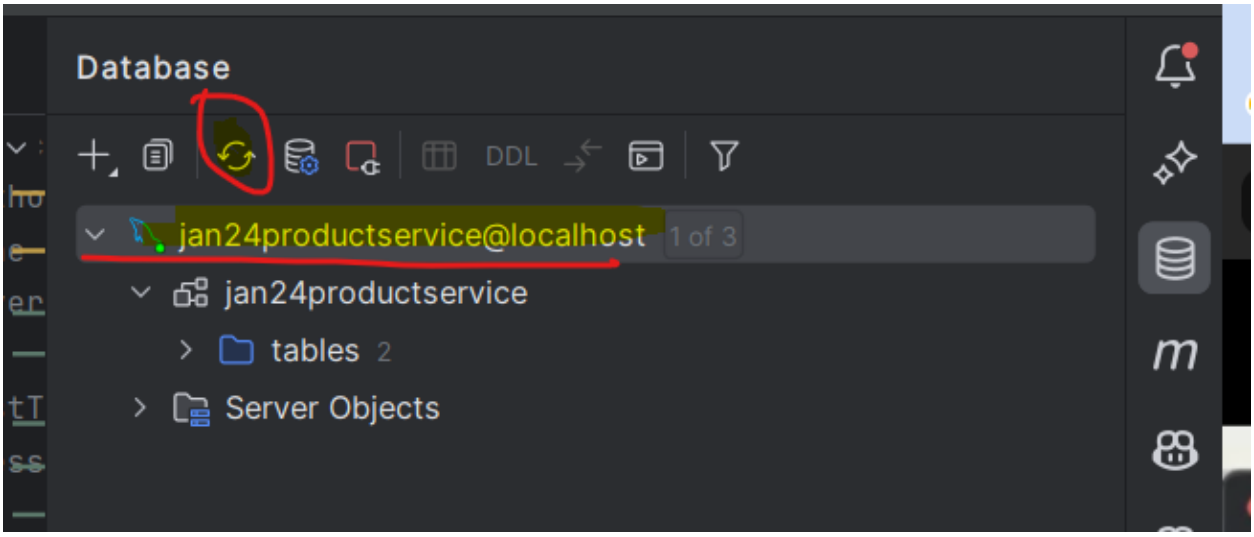
Lets drop the table and rerun..



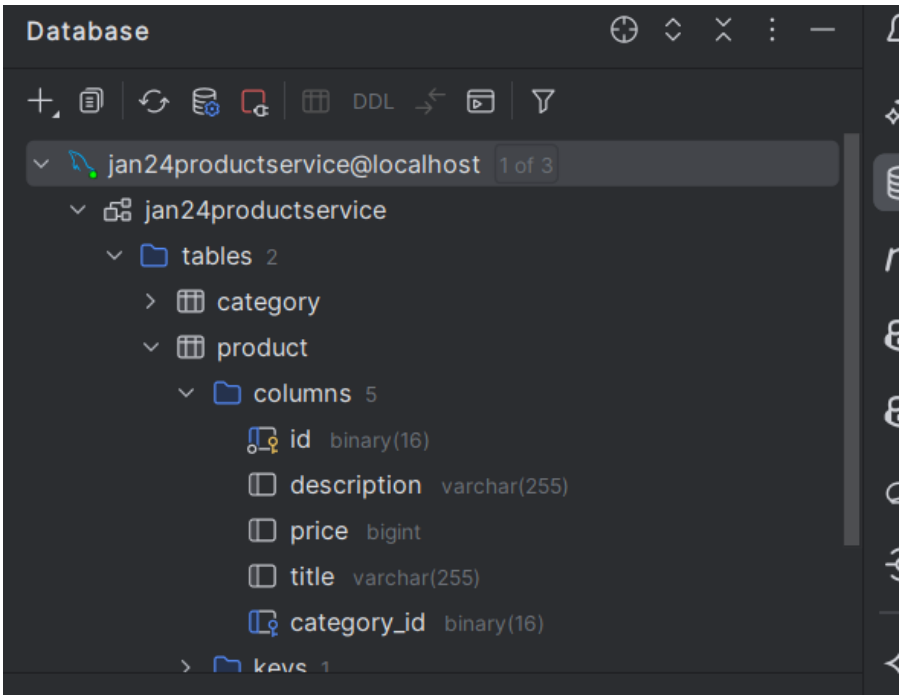


Lets rerun..

Now run and refresh the database server



Now the id column of product table is binary 16



do not we have to use strategy as UUID

```
public class BaseModel {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;
}
```

Stretegy is to generate the UUID.

Auto will see which kind of UID it should generate. By default it will have a UID 5 of strategy.. AUTO by default generates UID. But if your DB doesn't support it will generate diff type of UUID. Some MYsql support UID4, some support UID5. There are 1 to 7 version of UID. That's why we use Auto..

We will write a custom generator..

```
import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import org.hibernate.annotations.GenericGenerator;

import java.util.UUID;

@Getter
@Setter
@MappedSuperclass
public class BaseModel {
    @Id
    //@GeneratedValue(strategy = GenerationType.AUTO)
    // writing custom generator...
    @GeneratedValue(generator = "generator_name")
    @GenericGenerator(name = "generator_name", strategy = "uudi2")
    @Column(name = "id", columnDefinition = "BINARY(16)", updatable = false,
nullable = false)
    private UUID id;
}
```

own custom generator.. which kind of UUID. Kind of definition… etc..

```
2 usages  2 inheritors  Ashim Roy *
@Getter
@Setter
@MappedSuperclass
public class BaseModel {
    @Id
    //@GeneratedValue(strategy = GenerationType.AUTO)
    // writing custom generator...
    @GeneratedValue(generator = "generator_name")
    @GenericGenerator(name = "generator_name", strategy = "uudi2")
    @Column(name = "id", columnDefinition = "BINARY(16)", updatable = false, nullable = false)
    private UUID id;
}
```

We should not use Long.. UID makes more sense in uniqueness sense.. long is dur to inconsistency in Distributed system or when you make concurrent calls…

```java
import java.util.UUID;

2 usages  2 inheritors  Ashim Roy *
@Getter
@Setter
@MappedSuperclass
public class BaseModel {

    // writing custom generator...
    /*
    @GeneratedValue(generator = "generator_name")
    @GenericGenerator(name = "generator_name", strategy = "uudi2")
    @Column(name = "id", columnDefinition = "BINARY(16)", updatable = false, nullable = fa
    */

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;
}
```

but AUTO is giving us uniqueness as auto is giving us uniqueness we can make type as long

it will try to do unique long but long has own issue. It will try to auto increment..

lest look at generationType class:

```java
package jakarta.persistence;

public enum GenerationType {
    TABLE,
    SEQUENCE,
    IDENTITY,
    UUID,
    AUTO;

    private GenerationType() {
    }
}
```

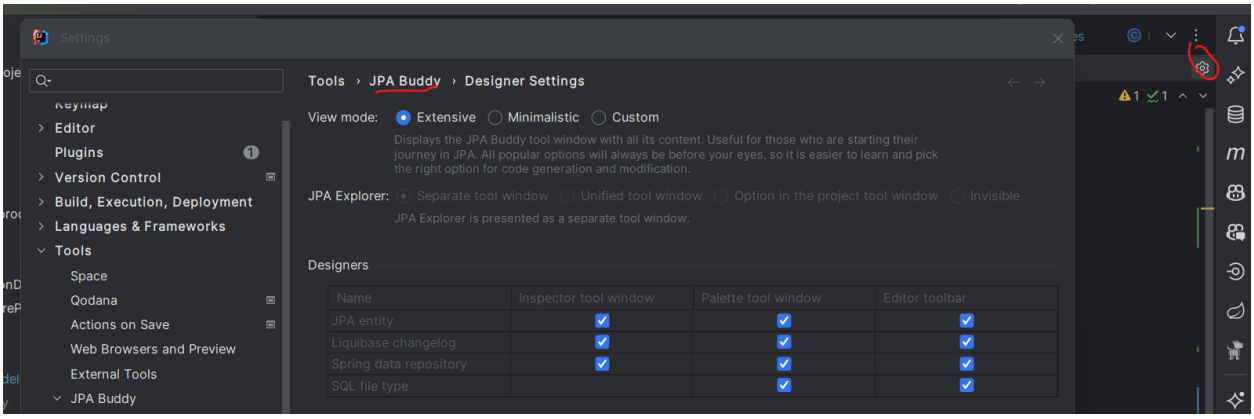
to read other generation logic: https://www.baeldung.com/hibernate-identifiers

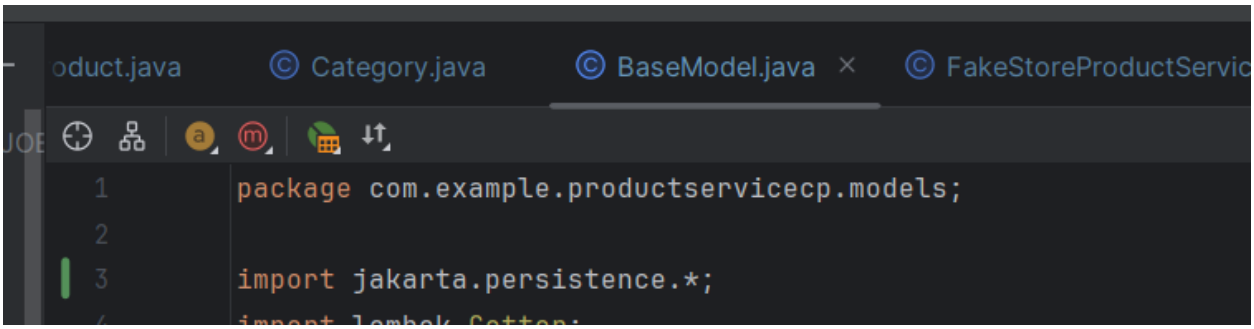
so in all distributed env uuid is used?

In consistence hashing… how CH work… what does it use for CH. what kind of UID it generates? gouied..

Add jpa buddy in intellij.. it gives these options.

Click on setting icon to see jpa buddy.. ka options.. helps in migration, schema verisononing.





Inheritance in database and association will come next class.

<u>Use gemini.google.com</u>