HIBERNATE

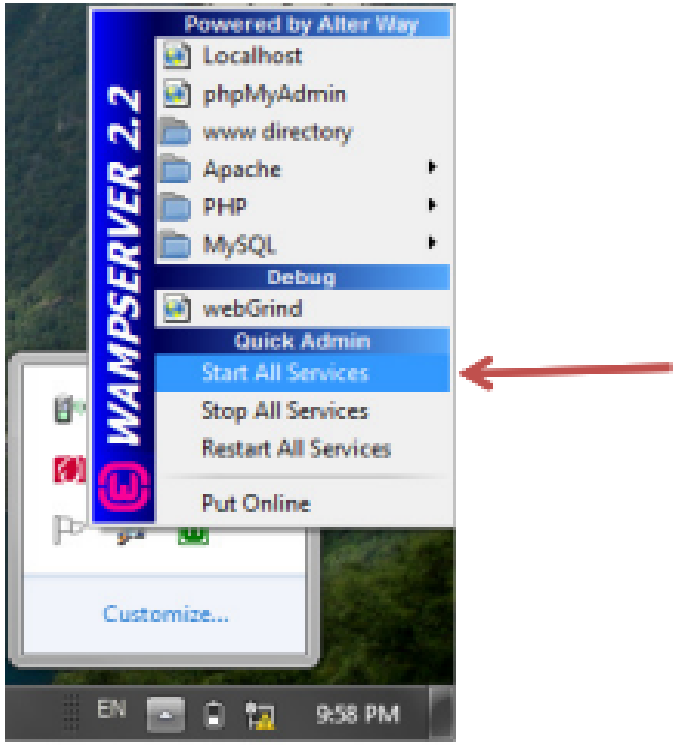
Step 1: Create Database and Table

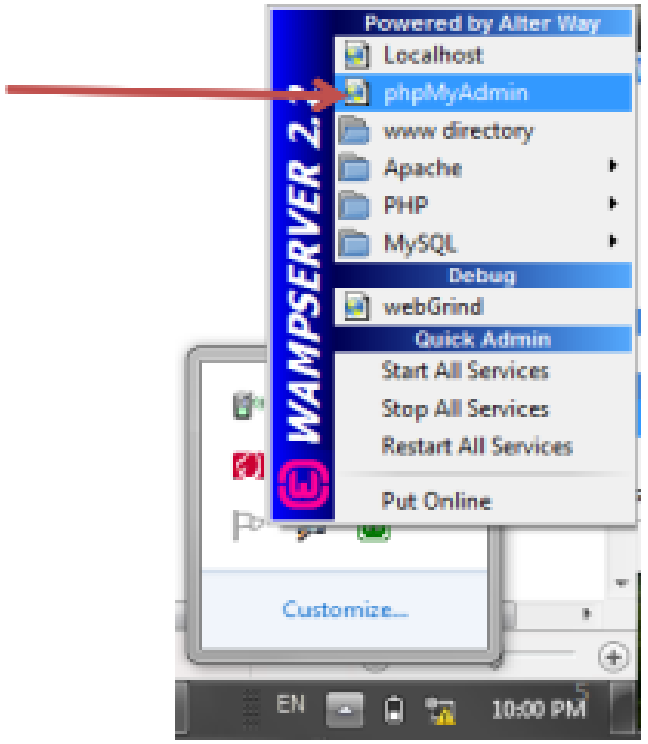
Installing MySQL

- Way 1
  - Install from MySQL website
  - Install GUI tool such as SQLWave to access the database
- Way 2
  - Download and install WAMP server
    - Apache, Mysql, PHP
    - Website: www.wampserver.com/en
  - GUI tool, phpMyAdmin is available as part of wamp server

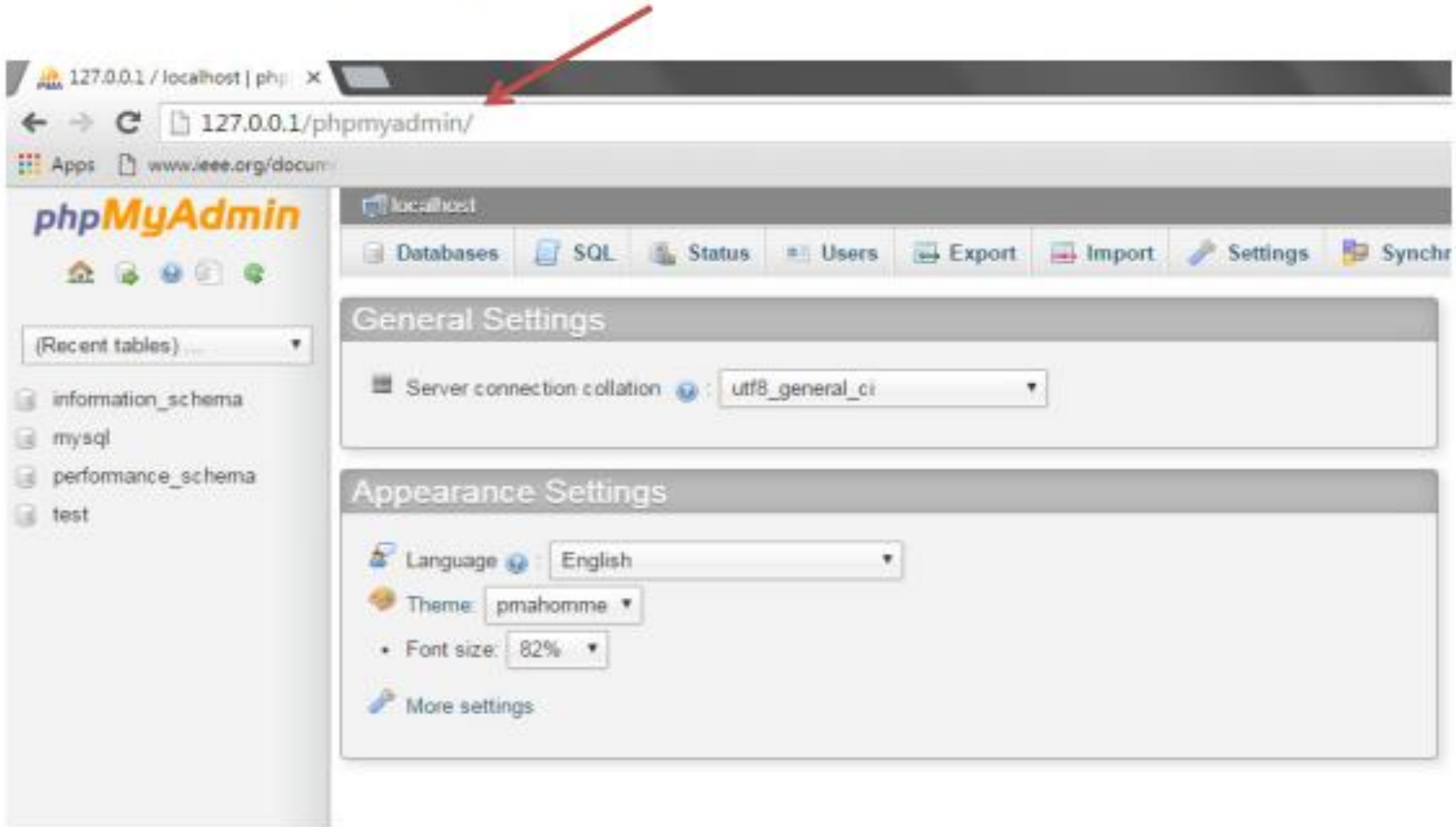
Starting WAMP server

- Click on the icon of WAMP server and start all services
- Then, open phpMyAdmin

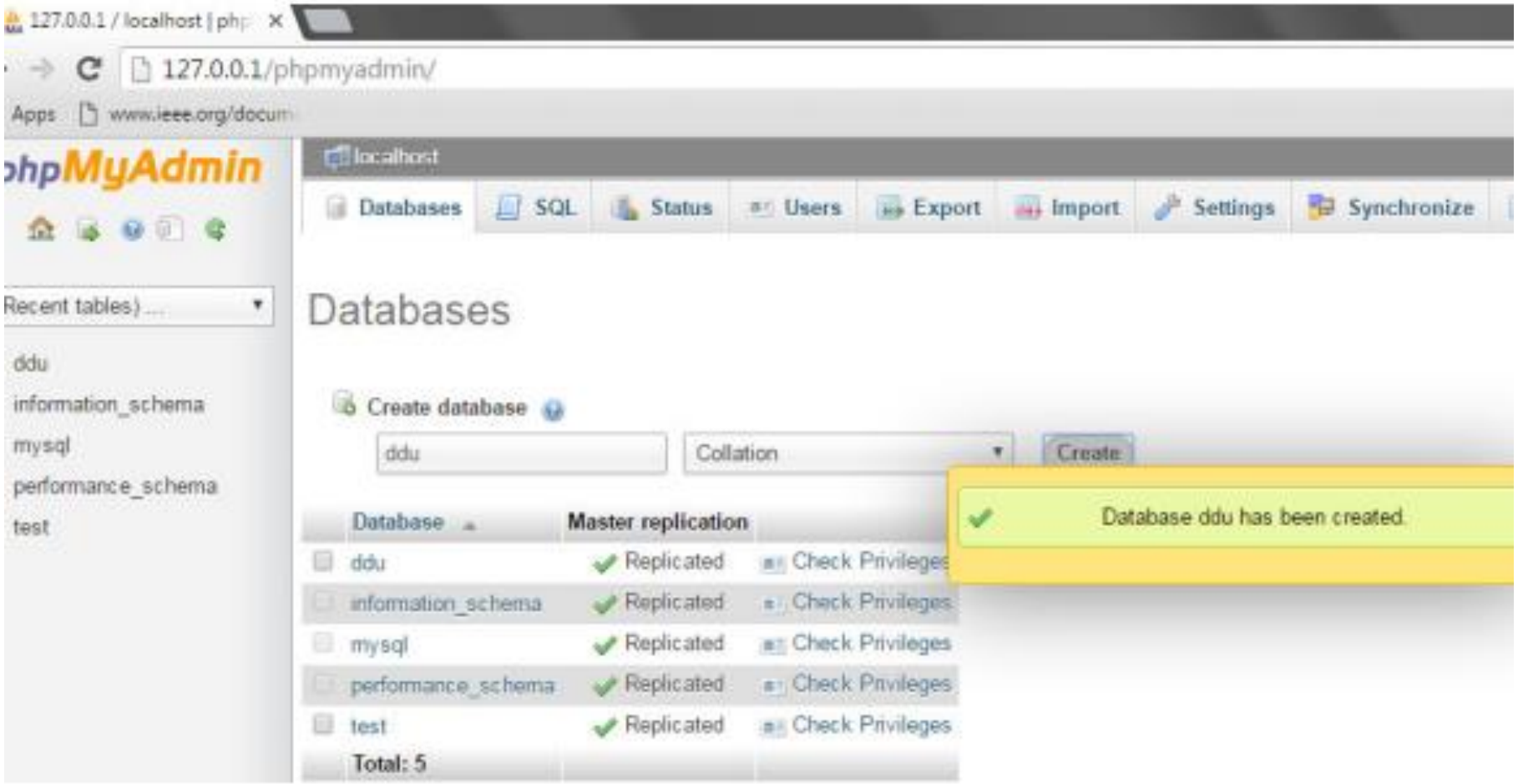




phpmyadmin web console

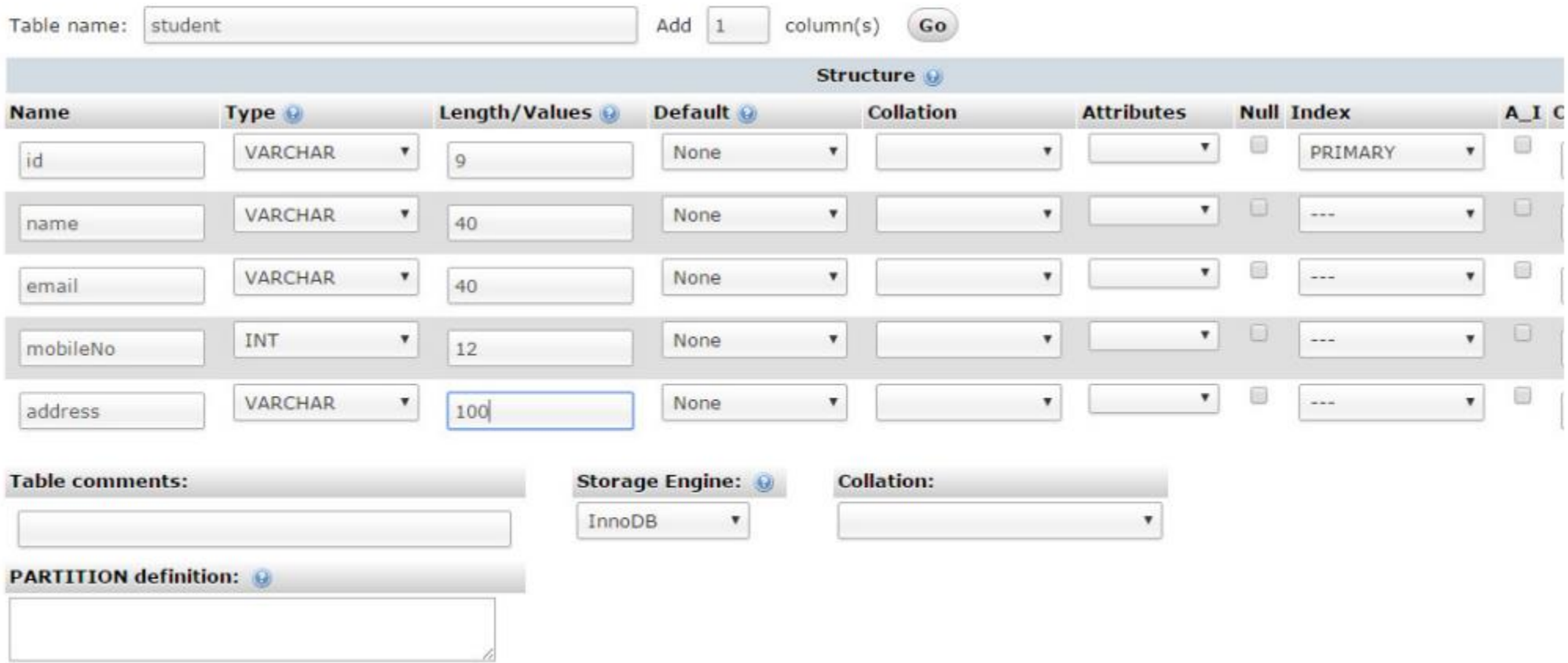


Create a new database



Create a database table

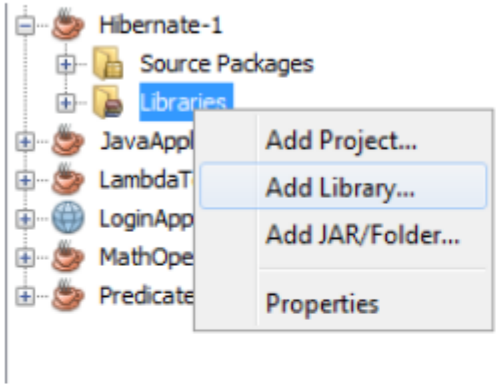
- Specify name of the table (student) and # columns (5)



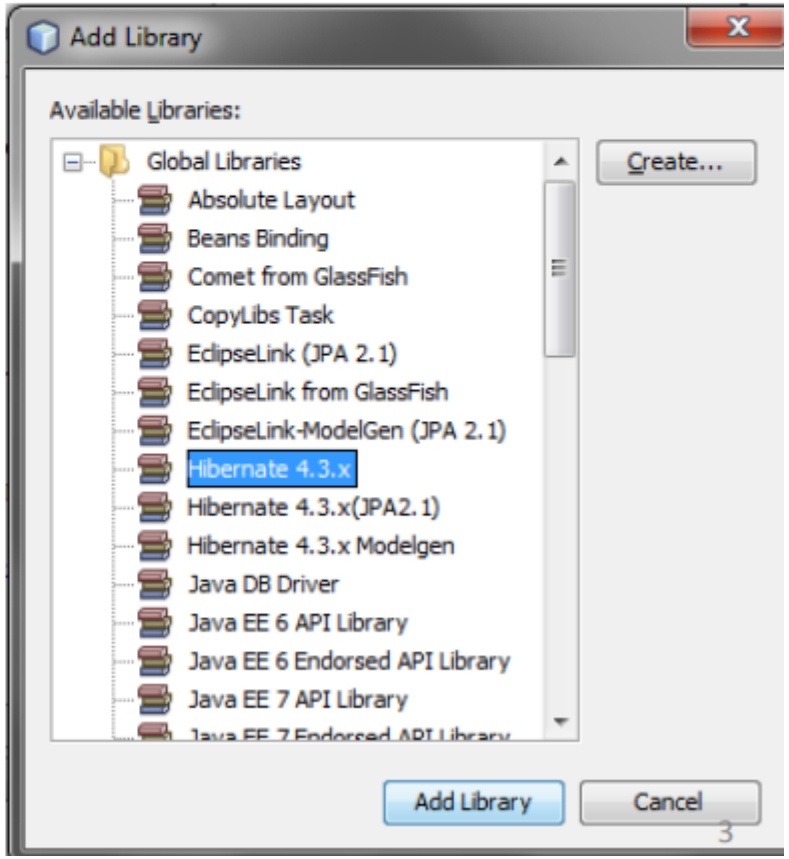
Step 2: SetUp Hibernate in IDE

- Download Hibernate jar file from www.hibernate.org/downloads
- Visit release bundles
- Select latest repository
- Download zip file
- Extract zip file in a folder
- The folder lib\required contains all necessary jar files.
  - Main jar file : hibernate-core*.jar
  - Other jar files: dependencies of core file
- The other folders: envers, jpa, and optional provide advanced features
- Create a new User library (Hibernate Library) and add it to the project path
- Also include jar files for the selected database system
  - E.g., for MySQL the connector file is mysql-connector-java*-bin.jar

- Add Library



- Select Hibernate



Step 3: Hibernate Based Application Creation

- There are two ways to create hibernate based application
  - Create mapping (of Java class to DB table) using a mapping file.
  - Create the mapping using annotations.

Creating a hibernate based application using Mapping file

- Three major steps
  - Create configuration files
    - Hibernate configuration
      - Database information, and other properties
      - hibernate.cfg.xml
    - Mapping configuration
      - <Class_name>.hbm.xml
  - Create a model object
    - Define data:
  - Define behaviors/operations on data:
    - Methods to deal (save, delete, etc) with the database

Step 4: Create Hibernate Configuration File

- Name of the configuration file
  - hibernate.cfg.xml
- Information in the configuration file
  - Database connection settings
  - JDBC connection pool
  - SQL dialect
    - Hibernate can generate optimized query for the specified dialect (E.g., MySQLDialect)
  - Second level cache provider
  - Echo all queries to stdout
  - Property: hbm2ddl.auto
  - Mention all model classes (with package path)

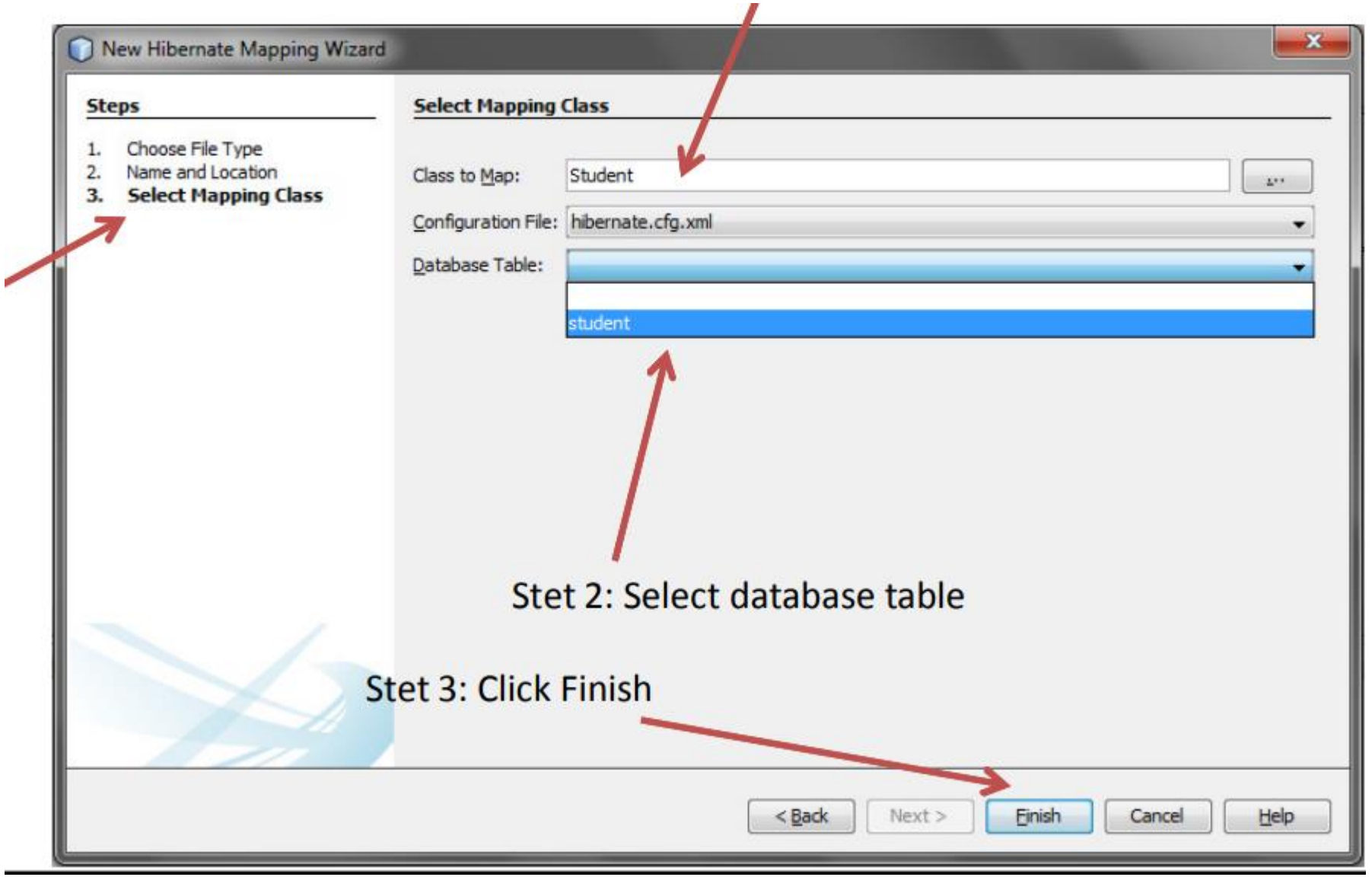
```
<hibernate-configuration>
  <session-factory>
    <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver
</property>
    <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/ddu?zeroD
ateTimeBehavior=convertToNull</property>
    <property name="hibernate.connection.username">root</property>
  </session-factory>
</hibernate-configuration>
```

Step 5: Create Java File

1. Main File with hibernate logic (Main.java)

2. Data Fetching File (Student.java)

Step 6: Create Mapping Configuration File



```
<hibernate-mapping>
  <class name="hibernate.model.Student" table="student2">
<id name="id" type="int" column="id">
<generator class="native"/>
</id>
<property name="firstName" column="first_name" type="string"/>
<property name="lastName" column="last_name" type="string"/>
</class>
</hibernate-mapping>
```

```
public class Main {
public static void main(String[] args) {

    Student st=new Student("Fatema","Vhora",7);

Configuration con = new Configuration().configure();
//Build Session Factory
SessionFactory sf= con.buildSessionFactory();

//Create a session
Session session = sf.openSession();
//Create a transaction to start interaction
session.beginTransaction();
session.save(st);
session.getTransaction().commit();
session.close();
sf.close();
```