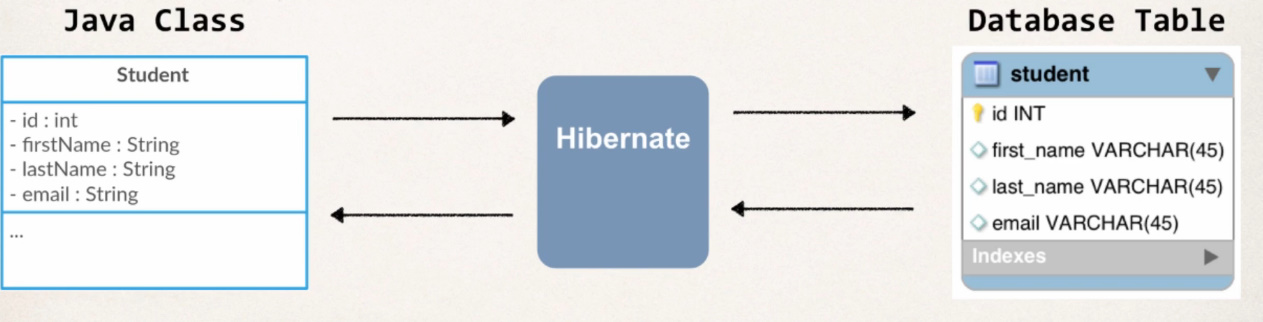
**Ce este**

* Hibernate este un framework pentru a salva obiectele noastre java intr-o baza de date.
* Simplifica JDBC(un Java API) code
* Ofera Object to Relational Mapping (ORM) – s-ar traduce ca Maparea/Reprezentarea obiect-relationala.

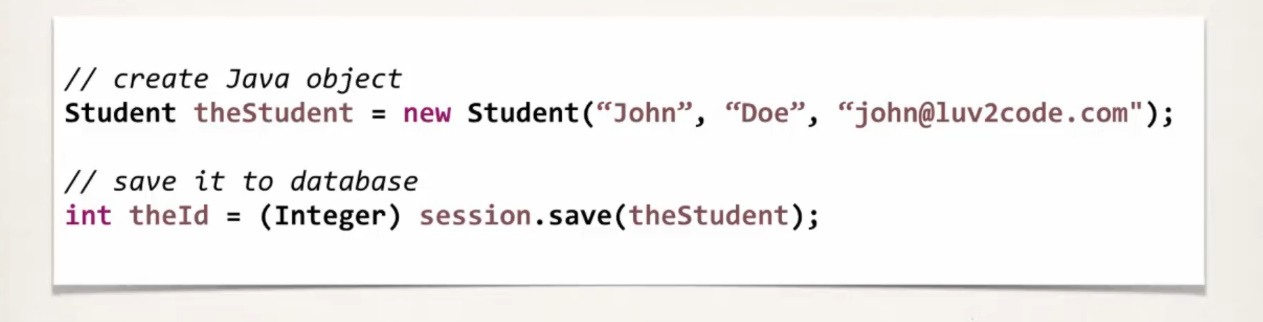
**Object to Relational Mapping(ORM)**

* Aceasta consta in faptul ca developerul defineste maparea/reprezentarea intre Java clase si tabel din baza de date.
* Trebuie sa-i spunem lui Hibernate cum datele din clase se mapeaza sau se regasesc in cele din baza de date.
* Prevede ca mapam o clasa intr-un tabel, adica o facem compatibila cu el.



* Hibernate foloseste JDBC pentru a lucra cu bazele de date. Noi folosim doar Hibernate API, insa in spatele scenei el foloseste JDBC pentru a face toate operatiile necesare in locul nostru.

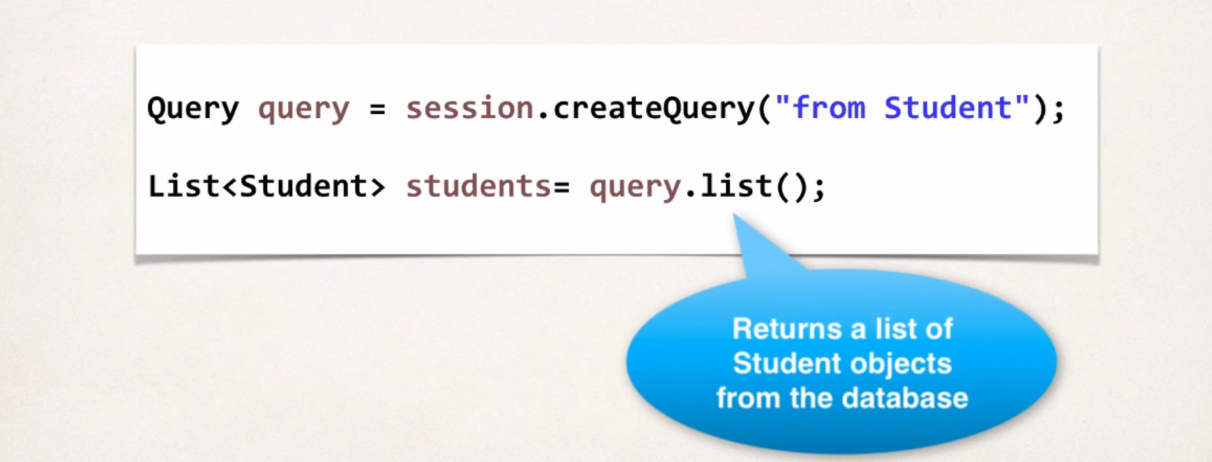
**Saving on Object with Hibernate**

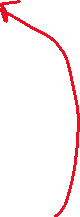
****

* Session este un obiect Hibernate special, si il folosim pentru a salva obiecte in baza de date.
* **Session.save(obj)** – salveaza obiectul in baza de date si returneaza ID al obiectului
* Hibernate face totul pentru noi, si nu trebuie sa scriem codul SQL si sa adaugam in el obiecte.
* **Session.get(obj.class, id)** – returneaza un obiect din baza de date. Hibernate se ocupa de gasirea datelor, crearea obiectului si setarea datelor in obiect.

**Query**

* Hibernate permite sa cream si propriile noastre queries.





**Hibernate Query Language**

* **Query query = session.createQuery(“query”);** - creaza query
* **query.list()** – returneaza tot ce a obtinut cu query din baza de date

**MySql**

CREATE DATABASE IF NOT EXISTS `hb\_student\_tracker`;

USE `hb\_student\_tracker`;

--

-- Table structure for table `student`

--

DROP TABLE IF EXISTS `student`;

CREATE TABLE `student` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`first\_name` varchar(45) DEFAULT NULL,

`last\_name` varchar(45) DEFAULT NULL,

`email` varchar(45) DEFAULT NULL,

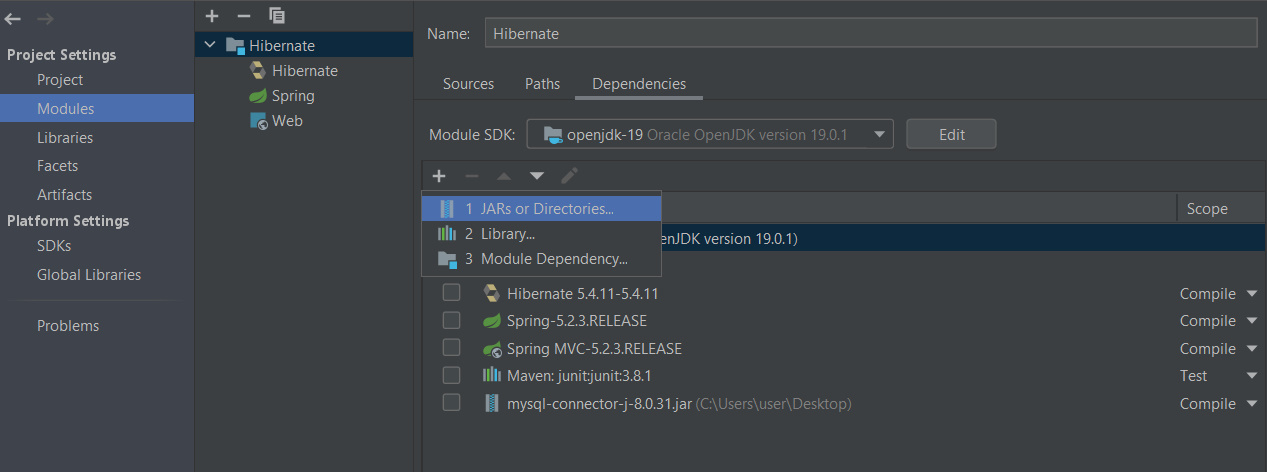
PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=latin1;

**Hibernate 6 suporta Jakarta EE9, dar Spring 5 nu suporta Jakarta EE9, si deci nici Hibernate 6, dar suporta Hibernate 5.**

**Crearea unui proiect cu Hibernate si MySql**

1. Cream proiectul ca si pana acum, adaugand si Hibernate
2. Selectam **View --> ToolsWindows --> DataBase**
3. Apoi click pe **+** si **DataSource**, si alegem MySql. Asa doar facem conexiune la baza de date, nu o adaugam la proiect.
4. Pentru a adauga baza de date la proiect, ne trebuie un jar pentru MySql Connection, care se cheama ConnectorJ, si nu alegem pentru WINDOWS, ci Platform Independent
5. Apoi, dupa ce am creat proiectul in Maven, mergem in **Project Structure**-->**Modules**-->**Dependecies**-> **+** -> **JaRs or Directory** si alegem unde e fisierul .jar connector, si atat



**Verificam Daca se poate face conexiunea la baza de date**

try{  
String jdbUrl = "jdbc:mysql://localhost:3306/hb\_student\_tracker?useSSL=false";  
String user = "testuser";  
String pass = "Frb2eshox!";  
  
System.*out*.println("Connect to DataBase");  
  
 Connection myConn = DriverManager.*getConnection*(jdbUrl,user,pass);  
  
 System.*out*.println("Connection Successful");  
}  
catch (Exception e){  
 e.printStackTrace();  
}

acolo unde e subliniat cu rosu nu e user, ci numele bazei de date la care ne conectam!!

**Hibernate Dev Process**

1. **Adaugam configuration file**

Config file ii spune lui Hibernate cum sa se conecteze la baza de date, folosind JDBC la urma.

Pentru asta, **Files** -> **Project Structure** -> **Modules** -> **+** si il punem in resources.

Config file oferit de compilator e incomplet.

1. Ne asiguram sa avem:

<property>name="connection.driver\_class">com.mysql.jdbc.Driver</property>

1. Adaugam un link pentru baza de date

<property name="connection.url">jdbc:mysql://localhost:3306/hb\_student\_tracker?useSSL=false </property>

1. Dam loginul si parola la baza de date

<property name="connection.username">username </property>   
<property name="connection.password"> password </property>

1. Hibernate 5 cere si urmatorul atribut:

<property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>

Desi toate DBMS folosesc SQL, ele au si ceva a lor specific. Anume tagul de sus permite lui Hibernate sa foloseasca dialectul sau limbajul specific lui MySql8, dar putem seta si altele, depinde ce DBMS folosim.

1. Urmatoarea linie e necesara daca rulam cod in main class:

<property name="hibernate.current\_session\_context\_class">thread</property>

1. Optional:

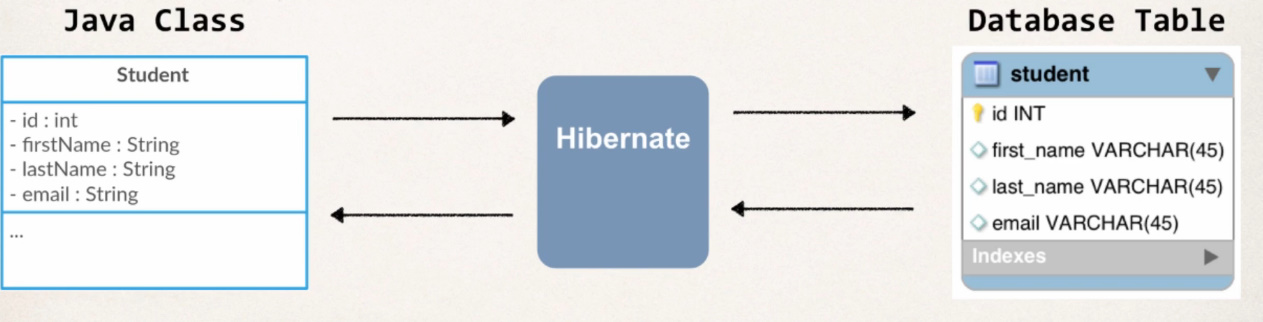
<property name="show\_sql">true</property>

Pentru ca Hibernate sa arate in consola ce comenzi SQL executa.

**Entity class**

* Hibernate foloseste conceptul de Entity class.

**Entity class** – e o clasa care e mapata la un tabel din baza de date. E ca o reprezentare a acestuia sau ca un tabel gol(asta si e entity), doar ca pentru o singura linie, care va fi umpluta cu date din baza de date,pentru fiecare coloana, si poate trimite datele la ea.



Fiecare entity class are ca membri coloane din tabelul din BD.

Entity Class poate fi creata prin 2 metode:

* Xml config file
* Java Annotation(preferabil)

**Annotations si pasi**

Pentru a putea asigura transferul de date, trebuie sa asiguram 2 pasi:

1. Mapam(legam) o clasa la un tabel din baza de date
2. Mapam(legam) fieldurile la coloanele din baza de date
3. **@Entity**

**@Table**(name=”tabel”)

Public class Tabel{

....

}

Prima anotatie doar spune ca clasa e un entity de mapat.

A doua anotatie arata numele tabelului din baza de date cu care lucreaza clasa.

**Insa, anotatia a 2 e suplimentara daca clasa are acelasi nume ca tabelul din BD!**

1. Mapam fielduri(legam) de coloane din DB

@Entity

@Table(name=”tabel”)

Public class Tabel{

**@Id**

**@Column**(name=”id”)

Private int id;

@Column(name=”first\_name”)

Private String firstName;

}

@Id – arata ca ne referim la primary key din tabel

@Column – numele coloanei la care ne referim din tabel

**Daca numele membrului si a coloanei sunt la fel, @Column nu e necesar.**