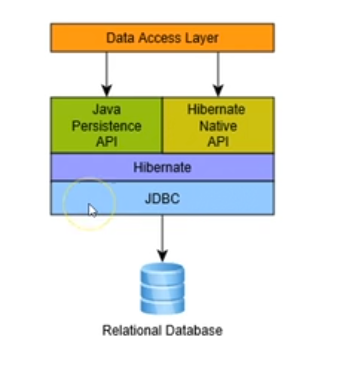
**ORM -**  objektovo relačné mapovanie, mapovanie medzi java objektami a databazou

* Pri JDBC pristupe je nevyhoda, ze ak zmenim nazov stlpca, tak to musim menit na viacerych miestach, ORM to vyrieši, to je jeho vyhoda, ORM nepotrebuje ručne písať SQL prikazy

**Persistance** - uchovanie dát do databázy

**JPA** - Java Persistance API - špecifikácia pre ORM mapovanie, je to štandard(implementácia), ktorý musia niektoré aplikácie podporovať aby kod bol funkcny, aj Hibernate ma vlastny štandard, tak budem používať ten

Rozdiel medzi JPA, JDBC a Hibernate:

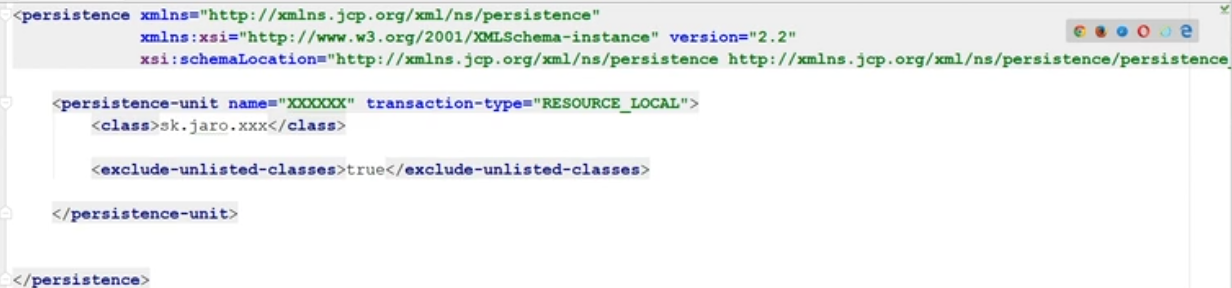


Implementácia Hibernate Native API sa postara o komunikaciu s JDBC, nemusime to priamo my robiť.

**Entity Manager** - nachadza sa v JPA, ulahcuje pracu s entitami, cez dependencies mozno dotiahnut

Nastavenie pomocou persistence.xml:

* Do persitance-unit do class ukladam moje java triedy, ktore budem chcieť mapovať do databazy

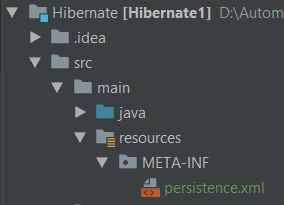


* Ked nastavim transakcny typ resources\_local tak sa o transakcie postarame my, pokial by bol pouzity spring, riesil by to za nas on, aj keby to zadame
* Nastavenie properties pre MS SQL:
* <properties>  
   <property name="javax.persistence.jdbc.driver" value="com.microsoft.sqlserver.jdbc.SQLServerDriver"/>  
   <property name="javax.persistence.jdbc.url" value="jdbc:sqlserver://10.10.3.98;databaseName=Posybe" />  
   <property name="javax.persistence.jdbc.user" value="sa" />  
   <property name="javax.persistence.jdbc.password" value="StarForce4" />  
   <property name="hibernate.dialect" value="org.hibernate.dialect.SQLServerDialect"/>  
   <property name="show\_sql" value="true"/>  
   <property name="hibernate.temp.use\_jdbc\_metadata\_defaults" value="false"/>  
  </properties>
* Ak nechcem pisat sql prikazy, tak pridam property:

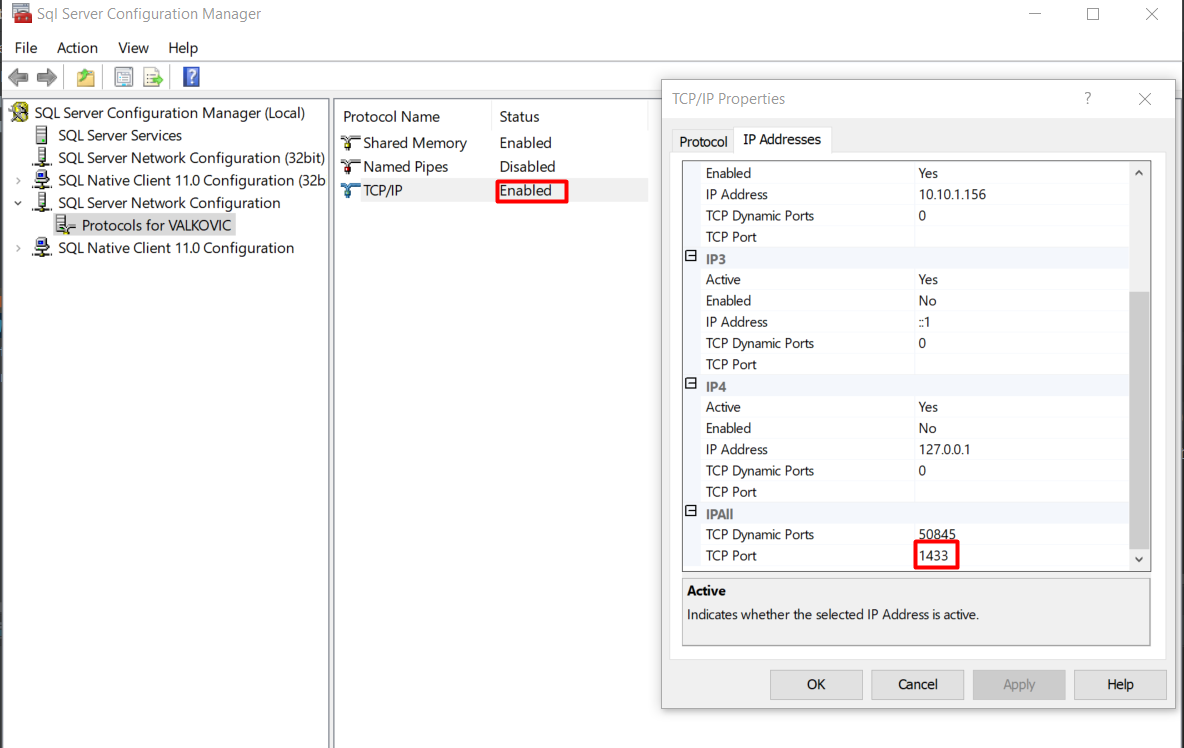
<property name="hibernate.hbm2ddl.auto" value="update"/>

Ak by som miesto update dal create\_drop, vzdy by sa mi vsetko z databazy zmazalo a naplnilo, v databaze by nic nebolo

* Ak vypise program chybu ze nenaslo META-INF/persistence.xml, je potrebne skontrolovat cestu a nazov xml, hibernate ocakava pevnu cestu s pevnym nazvom suboru persistence.xml



* Pokial sa program nedokaze pripojit k databaze, vypise connect refuse na nejakom porte, je potrebne v MSSQL povolit TCP/IP na konkretnom porte, na ktorom chybu zahlasi, u mna to bol port 1433



Pravidlá triedy pre mapovanie:

* Trieda mapovaná nesmie byť final
* Nesmie byť vnútorne vnorená trieda
* Obsahuje public alebo protected bez parametrický konštruktor
* Fieldy, ktoré sa mapujú na databázu musia byť prístupné len cez get a set metódy
* Trieda je označená ako entita pomocou @Entity anotacie alebo pomocou XML nastavenia
* Enum ani interface nemoze byt entita
* Abstraktne triedy mozu byt entity
* Entity môžu dediť(extends) neentitné triedy a naopak

Vytvorenie MS SQL tabulky:

CREATE TABLE Messages (

ID INT IDENTITY(1,1) PRIMARY KEY NOT NULL,

N\_MESSAGE VARCHAR(255),

CREATE\_DATE datetime default getdate(),

);

//IDENTITY(1,1) urcuje autoincrement

Definovanie triedy ako entity:

@Entity  
//entitou definujem ze sa budem pripajat k databaze(pouzije hibernate)  
@Table(name = "Messages")  
//takto definujem na ktoru tabulku sa bude pripajat  
public class Message {

@Id  
 //anotacia pre parameter ktory sa bude inkrementovat, bude to jeho identifikator

@GeneratedValue(strategy = GenerationType.*IDENTITY*)

//generovane ID podla IDENTITY

@Column(name = "ID")  
 //definovanie stlpca v databaze, ak je rovnake // ako premenná, netreba column definovat, // ako napr tu  
 private Long id;

@Column(name = "N\_MESSAGE")  
 private String message;

@Column(name = "CREATE\_DATE",insertable = false)  
 private Date CREATE\_DATE;  
 //pridane len kvôli tomu ze sa to nachadza v // databaze,ORM to sam rozozna a namapuje, // databaza si uz hodnotu sama vygeneruje,

// kvoli selectu to potrebujem

//pokial sa tento parameter ma generovat // databazou, treba dat insertable = false

Pokial sa trieda a premenne volaju rovnako v programe ako v databaze, **netreba** pisat anotacie @Table a @Column, myslim ze moze byt aj rozdiel v malych a velkych pismenach, mapovačka si s tym poradí

Strategie generovania ID:

* **AUTO** - default, vyberie strategiu hibernate, odporuca sa nepouzivat
* **IDENTITY** - zvacsovanie ID je ponechanie na databazu, z pohladu databazy idealne riesenie
* **SEQUENCE** - pouzije sa generovanie ID pomocou databazovych sekvencii
* **TABLE -** aktualnu hodnotu ID si uklada do databazovej tabulky

**Zapis do DB:**

public class App   
{  
 public static void main( String[] args )  
 {  
 EntityManagerFactory entityManagerFactory =  
 Persistence.*createEntityManagerFactory*("nazovPersistenceUnit");  
 EntityManager entityManager = entityManagerFactory.createEntityManager();  
 //vytvorenie entity managera, ktory je naviazany na nazov v persistence.xml  
 // pomocou persistanceUnitName  
  
 entityManager.getTransaction().begin();  
 //zacinam transakciu  
  
 entityManager.persist(new Message("Prva sprava"));  
 //fakeovy zapis do databazy, realny az po commite  
 entityManager.persist(new Message("Druha sprava"));  
 //fakeovy zapis do databazy, realny az po commite  
  
 entityManager.getTransaction().commit();  
 //az po tomto prikaze sa realne ulozia hodnoty do databazy  
  
 entityManager.close();  
 //zatvorenie transakcie  
 }

**Citanie z DB**:

//CITANIE Z DB  
 entityManager.getTransaction().begin();  
 List<Message> messageList =  
 entityManager.createQuery("from Message"

,Message.class).getResultList();  
  
 //vypis listu sprav  
 for(Message msg:messageList)  
 {  
 System.*out*.println("id: "+msg.getId() +  
 "\nsprava: "+msg.getMessage() +   
 "\ndatumCas: "+msg.getCreateDate());  
 }  
  
  
 entityManager.getTransaction().commit();  
 entityManager.close();  
 //zatvorenie transakcie

**Konverzia** enumeratora na znak do databazy:

@Convert(converter = PohlavieConverter.class)  
//tato anotacia tam skonvertuje enum na znak, ktory ocakava databaza  
private Pohlavie pohlavie;

* PohlavieConverter.class je potrebne pozriet v projekte Hibernate, tiez enumerator Pohlavie
* Converter mozno pouzit konvertovanie z hocijakeho typu na iny

**Generovanie** entít z existujúcej DB:

* IntelliJ Idea pravdepodobne vo free verzii nepodporuje vytvaranie entít z existujúcej databázy, preto ak bude potrebné naťahať entity, tak použijeme netbeans, pripojenie beží na porte 1433

**Čítanie na základe Query:**

entityManager.getTransaction().begin();

List<Object[]> osoba =

entityManager.createNativeQuery(“select \* from osoba”).getResultList();

**@MappedSuperclass**

* Používa sa pre parrenta, ktoreho ina trieda extenduje a je v entite, ked pridam tuto anotaciu pre rodiča, tak sa budu brať do uvahy aj jeho parametre, budu sa doplnat do tabulky

**QueryDsl**

* Sluzi na vytvaranie podmienkovych prikazov, napr where, bez sql prikazu

**Query s parametrom**

* Vo query zadavam parameter nasledovne :nazov\_parametra
* Nasledne mu priradim hodnotu cez metodu setParameter(“nazov\_parametra”,hodnota\_parametra), priklad je nasledovny:
* entityManagerUIM.createNativeQuery("SELECT \"UserName\" FROM ids.\"Users\"\n order by \"UserName\" asc\n limit :usersOnPage" ,Users.class).setParameter("usersOnPage",usersOnPage).getResultList();
* druhy sposob je nie cez nazov, ale cez index, vtedy sa do query da otaznik a v setparameter sa nastavi hodnota pomocou indexu(index zacina asi od 1)

entityManagerUIM.createNativeQuery("SELECT \"UserName\" FROM ids.\"Users\"\n" +  
 "order by \"UserName\" asc\n" +  
 "limit ?" ,Users.class).setParameter(1,usersOnPage).getResultList();