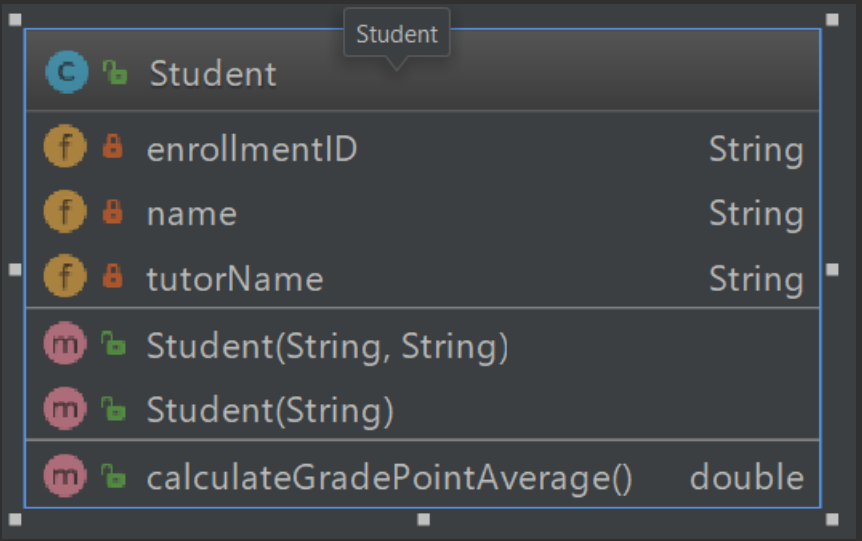
Hibernate i JPA

Co to jest klasa encji

- Zwykla klasa POJO z:
  - Adnotacja @Entity
  - Pustym konstruktorem
  - Jednym z pol nominowanym jako ID przez wstawienie adnotacji @ID

Założmy, że w aplikacji mamy klasę:




```
public class Student {
    private String enrollmentID;
    private String name;
    private String tutorName;

    public Student(String name,String tutorName) {
        this.name = name;
        this.tutorName = tutorName;
    }

    public Student(String name) {
        this.name = name;
    }

    public double calculateGradePointAverage(){
        /*
        Some complex business logic here
        */
        return 0;
    }
}
```

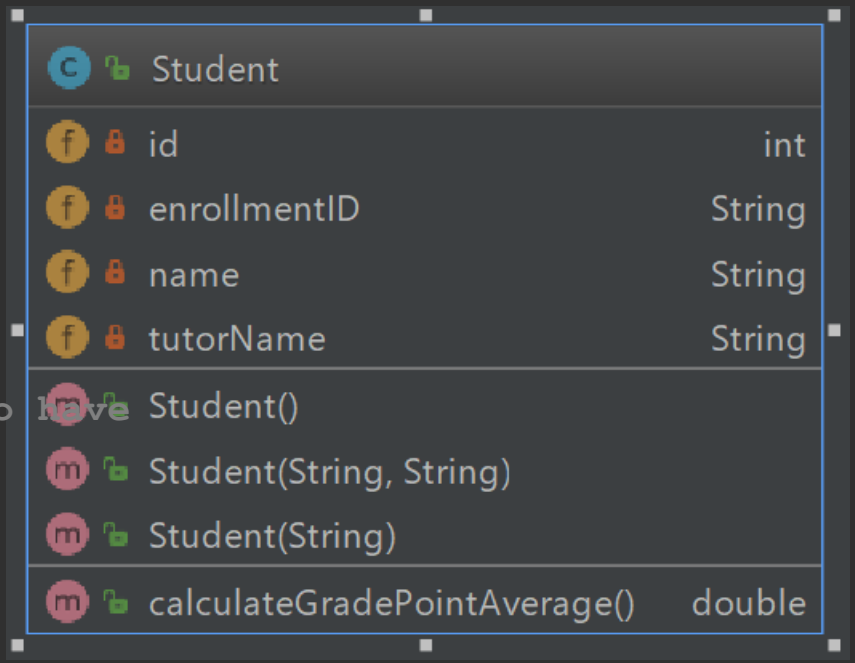
..i chcemy aby stała się ona persystentną klasą modelu, zarządzaną przez Hibernate'a. Modyfikacje które musimy wprowadzić to:

```
//Required by Hibernate
@Entity
public class Student {
//Required by Hibernate
    @Id
    @GeneratedValue(
    strategy = GenerationType.AUTO)
    private int dbID;

    private String enrollmentID;
    private String name;
    private String tutorName;

    public Student(){
    }

    public Student(String name,String tutorName) {
        this.name = name;
        this.tutorName = tutorName;
    }

    public Student(String name) {
        this.name = name;
    }

    //          Of course EntityClass is also going to have
    //          a complex business logic methods
    public double calculateGradePointAverage(){
        return 0;
    }
}
```

Adnotowanie klasy jako klasy Encji @Entity

Nominowanie jednego z pól jako id (@Id) połączone najczęściej z z delegowaniem do Hibernate'a generowania jego wartości zgo dnie z ustaloną strategią

Dodanie domyślnego konstruktora (przy zapisywaniu żaden problem, wyjdzie przy odczycie

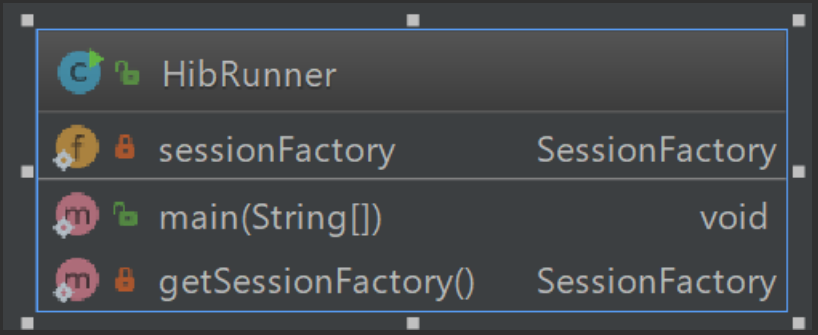


Żeby tego użyć, potrzebujemy:

Dostarczyć w projekcie konfigurację dla Hibernate'a: (domyślnie plik hibernate.cfg.xml w głównym drzewie projektu…..

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-
3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property
name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</p
roperty>
        <property
name="connection.url">jdbc:derby://localhost/MyDatabase</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">create</property>
        <mapping class="Domain.Student"></mapping>
    </session-factory>
</hibernate-configuration>
```

.....a w samej aplikacji:

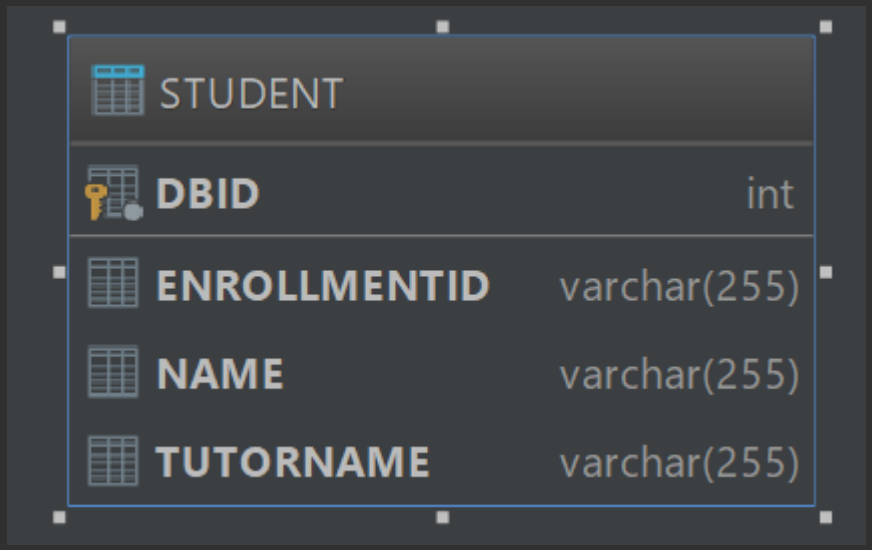


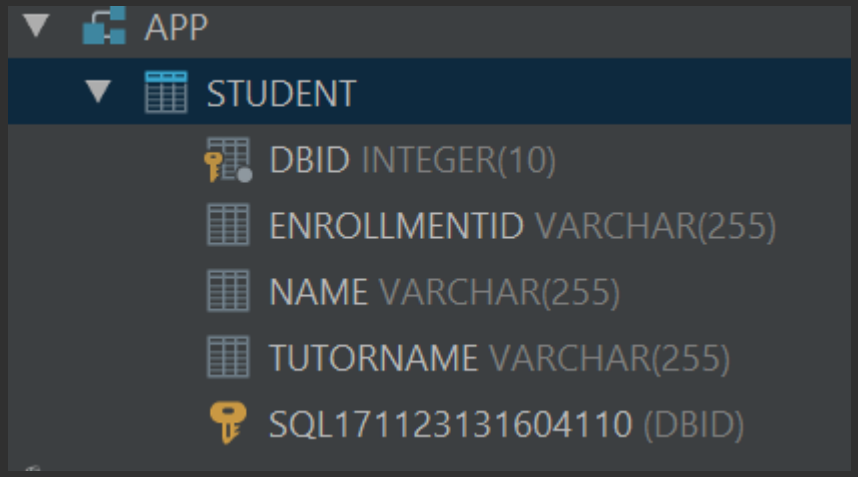

```
public class HibRunner {
    private static SessionFactory sessionFactory = null;

    public static void main(String[] args) {
        Student student = new Student("Studek Studkowski",
                "Tutek Tucinski");
        sessionFactory = getSessionFactory(); // Dostarczamy SessionFactory (jako
        Session session = sessionFactory.openSession();// Otwieramy sesję
        Transaction tx = session.beginTransaction(); // Rozpoczynamy transakjc
        session.save(student);// Pracujemy z warstwą persystentną
        tx.commit();  // Zatwierdzamy transakcję
        session.close();// Zamykamy sesję
    }

    private static SessionFactory getSessionFactory() {
        if (sessionFactory == null) {
            Configuration configuration = new Configuration();
            sessionFactory =
             configuration.configure().buildSessionFactory();
        }
        return sessionFactory;
    }
}
```

.....po uruchomieniu powyższego otrzymamy....

Hibernate: create table Student (dbID integer not null, enrollmentID varchar(255), name varchar(255), tutorName varchar(255), primary key (dbID))

...co daje...





...oraz...

Hibernate: values next value for hibernate_sequence

Hibernate: insert into Student (enrollmentID, name, tutorName, dbID) values (?, ?, ?, ?)

...co daje...

	DBID	ENROLLMENTID	NAME	TUTORNAME
1	1	<null>	Studek Studkowski	Tutek Tucinski

..jak to jest z generowaniem wartości @Id….

```
public static void main(String[] args) {
    Student student = new Student("Studek Studkowski",
            "Tutek Tucinski");

    System.out.println("Student id is: "+student.getDbID());
    sessionFactory = getSessionFactory();                    // Student id is: 0
    Session session = sessionFactory.openSession();
    Transaction tx = session.beginTransaction();
    session.save(student);

    System.out.println("...and now student id is:
                    "+student.getDbID());      //...and now student id is: 1

    tx.commit();
    session.close();
}
```

...powyższe generuje następującą sekwencję wywołań….

Hibernate: create sequence hibernate_sequence start with 1 increment by 1

Hibernate: create table Student (dbID integer not null, enrollmentID varchar(255), name varchar(255), tutorName varchar(255), primary key (dbID))

Hibernate: insert into Student (enrollmentID, name, tutorName, dbID) values (?, ?, ?, ?)

..jak coś odczytać z BD ….

```
public static void main(String[] args){
    sessionFactory=getSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction tx = session.beginTransaction();
    Student foundStudent = session.get(Student.class,1);
   System.out.println(foundStudent); // najprościej użyć session.get()
    tx.commit();
    session.close();
}
```

...powyższe generuje następującą sekwencję wywołań….

Hibernate: select student0_.dbID as dbID1_0_0_, student0_.enrollmentID as enrollme2_0_0_, student0_.name as name3_0_0_, student0_.tutorName as tutorNam4_0_0_ from Student student0_ where student0_.dbID=?

Student{name='Studek Studkowski'}

	DBID	ENROLLMENTID	NAME	TUTORNAME
1	1	<null>	Studek Studowski	Tutek Tutowski
2	2	<null>	Studka Stucinska	Tutka Tucinska

...gdyby brakło domyślnego konstruktora...

Exception in thread "main" org.hibernate.InstantiationException: No default constructor for entity: : Domain.Student

..a gdyby zrobić tak ….

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

    sessionFactory=getSessionFactory();
    Session session = sessionFactory.openSession();

    Student foundStudent = session.get(Student.class,1);

   System.out.println(foundStudent);

    session.close();
}
```

...powyższe generuje następującą sekwencję wywołań….

Hibernate: select student0_.dbID as dbID1_0_0_, student0_.enrollmentID as enrollme2_0_0_, student0_.name as name3_0_0_, student0_.tutorName as tutorNam4_0_0_ from Student student0_ where student0_.dbID=?

Student{name='Studek Studkowski'}

...w sumie to samo….. to „z” czy „bez” transakcji? → do samodzielnego przestudiowania

https://stackoverflow.com/questions/818074/transactions-for-read-only-db-access

https://stackoverflow.com/questions/13539213/why-do-i-need-transaction-in-hibernate-for-read-only-operation

..jak coś usunąć z BD ….

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

    sessionFactory = getSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction tx = session.beginTransaction();
    Student foundStudent = session.get(Student.class,1);
    session.delete(foundStudent);
    tx.commit();
    session.close();
}
```

	DBID	ENROLLMENTID	NAME	TUTORNAME
1	1	<null>	Studek Studowski	Tutek Tutowski
2	2	<null>	Studka Stucinska	Tutka Tucinska

...powyższe generuje następującą sekwencję wywołań….

Hibernate: select student0_.id as id1_0_0_, student0_.enrollmentID as enrollme2_0_0_, student0_.name as name3_0_0_, student0_.tutorName as tutorNam4_0_0_ from Student student0_ where student0_.id=?

Hibernate: delete from Student where id=?

	DBID	ENROLLMENTID	NAME	TUTORNAME
1	2	<null>	Studka Stucinska	Tutka Tucinska

..jak coś zmodyfikować w BD ….

	DBID	ENROLLMENTID	NAME	TUTORNAME
1	2	<null>	Studka Stucinska	Tutka Tucinska

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

    sessionFactory=getSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction tx = session.beginTransaction();

    Student foundStudent = session.get(Student.class,2);
    foundStudent.setTutorName("Tutor Tutorowski");

    tx.commit();
    session.close();
}
```

...powyższe generuje następującą sekwencję wywołań….

Hibernate: select student0_.id as id1_0_0_, student0_.enrollmentID as enrollme2_0_0_, student0_.name as name3_0_0_, student0_.tutorName as tutorNam4_0_0_ from Student student0_ where student0_.id=?

Hibernate: update Student set enrollmentID=?, name=?, tutorName=? where id=?

	DBID	ENROLLMENTID	NAME	TUTORNAME
1	2	<null>	Studka Stucinska	Tutor Tutorowski

Migracja modelu

```
<property name="hbm2ddl.auto">update</property>
```

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

    sessionFactory=getSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction tx = session.beginTransaction();
    Student student = new Student("student studencki");
    session.save(student);

    tx.commit();
    session.close();
}
```

Student	
enrollmentID	String
name	String
numberOfCourses	int
Student()	
Student(String, String)	
Student(String)	
toString()	String
calculateGradePointAverage()	double
tutorName	String
id	int

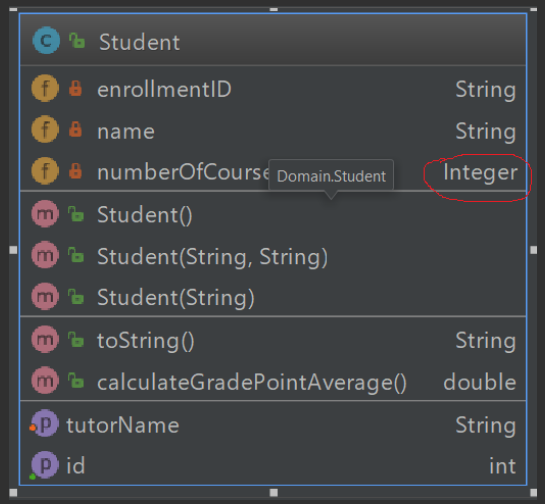
```
private String enrollmentID;
private String name;
private String tutorName;
private int numberOfCourses;
```

Caused by: ERROR 42X14: 'NUMBEROFCOURSES' is not a column in table or VTI 'APP.STUDENT'.
at org.apache.derby.client.am.ClientStatement.completeSqlca(Unknown Source)
at org.apache.derby.client.net.NetStatementReply.parsePrepareError(Unknown Source)

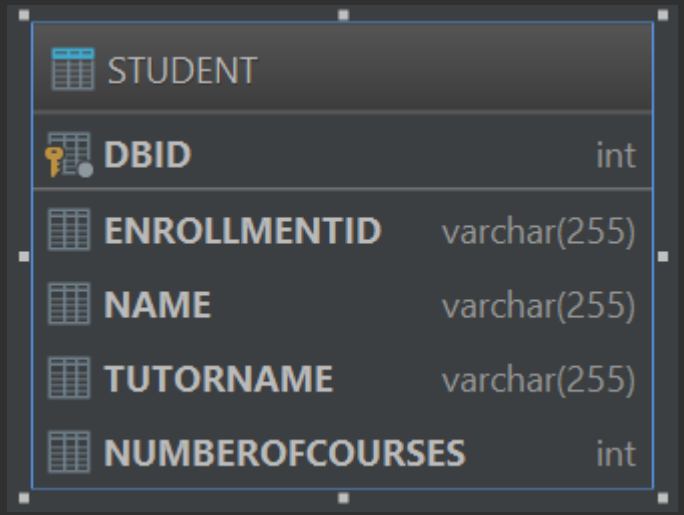
Hibernate: alter table APP.STUDENT add column numberOfCourses integer not null

Caused by: java.sql.SQLSyntaxErrorException: In an ALTER TABLE statement, the column 'NUMBEROFCOURSES' has been specified as NOT NULL and either the DEFAULT clause was not specified or was specified as DEFAULT NULL.

```
private String enrollmentID;
private String name;
private String tutorName;
private Integer numberOfCourses;
```



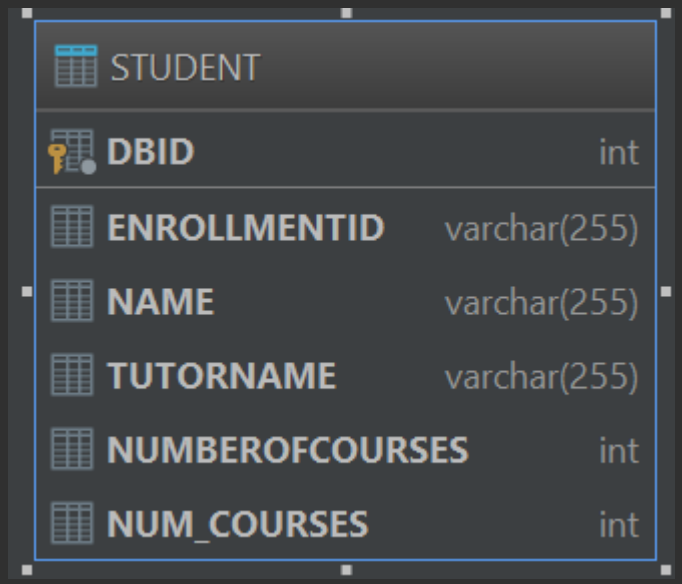
Hibernate: alter table APP.STUDENT add column numberOfCourses integer
Hibernate: values next value for hibernate_sequence
Hibernate: insert into Student (enrollmentID, name, numberOfCourses, tutorName, id) values (?, ?, ?, ?, ?)

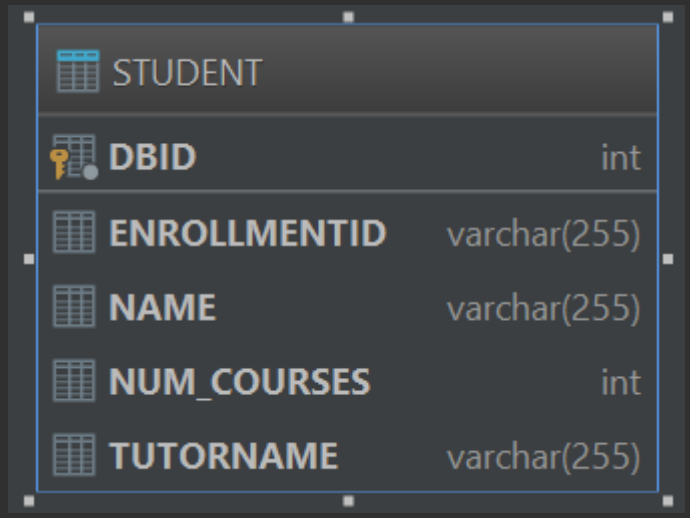


	DBID	ENROLLMENTID	NAME	TUTORNAME	NUMBEROFCOURSES
1	2	<null>	Studka Stucinska	Tutor Tutorowski	<null>

```
private String enrollmentID;
private String name;
private String tutorName;

@Column(name="NUM_COURSES")
private Integer numberOfCourses;
```

Hibernate: alter table APP.STUDENT add column NUM_COURSES integer





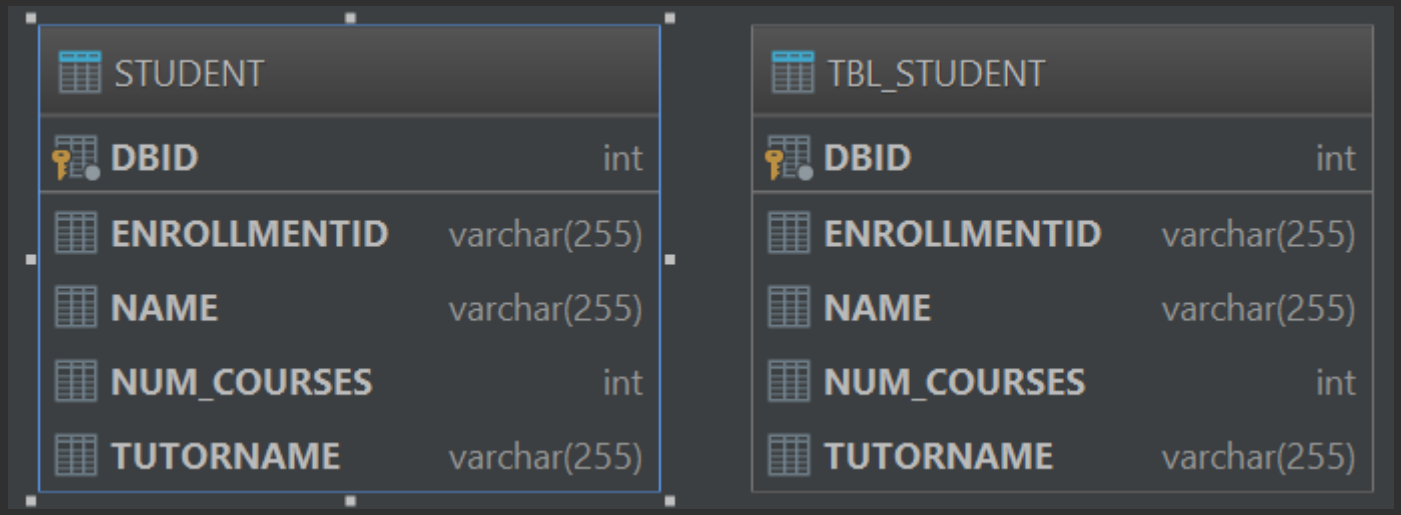

```
<property name="hbm2ddl.auto">create</property>
```

Hibernate: create table Student (dbID integer not null, enrollmentID varchar(255), name varchar(255), NUM_COURSES integer, tutorName varchar(255), primary key (dbID))

```
@Entity
@Table(name="TBL_STUDENT")
public class Student {
```

```
<property name="hbm2ddl.auto">update</property>
```

Hibernate: create table TBL_STUDENT (dbID integer not null, enrollmentID varchar(255), name varchar(255), NUM_COURSES integer, tutorName varchar(255), primary key (dbID))




```
<property name="hbm2ddl.auto">create</property>
```

Hibernate: create table TBL_STUDENT (id integer not null, enrollmentID varchar(255), name varchar(255), NUM_COURSES integer, tutorName varchar(255), primary key (id))

.....do samodzielnego przestudiowania

field vs propertyAccess

https://stackoverflow.com/questions/594597/hibernate-annotations-which-is-better-field-or-property-access

```
public class Student {
//Required by Hibernate
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int dbID;

    private String enrollmentID;
    private String name;
    private String tutorName;

    @Column(name="NUM_COURSES")
    private Integer numberOfCourses;

    private double averageScoreAcrossAllExams;
```

TBL_STUDENT	
DBID	int
AVERAGESCOREACROSSALLEXAMS	float(52)
ENROLLMENTID	varchar(255)
NAME	varchar(255)
NUM_COURSES	int
TUTORNAME	varchar(255)

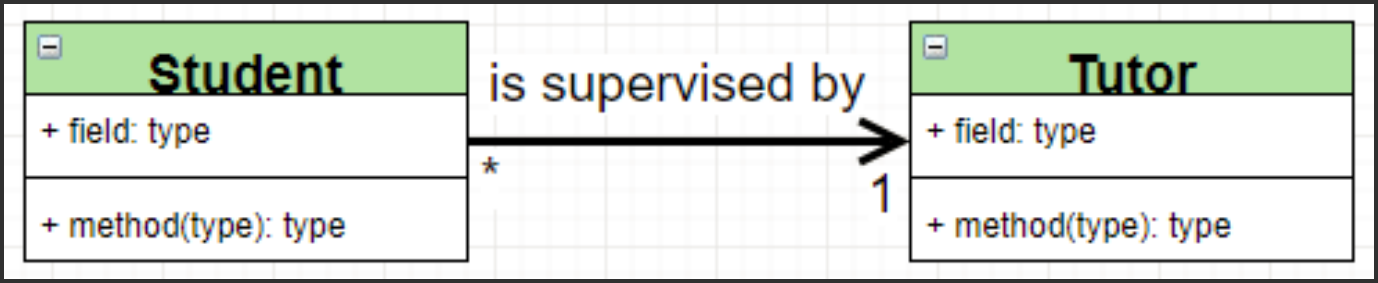
Hibernate: create table TBL_STUDENT (dbID integer not null, averageScoreAcrossAllExams double not null, enrollmentID varchar(255), name varchar(255), NUM_COURSES integer, tutorName varchar(255), primary key (dbID))

```
  @Transient
  private double averageScoreAcrossAllExams;
```

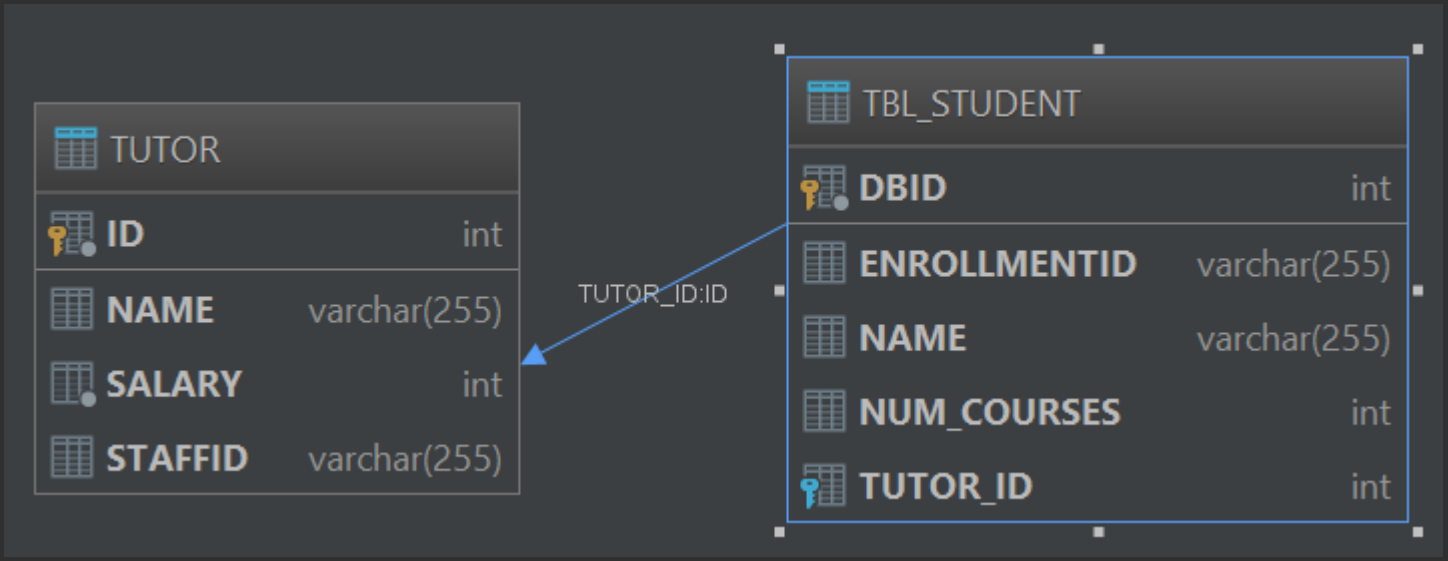
TBL_STUDENT	
DBID	int
ENROLLMENTID	varchar(255)
NAME	varchar(255)
NUM_COURSES	int
TUTORNAME	varchar(255)

Hibernate: create table TBL_STUDENT (dbID integer not null, enrollmentID varchar(255), name varchar(255), NUM_COURSES integer, tutorName varchar(255), primary key (dbID))

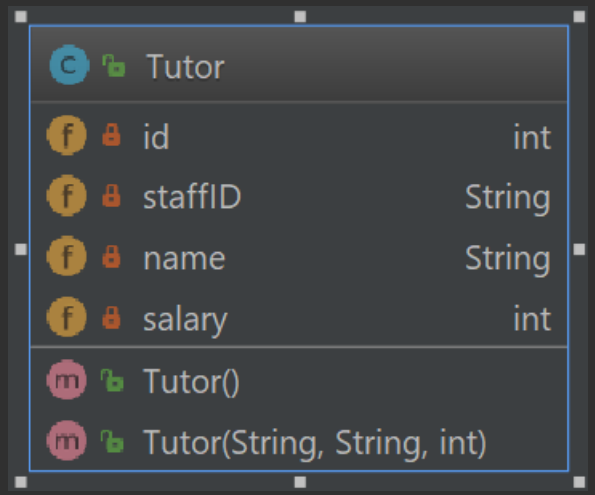
...a gdybyśmy chcieli...



….mieć zamodelowane jako….



```java
@Entity
public class Tutor {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String staffID;
    private String name;
    private int salary;
//    Required by Hibernate
    public Tutor() {
    }
//    Business constructor
    public Tutor(String staffID, String name,
        this.staffID = staffID;
        this.name = name;
        this.salary = salary;
    }
}
```




```java
public class Student {
//Required by Hibernate
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    private String enrollmentID;
    private String name;
//    private String tutorName;
    private Tutor tutor;
```

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

    sessionFactory = getSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction tx = session.beginTransaction();

    session.save(student);
    tx.commit();
    session.close();
}
```

Exception in thread "main" org.hibernate.MappingException: Could not determine type for: Domain.Tutor, at table: Student, for columns: [org.hibernate.mapping.Column(tutor)]

```
@ManyToOne
// @JoinColumn(name="TUTOR_FK")
private Tutor tutor;
```

Exception in thread "main" org.hibernate.AnnotationException: @OneToOne or @ManyToOne on Domain.Student.tutor references an unknown entity: Domain.Tutor

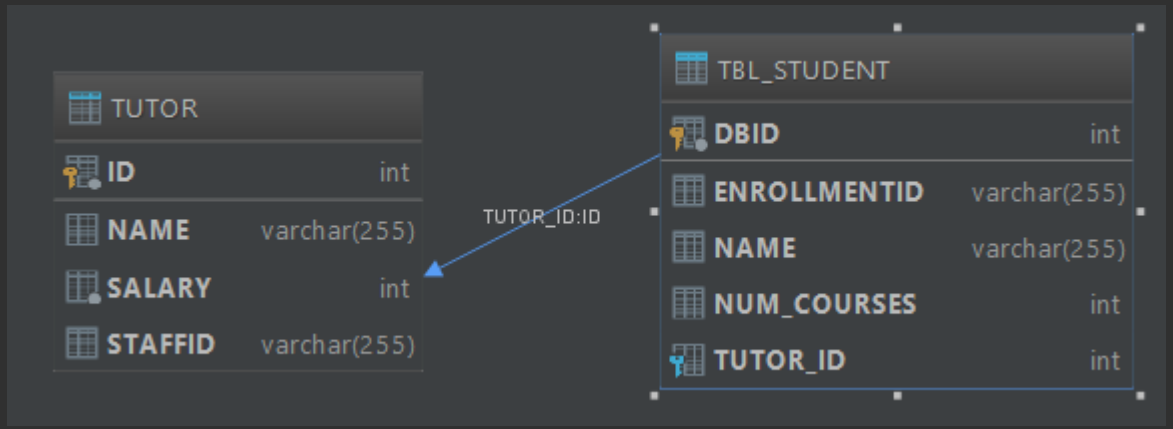
```
<mapping class="Domain.Tutor"></mapping>
```

...powyższe generuje następującą sekwencję wywołań....

Hibernate: create table TBL_STUDENT (dbID integer not null, enrollmentID varchar(255), name varchar(255), NUM_COURSES integer, tutor_id integer, primary key (dbID))

Hibernate: create table Tutor (id integer not null, name varchar(255), salary integer not null, staffID varchar(255), primary key (id))

Hibernate: alter table TBL_STUDENT add constraint FKs22aayhbkrka84vw206u2cuk5 foreign key (tutor_id) references Tutor



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

    sessionFactory = getSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction tx = session.beginTransaction();
    Student student = new Student("Studus Studowski");
    Tutor newTutor = new Tutor("Tut-23", "Tutus Tutowski", 20000);
    session.save(student);
    session.save(newTutor);
    student.setTutor(newTutor);
    System.out.println(student.getTutorName());
    tx.commit();
    session.close();
}
```

...powyższe generuje następującą sekwencję wywołań....

Hibernate: insert into TBL_STUDENT (enrollmentID, name, NUM_COURSES, tutor_id, dbID) values (?, ?, ?, ?, ?)

Hibernate: insert into Tutor (name, salary, staffID, id) values (?, ?, ?, ?)

Hibernate: update TBL_STUDENT set enrollmentID=?, name=?, NUM_COURSES=?, tutor_id=? where dbID=?

	ID	NAME	SALARY	STAFFID
1	2	Tutus Tutowski	20000	ABC123

	DBID	ENROLLMENTID	NAME	NUM_COURSES	TUTOR_ID
1	1	<null>	studus studowski	<null>	2

	DBID	ENROLLMENTID	NAME	NUM_COURSES	TUTOR_ID
1	1	<null>	studus studowski	<null>	2

	ID	NAME	SALARY	STAFFID
1	2	Tutus Tutowski	20000	ABC123

```
public static void main(String[] args) {
        sessionFactory = getSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();

        Student foundStudent = session.get(Student.class,1);
        Tutor supervisor = foundStudent.getSupervisor();
        System.out.println(supervisor.getName());

      tx.commit();
        session.close();
    }
```

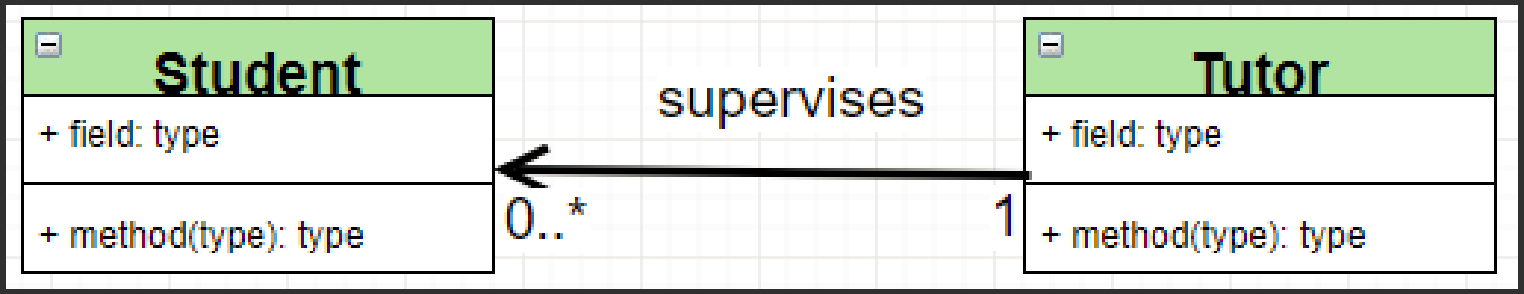
Tutus Tutowski

Hibernate: select student0_.dbID as dbID1_0_0_, student0_.enrollmentID as enrollme2_0_0_, student0_.name as name3_0_0_, student0_.NUM_COURSES as NUM_COUR4_0_0_, student0_.tutor_id as tutor_id5_0_0_, tutor1_.id as id1_1_1_, tutor1_.name as name2_1_1_, tutor1_.salary as salary3_1_1_, tutor1_.staffID as staffID4_1_1_
from TBL_STUDENT student0_ left outer join Tutor tutor1_
on student0_.tutor_id=tutor1_.id where student0_.dbID=?

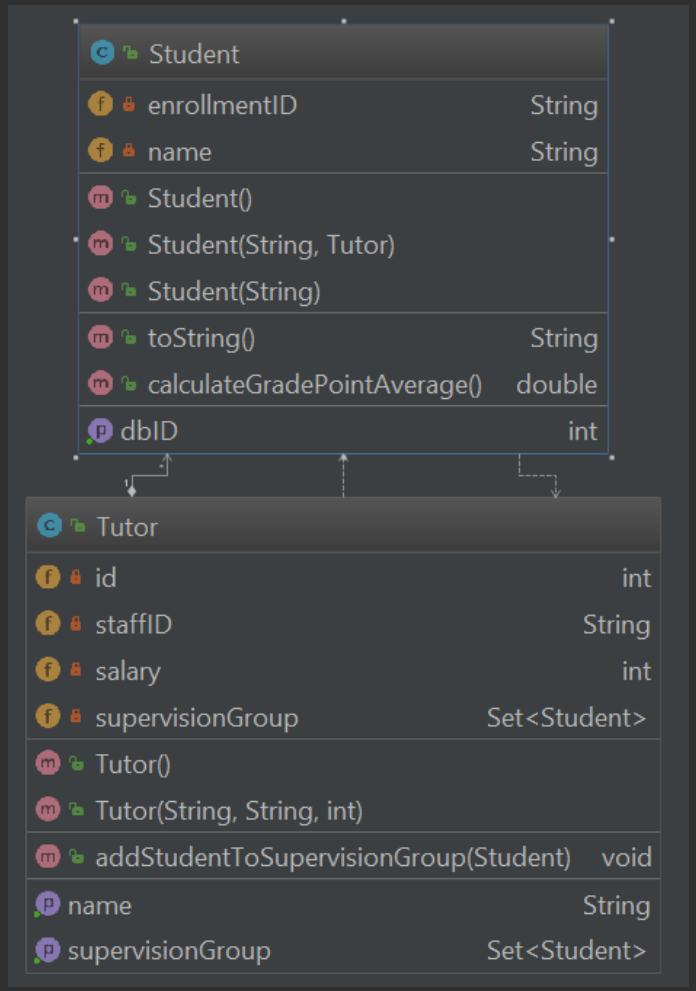
```java
public static void main(String[] args) {
        sessionFactory = getSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();

        Student foundStudent = session.get(Student.class,1);
        foundStudent.setTutor(null);

        tx.commit();
        session.close();
    }
```

	DBID	ENROLLMENTID	NAME	TUTOR_FK
1	1	<null>	studus studencki	<null>

...a gdybyśmy chcieli...



```java
@Entity
public class Tutor {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String staffID;
    private String name;
    private int salary;
    private Set<Student> supervisionGroup;

//    Required by Hibernate
    public Tutor() {
    }
```



Przy próbie uruchomienia otrzymamy:

Exception in thread "main" org.hibernate.MappingException: Could not determine type for: java.util.Set, at table: Tutor, for columns: [org.hibernate.mapping.Column(supervisionGroup)]

Po adnotowaniu grupy studentów jako:

```
@OneToMany
private Set<Student> supervisionGroup;
```

I uruchomieniu otrzymujemy następującą sekwencję wywołań

Hibernate: create table TBL_STUDENT (dbID integer not null, enrollmentID varchar(255), name varchar(255), NUM_COURSES integer, primary key (dbID))

Hibernate: create table Tutor (id integer not null, name varchar(255), salary integer not null, staffID varchar(255), primary key (id))

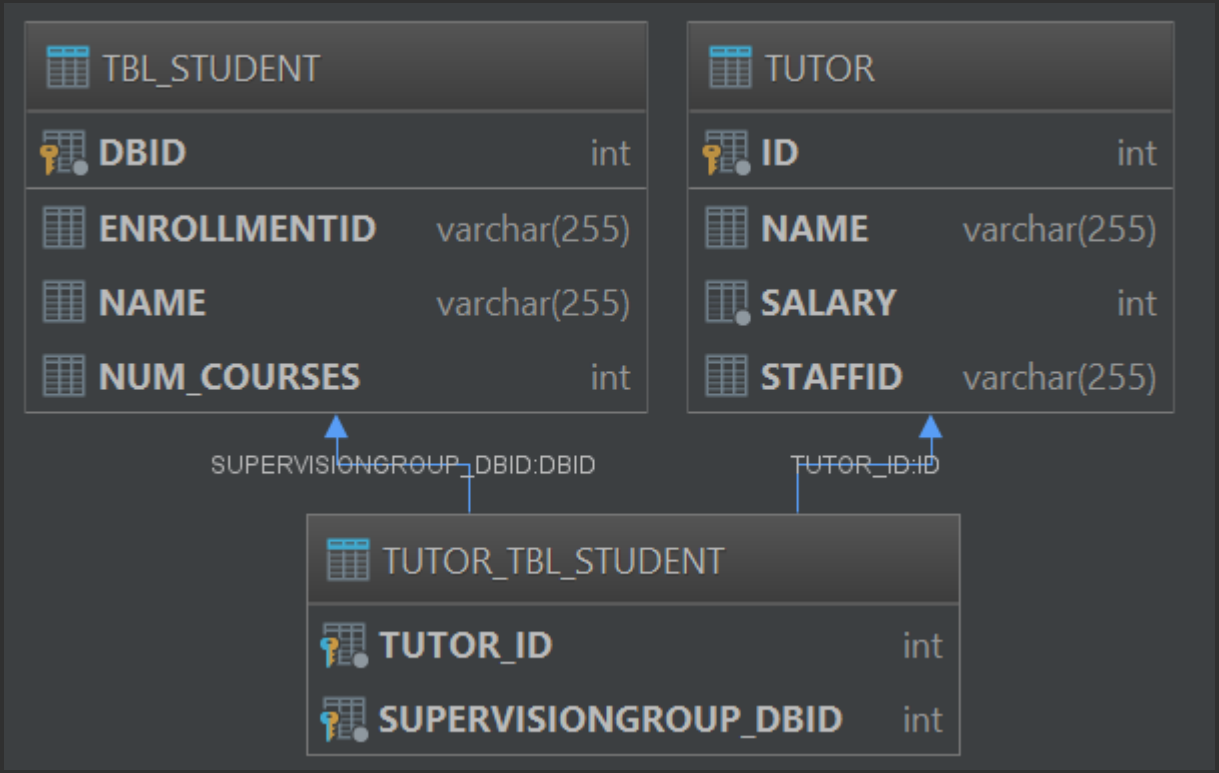
Hibernate: create table Tutor_TBL_STUDENT (Tutor_id integer not null, supervisionGroup_dbID integer not null, primary key (Tutor_id, supervisionGroup_dbID))

Hibernate: alter table Tutor_TBL_STUDENT add constraint UK_8k1o4budixtdrpc79qq92ffm7 unique (supervisionGroup_dbID)

Hibernate: alter table Tutor_TBL_STUDENT add constraint FKiri63xpmai9kafijeisrjv76o foreign key (supervisionGroup_dbID) references TBL_STUDENT

Hibernate: alter table Tutor_TBL_STUDENT add constraint FKbce5ubrq4vsim0wkp9pe5170c foreign key (Tutor_id) references Tutor

...czyli w praktyce otrzymujemy...



```java
public static void main(String[] args) {
    sessionFactory = getSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction tx = session.beginTransaction();
    Student student1 = new Student("Studek Studowski");
    Student student2 = new Student("Stutek Stucinski");
    Student student3 = new Student("Stutka Stucinska");
    session.save(student1);
    session.save(student2);
    session.save(student3);
    Tutor tutor = new Tutor("TutekStaffID","Tutek Tutkowski",5000);
    session.save(tutor);
    tutor.addStudentToSupervisionGroup(student1);
    tutor.addStudentToSupervisionGroup(student2);
    tutor.addStudentToSupervisionGroup(student3);
    tx.commit();
    session.close();
}
```

```
Hibernate: insert into Student (enrollmentID, name, dbID) values (?, ?, ?)
Hibernate: insert into Student (enrollmentID, name, dbID) values (?, ?, ?)
Hibernate: insert into Student (enrollmentID, name, dbID) values (?, ?, ?)
Hibernate: insert into Tutor (name, salary, staffID, id) values (?, ?, ?, ?)
Hibernate: insert into Tutor_Student (Tutor_id, supervisionGroup_dbID) values (?, ?)
Hibernate: insert into Tutor_Student (Tutor_id, supervisionGroup_dbID) values (?, ?)
Hibernate: insert into Tutor_Student (Tutor_id, supervisionGroup_dbID) values (?, ?)
```

	DBID	ENROLLMENTID	NAME	NUM_COURSES
1	1	<null>	Studek Studowski	<null>
2	2	<null>	Stutek Stucinski	<null>
3	3	<null>	Stutka Stucinska	<null>

	TUTOR_ID	SUPERVISIONGROUP_DBID
1	4	1
2	4	2
3	4	3

	ID	NAME	SALARY	STAFFID
1	4	Tutek Tutkowski	5000	TutekStaffID

```java
public static void main(String[] args) {
        sessionFactory = getSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();

        Tutor tutor = session.get(Tutor.class,4);

        Set<Student> supervisedStudents = tutor.getSupervisionGroup();
        for (Student s: supervisedStudents) {
            System.out.println(s);
        }

        tx.commit();
        session.close();
}
```

Próba dostępu generuje następującą sekwencję wywołań….

Hibernate: select tutor0_.id as id1_1_0_, tutor0_.name as name2_1_0_, tutor0_.salary as salary3_1_0_, tutor0_.staffID as staffID4_1_0_ from Tutor tutor0_ where tutor0_.id=?

Hibernate: select supervisio0_.Tutor_id as Tutor_id1_2_0_, supervisio0_.supervisionGroup_dbID as supervis2_2_0_, student1_.dbID as dbID1_0_1_, student1_.enrollmentID as enrollme2_0_1_, student1_.name as name3_0_1_
from Tutor_Student supervisio0_ inner join Student student1_ on supervisio0_.supervisionGroup_dbID=student1_.dbID where supervisio0_.Tutor_id=?

Student{name='Studek Studowski'}
Student{name='Stutek Stucinski'}
Student{name='Stutka Stucinska'}

Czy w poprzednim potrzebna tabela łącznikowa? Oczywiście Nie choć podnosi się że w jej zapisach lepiej widać kolekcje i ich przynalezności:

	TUTOR_ID	SUPERVISIONGROUP_DBID
1	4	1
2	4	2
3	4	3

Jeśli chcemy „klasycznie” adnotacje `@OneToMany` uzupełniamy o `@JoinColumn`

```
@OneToMany
@JoinColumn(name="TUTOR_FK")
private Set<Student> supervisionGroup;
```

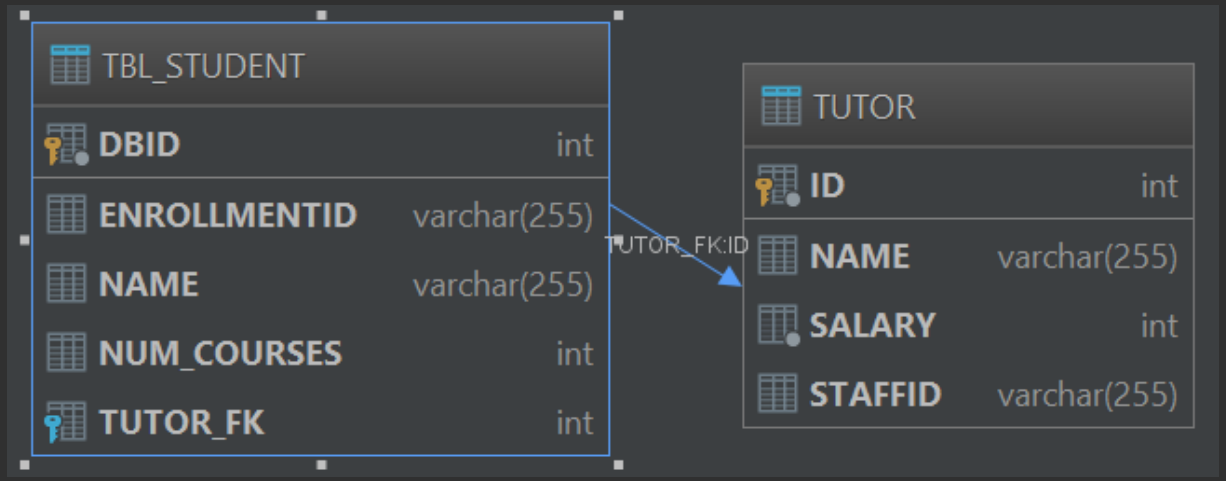
Wówczas próba uruchomienia generuje sekwencje wywołań:

Hibernate: create table TBL_STUDENT (dbID integer not null, enrollmentID varchar(255), name varchar(255), NUM_COURSES integer, TUTOR_FK integer, primary key (dbID))

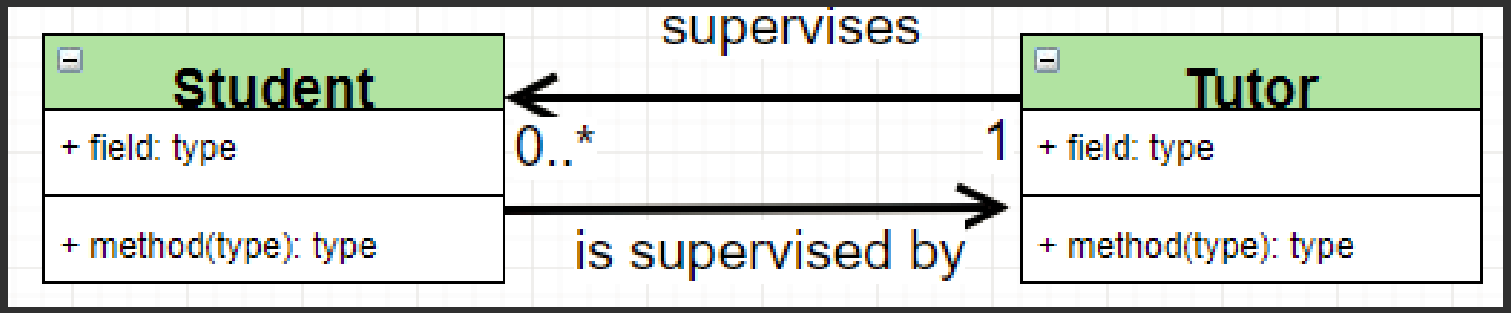
Hibernate: create table Tutor (id integer not null, name varchar(255), salary integer not null, staffID varchar(255), primary key (id))

Hibernate: alter table TBL_STUDENT add constraint FK4xqpn8eux11y7s80kammixhey foreign key (TUTOR_FK) references Tutor

...i po stronie BD otrzymujemy….



...a gdybyśmy chcieli...



Oczywiście możliwe pamiętając, że:

Daje większą elastyczność ale trzeba być bardzo skrupulatnym i ostrożnym w pilnowaniu relacji po stronie Javy

...jak łatwo się domyślić, łączymy poprzednie rozwiązania:

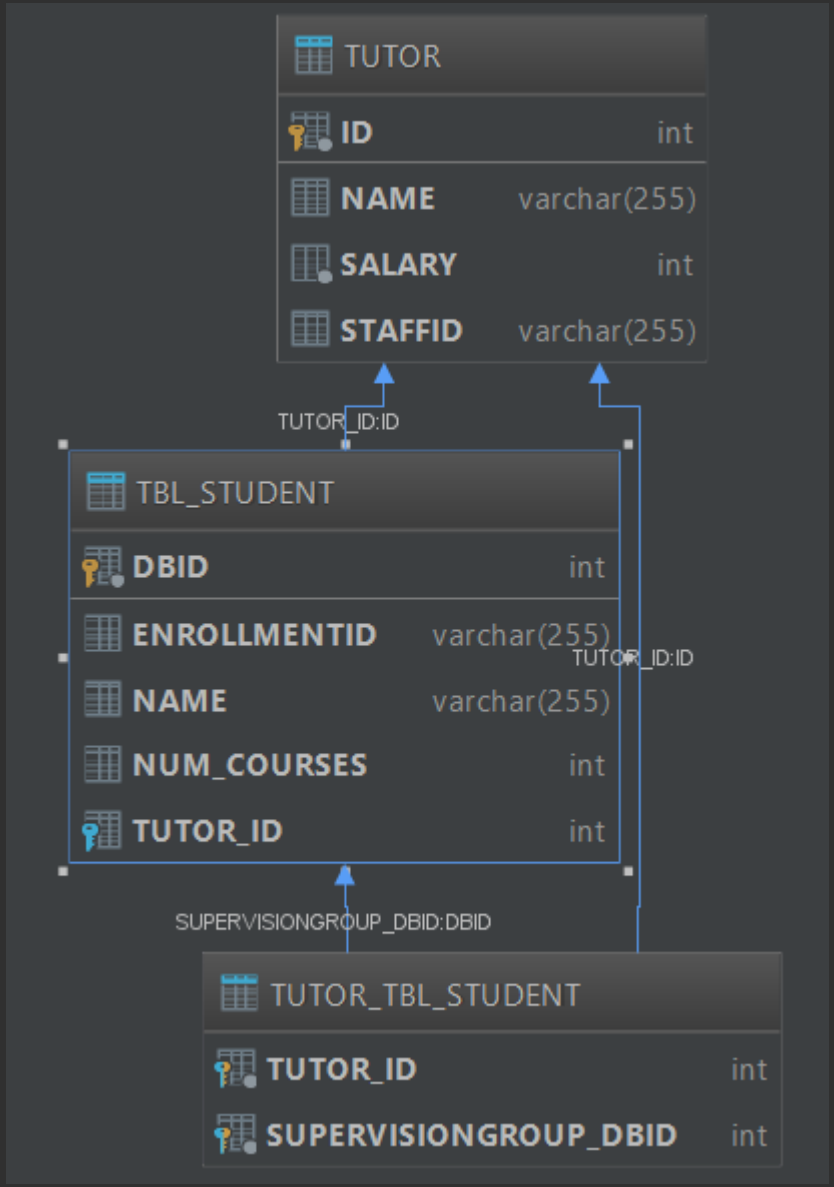
```
// in Student Class
@ManyToOne
private Tutor tutor;

….and in Tutor class:
@OneToMany
private Set<Student> supervisionGroup;
```

...co skutkuje następującą sekwencją wywołań….

Hibernate: create table TBL_STUDENT (
dbID integer not null, enrollmentID varchar(255),
name varchar(255), NUM_COURSES integer,
tutor_id integer, primary key (dbID))

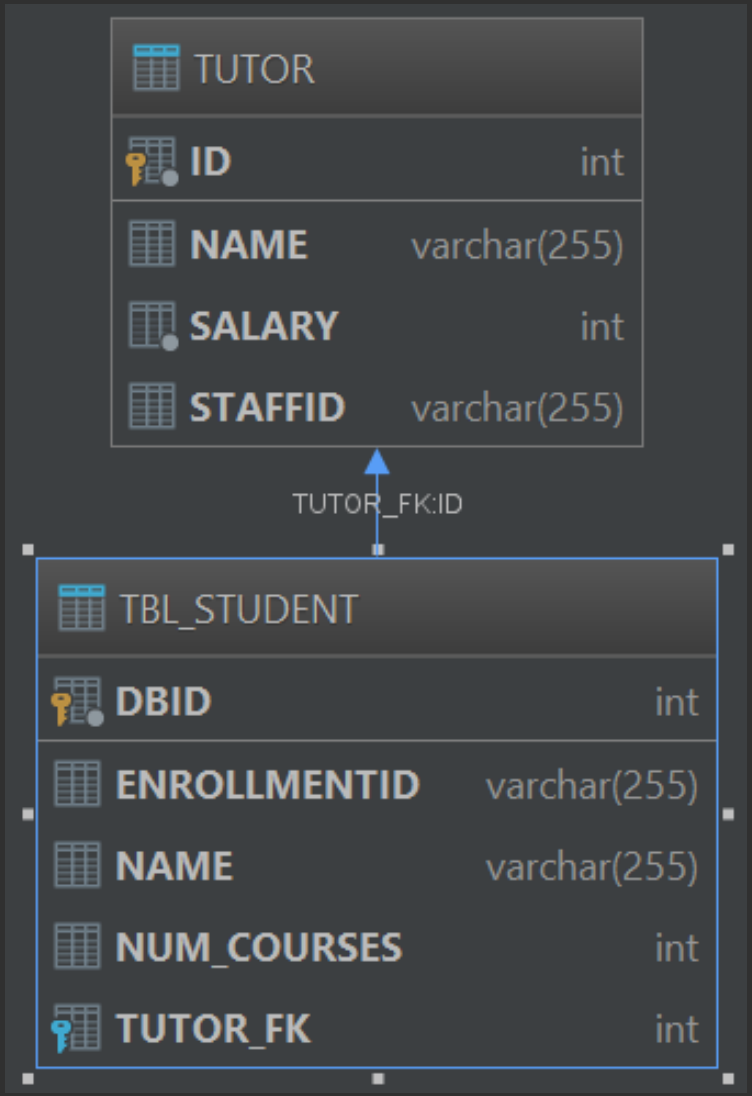
Hibernate: create table Tutor (id integer not null,
name varchar(255), salary integer not null,
staffID varchar(255), primary key (id))

Hibernate: create table Tutor_TBL_STUDENT (
Tutor_id integer not null,
supervisionGroup_dbID integer not null,
primary key (Tutor_id, supervisionGroup_dbID))



…. lub w wersji „klasycznej”

```
In Student class
@ManyToOne
@JoinColumn(name="TUTOR_FK")
private Tutor tutor;

… and in Tutor class
@OneToMany
@JoinColumn(name="TUTOR_FK")
private Set<Student> supervisionGroup;
```



**Hibernate: create table TBL_STUDENT (
dbID integer not null, enrollmentID varchar(255),
name varchar(255), NUM_COURSES integer,
TUTOR_FK integer, primary key (dbID))**

**Hibernate: create table Tutor (id integer not null,
name varchar(255), salary integer not null,
staffID varchar(255), primary key (id))**

**Hibernate: alter table TBL_STUDENT
add constraint FK4xqpn8eux11y7s80kammixhey
foreign key (TUTOR_FK) references Tutor**

```
public static void main(String[] args) {
    Student student1 = new Student("Studek Studowski");
    Student student2 = new Student("Stutek Stucinski");
    Student student3 = new Student("Stutka Stucinska");
    session.save(student1);
    session.save(student2);
    session.save(student3);

    Tutor tutor = new Tutor("TutekStaffID","Tutek
Tutkowski",5000);
    session.save(tutor);

    tutor.addStudentToSupervisionGroup(student1);
    tutor.addStudentToSupervisionGroup(student2);
    tutor.addStudentToSupervisionGroup(student3);
```

Hibernate: insert into TBL_STUDENT (enrollmentID, name, NUM_COURSES, TUTOR_FK, dbID) values (?, ?, ?, ?, ?)

Hibernate: insert into TBL_STUDENT (enrollmentID, name, NUM_COURSES, TUTOR_FK, dbID) values (?, ?, ?, ?, ?)

Hibernate: insert into TBL_STUDENT (enrollmentID, name, NUM_COURSES, TUTOR_FK, dbID) values (?, ?, ?, ?, ?)

Hibernate: insert into Tutor (name, salary, staffID, id) values (?, ?, ?, ?)
……dotąd niby wszystko ok, ale……

Hibernate: update TBL_STUDENT set TUTOR_FK=? where dbID=?
Hibernate: update TBL_STUDENT set TUTOR_FK=? where dbID=?
Hibernate: update TBL_STUDENT set TUTOR_FK=? where dbID=?

….efekt niby taki jak chcemy…..

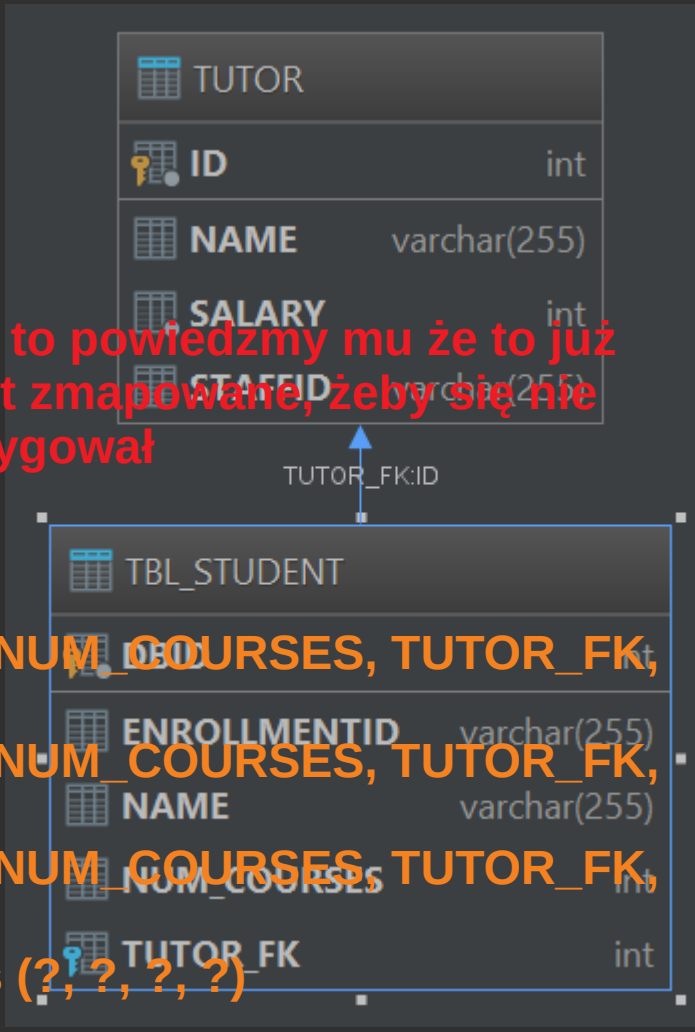
	ID	NAME	SALARY	STAFFID
1	4	Tutek Tutkowski	5000	TutekStaffID

	DBID	ENROLLMENTID	NAME	NUM_COURSES	TUTOR_FK
1	1	<null>	Studek Studowski	<null>	4
2	2	<null>	Stutek Stucinski	<null>	4
3	3	<null>	Stutka Stucinska	<null>	4

….ale dwukrotnie ustawiamy de facto to samo……
moze się rozjechać?!

```
In student class
@ManyToOne
@JoinColumn(name="TUTOR_FK")
private Tutor tutor;

...and in Tutor Class
@OneToMany(mappedBy = "tutor")
private Set<Student> supervisionGroup;
```

No to powiedzmy mu że to już jest zmapowane, żeby się nie fatygował



...i niby lepiej.....

Hibernate: insert into TBL_STUDENT (enrollmentID, name, NUM_COURSES, TUTOR_FK, dbID) values (?, ?, ?, ?, ?)
Hibernate: insert into TBL_STUDENT (enrollmentID, name, NUM_COURSES, TUTOR_FK, dbID) values (?, ?, ?, ?, ?)
Hibernate: insert into TBL_STUDENT (enrollmentID, name, NUM_COURSES, TUTOR_FK, dbID) values (?, ?, ?, ?, ?)
Hibernate: insert into Tutor (name, salary, staffID, id) values (?, ?, ?, ?)

...ale....

	ID	NAME	SALARY	STAFFID
1	4	Tutus Tutowski	50000	50300

	DBID	ENROLLMENTID	NAME	TUTOR_FK
1	1	<null>	Tubby Dubby	<null>
2	2	<null>	Ricky Ticky	<null>
3	3	<null>	Tweeky Sweety	<null>

...powód jest bardzo prosty.....

```
Set<Student> supervisedStudents = newTutor.getSupervisionGroup();
for (Student s: supervisedStudents) {
    System.out.println(s+ " but tutor reference in student is: "+
s.getTutor());
}
```

Student{name='Studek Studowski'}but tutor reference in student is null
Student{name='Stutka Stucinska'}but tutor reference in student is null
Student{name='Stutek Stucinski'}but tutor reference in student is null

...musimy sami zadbać o ustawienie tej referencji.....

```
student1.setTutor(newTutor);
student2.setTutor(newTutor);
student3.setTutor(newTutor);

Set<Student> supervisedStuden
newTutor.getSupervisionGroup();
for (Student s: supervisedStudents) {
    System.out.println(s+ " but tutor reference in student is:
"+ s.getTutor());
}
```

	ID	NAME	SALARY	STAFFID
1	1	Tutus Tutowski	50000	50300

	DBID	ENROLLMENTID	NAME	TUTOR_FK
1	2	<null>	Tubby Dubby	1
2	3	<null>	Ricky Ticky	1
3	4	<null>	Tweeky Sweety	1

....referencje w oczywisty sposób lepiej.....

Student{name='Studek Studowski'}but tutor reference in student is Tutor{name='Tutek Tutkowski'}
Student{name='Stutka Stucinska'}but tutor reference in student is Tutor{name='Tutek Tutkowski'}
Student{name='Stutek Stucinski'}but tutor reference in student is Tutor{name='Tutek Tutkowski'}

...a w kontekście bazy i wywołań hibernate'a...

```
ibernate: insert into Tutor (name, salary, staffID, id) values (?, ?, ?, ?)
Hibernate: insert into Student (enrollmentID, name, TUTOR_FK, dbID) values (?, ?, ?, ?)
Hibernate: insert into Student (enrollmentID, name, TUTOR_FK, dbID) values (?, ?, ?, ?)
Hibernate: insert into Student (enrollmentID, name, TUTOR_FK, dbID) values (?, ?, ?, ?)
Hibernate: update Student set enrollmentID=?, name=?, TUTOR_FK=? where dbID=?
Hibernate: update Student set enrollmentID=?, name=?, TUTOR_FK=? where dbID=?
Hibernate: update Student set enrollmentID=?, name=?, TUTOR_FK=? where dbID=?
```

	ID	NAME	SALARY	STAFFID
1	4	Tutek Tutkowski	5000	TutekStaffID

	DBID	ENROLLMENTID	NAME	NUM_COURSES	TUTOR_FK
1	1	<null>	Studek Studowski	<null>	4
2	2	<null>	Stutek Stucinski	<null>	4
3	3	<null>	Stutka Stucinska	<null>	4

```
In Tutor class
@OneToMany
@JoinColumn(name="TUTOR_FK")
private Set<Student> supervisionGroup;

...and in Student class
//     @OneToMany(mappedBy = "tutor")
    @OneToMany
    @JoinColumn(name = "TUTOR_FK")
    private Set<Student> supervisionGroup;
```

A gdyby zostawić z JoinColumn bez mappedBy....

```
Hibernate: insert into Tutor (name, salary, staffID, id) values (?, ?, ?, ?)
Hibernate: insert into Student (enrollmentID, name, TUTOR_FK, dbID) values (?, ?, ?, ?)
Hibernate: insert into Student (enrollmentID, name, TUTOR_FK, dbID) values (?, ?, ?, ?)
Hibernate: insert into Student (enrollmentID, name, TUTOR_FK, dbID) values (?, ?, ?, ?)
Hibernate: update Student set enrollmentID=?, name=?, TUTOR_FK=? where dbID=?
Hibernate: update Student set enrollmentID=?, name=?, TUTOR_FK=? where dbID=?
Hibernate: update Student set enrollmentID=?, name=?, TUTOR_FK=? where dbID=?
Hibernate: update Student set TUTOR_FK=? where dbID=?
Hibernate: update Student set TUTOR_FK=? where dbID=?
Hibernate: update Student set TUTOR_FK=? where dbID=?
```

….żeby się nie rozjechało – jak chcesz mieć relacje dwukierunkową zawsze kiedy ustawiasz referencje ustawiaj ją od razu w dwie strony….

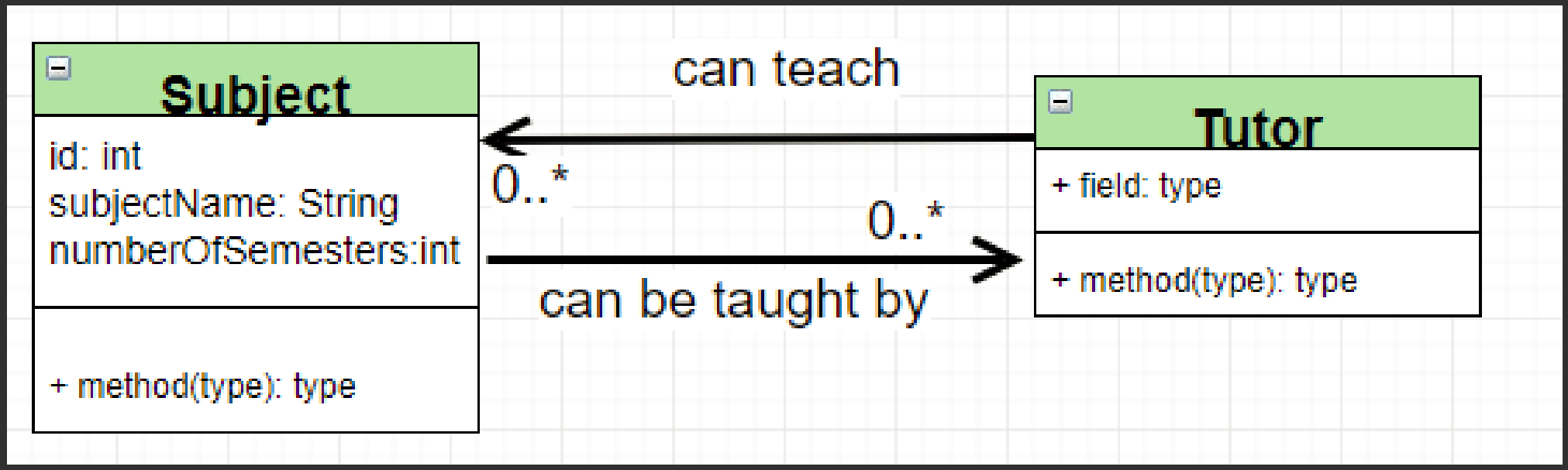
```
public void addStudentToSupervisionGroup(Student studentToAdd){
    this.supervisionGroup.add(studentToAdd);
    studentToAdd.setTutor(this);
}
```

```
public void setTutor(Tutor tutor) {
    this.tutor = tutor;
    this.tutor.getSupervisionGroup().add(this);
}
```

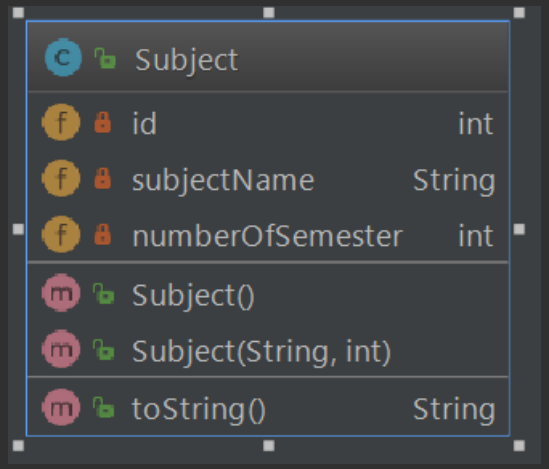
...a gdybyśmy chcieli wiele-do-wielu...

- Żeby zamodelować użyć adnotacji @ManyToMany na obu kolekcjach tworzących relacje wiele-do-wielu
- Dopisek myppedBy możesz dodać po dowolnej stronie relacji – ale tylko po jednej
- Zawsze zostanie stworzona tabela łącznikowa ale oczywiście nie potrzebujesz klasy łącznikowej

...no więc gdybyśmy chcieli...



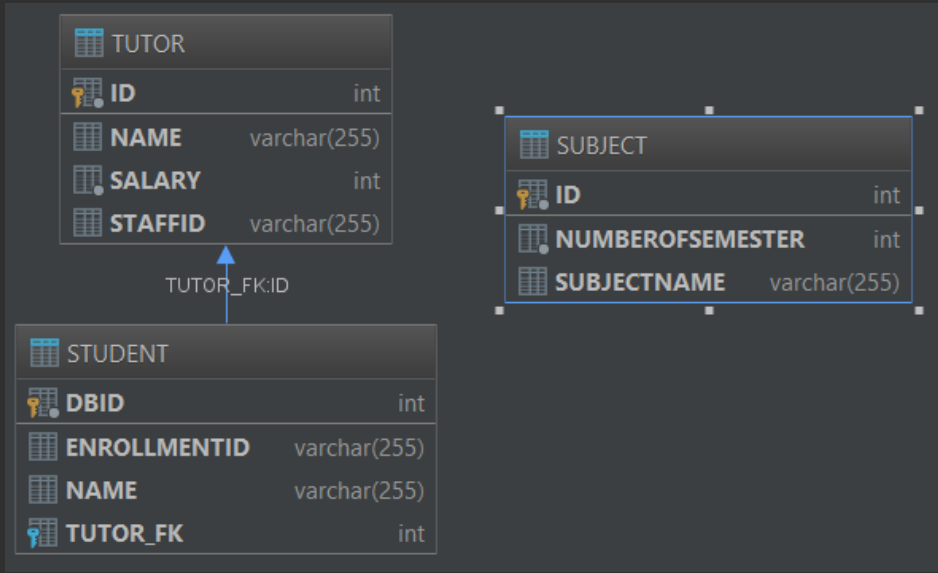
```java
@Entity
public class Subject {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String subjectName;
    private int numberOfSemester;

    //Required by Hibernate
    public Subject() {
    }
    //    business constructor and logic here...
}
```




```java
<mapping class="Domain.Subject"></mapping>
```

```java
Subject math = new Subject("Math",3);
Subject science = new
Subject("science",2);
session.save(math);
session.save(science);
```

	ID	NUMBEROFSEMESTER	SUBJECTNAME
1	5	3	Math
2	6	2	science



...po dodaniu odpowiednich pól i adnotacji....

```
In subject class..
@ManyToMany
Set<Tutor> qualifiedTutors;

...and in tutor class
@ManyToMany(mappedBy = "qualifiedTutors")
private Set<Subject> subjectsQualifiedToTeach;
```

Hibernate: create table Student (dbID integer not null, enrollmentID varchar(255), name varchar(255), TUTOR_FK integer, primary key (dbID))

Hibernate: create table Subject (id integer not null, numberOfSemester integer not null, subjectName varchar(255), primary key (id))

Hibernate: create table Subject_Tutor (subjectsQualifiedToTeach_id integer not null, qualifiedTutors_id integer not null, primary key (subjectsQualifiedToTeach_id, qualifiedTutors_id))

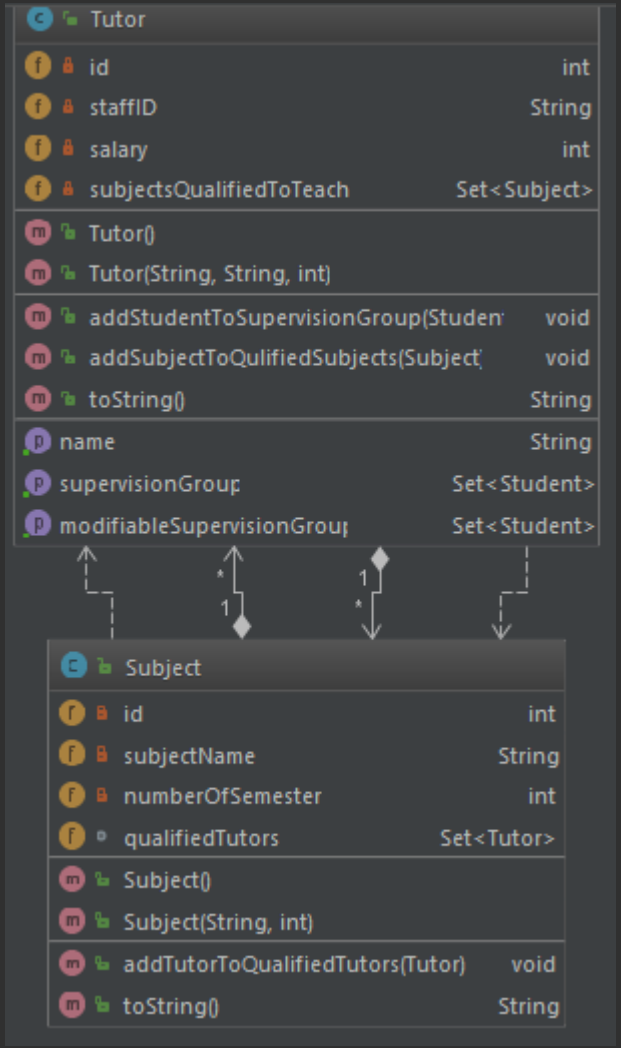
Hibernate: create table Tutor (id integer not null, name varchar(255), salary integer not null, staffID varchar(255), primary key (id))

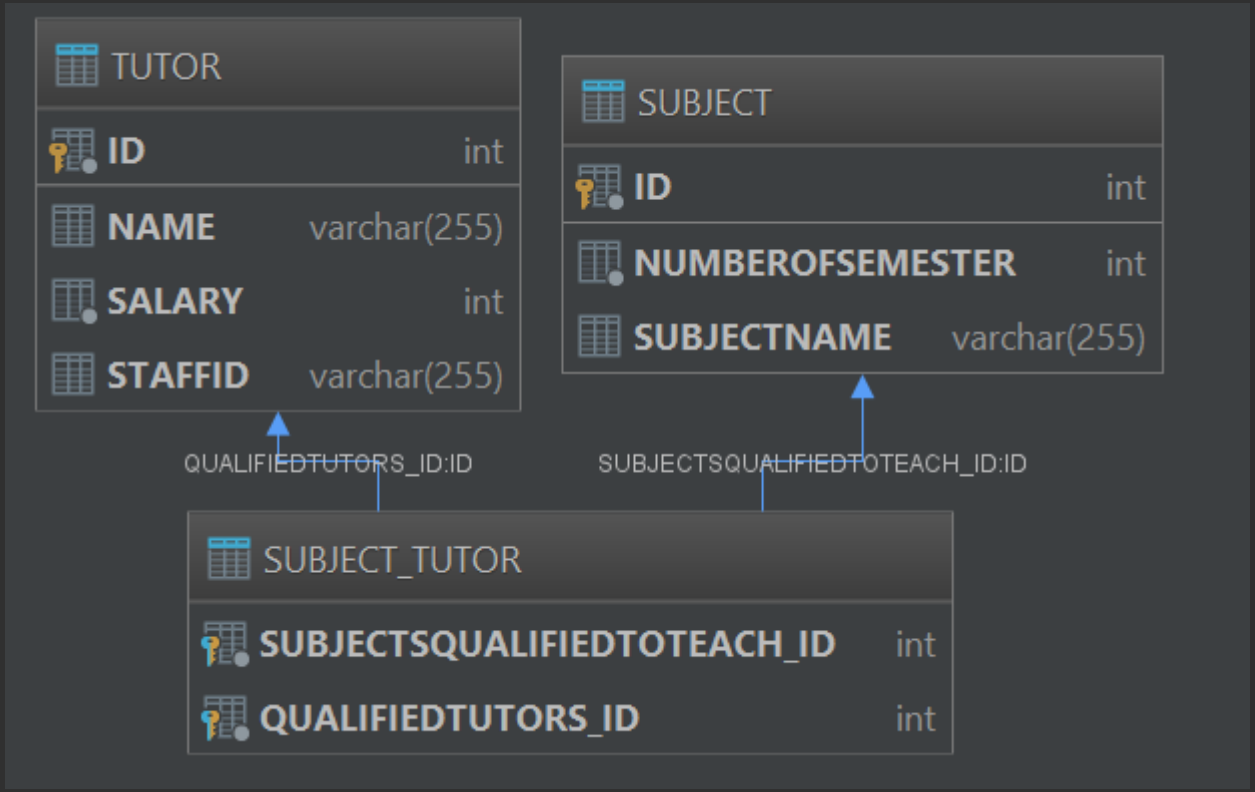
Hibernate: alter table Student add constraint FKq1yaj7ncvhjg8hru60nxl179w foreign key (TUTOR_FK) references Tutor

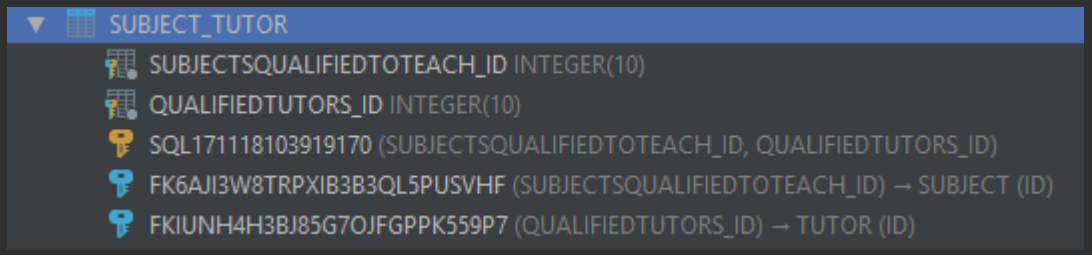
Hibernate: alter table Subject_Tutor add constraint FKiunh4h3bj85g7ojfgppk559p7 foreign key (qualifiedTutors_id) references Tutor

Hibernate: alter table Subject_Tutor add constraint FK6aji3w8trpxib3b3ql5pusvhf foreign key (subjectsQualifiedToTeach_id) references Subject

….i dostajemy następujące mapowanie….







Hibernate vs JPA

- Hibernate stworzony/zapoczątkowany przez pojedynczą osobę (Gavin King) co zawsze budzi kontrowersje
- Jak nie byłby popularny **Nie** jest oficjalnym standardem
- JPA jest oficjalnym standardem
- Dowolny vendor (firma, osoba) może dostarczyć swoją/kolejną implementację standardu
- Jako użytkownicy możemy wybrać dowolną z implementacji stosownie do własnych potrzeb, uwarunkowań politycznych, biznesowych etc.

JPA - (krótka) historia

The final release date of the JPA 1.0 specification was 11 May **2006** as part of Java Community Process JSR 220.

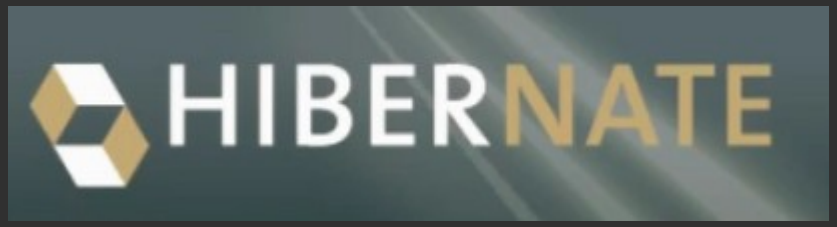
JPA 2.0[edit]
Development of a new version of JPA 2.0 was started in July 2007 in the Java Community Process as JSR 317. JPA 2.0 was approved as final on 10 December **2009**. The focus of JPA 2.0 was to address features that were present in some of the popular ORM vendors, but could not gain consensus approval for JPA 1.0.

Development of a new version of JPA 2.1 was started in July 2011 as JSR 338. JPA 2.1 was approved as final on 22 May **2013**.

Development of a maintenance release as JPA 2.2 was started in 2017 under JSR 338. The maintenance review was approved on 19 Jun, **2017**.
Main features included were:

Add @Repeatable to all relevant annotations
Allow all JPA annotations to be used in meta-annotations.
Add ability to stream a query result
Allow AttributeConverters to be CDI injectable
Support Java 8 Date and Time types

JPA – wybrane implementacje



Vendors supporting JPA 2.0:

- Batoo JPA
- DataNucleus (formerly JPOX)
- EclipseLink (formerly Oracle TopLink)
- IBM, for WebSphere Application Server[11]
- JBoss with Hibernate
- Kundera
- ObjectDB
- OpenJPA
- OrientDB from Orient Technologies
- Versant Corporation JPA (not relational, object database)[12]

Vendors supporting JPA 2.1

- DataNucleus
- EclipseLink
- Hibernate

Vendors supporting JPA 2.2

- DataNucleus

Hibernate vs JPA

- JPA jest łudząco podobny do idei i modelu realizowanego przez „klasyczny” Hibernate
  - Troche różnią się nazwy klas i metod ale
  - Model i koncepcja są takie same
- Session to teraz EntityManager
- Nazwy metod troche zmienione np..
  - save() to teraz persist()
  - Get() to teraz find()
  - Delete() to teraz remove()
- Delikatnie zmieniono plik konfiguracyjny (nazwa, format)
- Adnotacje znane z kalsycznego Hibernate stały się w zasadzie cześcią JPA

```xml
<?xml version="1.0"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
             version="2.0">
    <persistence-unit name="myDatabaseConfig"
                      transaction-type="RESOURCE_LOCAL">
        <properties>
            <property name="hibernate.connection.driver_class"
value="org.apache.derby.jdbc.ClientDriver"/>
            <property name="hibernate.connection.url"
value="jdbc:derby://localhost/MyDatabase"/>

            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.format_sql" value="true" />
            <property name="hibernate.hbm2ddl.auto" value="create" />
        </properties>
    </persistence-unit>
</persistence>
```

...persistence.xml.. (JPA config)

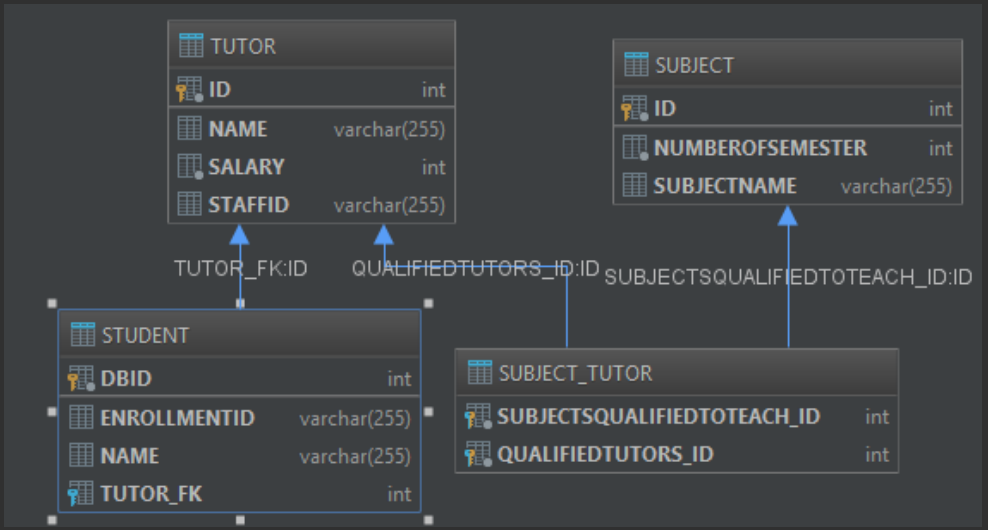
```xml
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property
name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property name="connection.url">jdbc:derby://localhost/MyDatabase</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">create</property>
        <mapping class="Domain.Student"></mapping>
    </session-factory>
</hibernate-configuration>
```

...hibernate.cfg.xml.. („klasczny” config)

EntityManager

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public static void main(String[] args) {

   EntityManagerFactory emf = Persistence.
         createEntityManagerFactory("myDatabaseConfig");
     EntityManager em = emf.createEntityManager();
     EntityTransaction etx = em.getTransaction();
     etx.begin();

   //do something
      Tutor newTutor = new Tutor("50300","Tutus Tutowski",50000);

     etx.commit();
     em.close();
 }
```

...jak to teraz działa...



...ano tak samo...

Hibernate: create table Student (dbID integer not null, enrollmentID varchar(255), name varchar(255), TUTOR_FK integer, primary key (dbID))

Hibernate: create table Subject (id integer not null, numberOfSemester integer not null, subjectName varchar(255), primary key (id))

Hibernate: create table Subject_Tutor (subjectsQualifiedToTeach_id integer not null, qualifiedTutors_id integer not null, primary key (subjectsQualifiedToTeach_id, qualifiedTutors_id))

Hibernate: create table Tutor (id integer not null, name varchar(255), salary integer not nul staffID varchar(255), primary key (id))

Hibernate: alter table Student add constraint FKq1yaj7ncvhjg8hru60nxl179w foreign key (TUTOR_FK) references Tutor
Hibernate: alter table Subject_Tutor add constraint FKiunh4h3bj85g7ojfgppk559p7 foreign key (qualifiedTutors_id) references Tutor
Hibernate: alter table Subject_Tutor add constraint FK6aji3w8trpxib3b3ql5pusvhf foreign key (subjectsQualifiedToTeach_id) references Subject

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

        EntityManagerFactory emf = Persistence.
                createEntityManagerFactory("myDatabaseConfig");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        etx.begin();


        Tutor myTutor = em.find(Tutor.class,1);
        System.out.println(myTutor);
        etx.commit();
        em.close();
    }
```

Hibernate: select tutor0_.id as id1_3_0_, tutor0_.name as name2_3_0_, tutor0_.salary as salary3_3_0_, tutor0_.staffID as staffID4_3_0_ from Tutor tutor0_ where tutor0_.id=?
Tutor{name='Tutus Tutowski'}

```
Student student = em.find(Student.class,2);
em.remove(student);
```

Hibernate: select student0_.dbID as dbID1_0_0_, student0_.enrollmentID as enrollme2_0_0_, student0_.name as name3_0_0_, student0_.TUTOR_FK as TUTOR_FK4_0_0_, tutor1_.id as id1_3_1_, tutor1_.name as name2_3_1_, tutor1_.salary as salary3_3_1_, tutor1_.staffID as staffID4_3_1_
from Student student0_ left outer join Tutor tutor1_ on student0_.TUTOR_FK=tutor1_.id where student0_.dbID=?

Hibernate: delete from Student where dbID=?

Operacje kaskadowe

```
In Tutor class...
@OneToMany(mappedBy = "tutor", cascade = {CascadeType.PERSIST})
private Set<Student> supervisionGroup;
```

...spowoduje, ze przy próbie utrwalenia Tutora kaskadowo zostaną utrwaleni wszyscy nieutrwaleni jeszcze studenci powiązani relacją z tym Tutorem

```
  @OneToMany(mappedBy = "tutor", cascade = CascadeType.PERSIST)
//     @JoinColumn(name="TUTOR_FK")
    private Set<Student> supervisionGroup;


      Student student1 = new Student("Tubby Dubby");
          Student student2 = new Student("Ricky Ticky");
          Student student3 = new Student("Tweeky Sweety");

  //  em.persist(student1);
  //  em.persist(student2);
  //  em.persist(student3);
```

Nie utrwalamy studentów ex plicite….

….ale ponieważ mamy zdefiniowanie utrwalanie kaskadowe…...

Hibernate: insert into Tutor (name, salary, staffID, id) values (?, ?, ?, ?)
Hibernate: insert into Student (enrollmentID, name, TUTOR_FK, dbID) values (?, ?, ?, ?)
Hibernate: insert into Student (enrollmentID, name, TUTOR_FK, dbID) values (?, ?, ?, ?)
Hibernate: insert into Student (enrollmentID, name, TUTOR_FK, dbID) values (?, ?, ?, ?)

	DBID	ENROLLMENTID	NAME	TUTOR_FK
1	2	<null>	Ricky Ticky	1
2	3	<null>	Tubby Dubby	1
3	4	<null>	Tweeky Sweety	1

```
//          em.persist(newTutor);

        Student student1 = new Student("Tubby Dubby");
        Student student2 = new Student("Ricky Ticky");
        Student student3 = new Student("Tweeky Sweety");

        em.persist(student1);
        em.persist(student2);
        em.persist(student3);
```

..a w drugą stronę….?

Nie działa…..

lis 18, 2017 5:38:21 PM org.hibernate.internal.ExceptionMapperStandardImpl mapManagedFlushFailure

ERROR: HHH000346: Error during managed flush [org.hibernate.TransientPropertyValueException: object references an unsaved transient instance - save the transient instance before flushing : Domain.Student.tutor -> Domain.Tutor]

```
@ManyToOne(cascade = CascadeType.PERSIST)
@JoinColumn(name="TUTOR_FK")
private Tutor tutor;
```

Ale jak mu powiemy żeby utrwalał Tuorów kaskadowo…..

Hibernate: insert into Student (enrollmentID, name, TUTOR_FK, dbID) values (?, ?, ?, ?)
Hibernate: insert into Student (enrollmentID, name, TUTOR_FK, dbID) values (?, ?, ?, ?)
Hibernate: insert into Student (enrollmentID, name, TUTOR_FK, dbID) values (?, ?, ?, ?)
Hibernate: insert into Tutor (name, salary, staffID, id) values (?, ?, ?, ?)
Hibernate: update Student set enrollmentID=?, name=?, TUTOR_FK=? where dbID=?
Hibernate: update Student set enrollmentID=?, name=?, TUTOR_FK=? where dbID=?
Hibernate: update Student set enrollmentID=?, name=?, TUTOR_FK=? where dbID=?

Usuwanie kaskadowe

...sytuacja wyjściowa w bazie….

	ID	NAME	SALARY	STAFFID
1	1	Tutus Tutowski	50000	50300

	DBID	ENROLLMENTID	NAME	TUTOR_FK
1	3	<null>	Tubby Dubby	1
2	4	<null>	Tweeky Sweety	1
3	2	<null>	Ricky Ticky	1

….próbując usunąć Tutora…..

```
Tutor foundTutor = em.find(Tutor.class,1);
em.remove(foundTutor);
```

...dostajemy….

ERROR: DELETE on table 'TUTOR' caused a violation of foreign key constraint 'FKQ1YAJ7NCVHJG8HRU60NXL179W' for key (1). The statement has been rolled back.
lis 18, 2017 5:49:01 PM org.hibernate.engine.jdbc.batch.internal.AbstractBatchImpl release

```
@OneToMany(mappedBy = "tutor", cascade =
{CascadeType.PERSIST,CascadeType.REMOVE})
private Set<Student> supervisionGroup;
```

...ale jeśli mu powiemy, żeby usuwał powiązanych studentów kaskadowo….

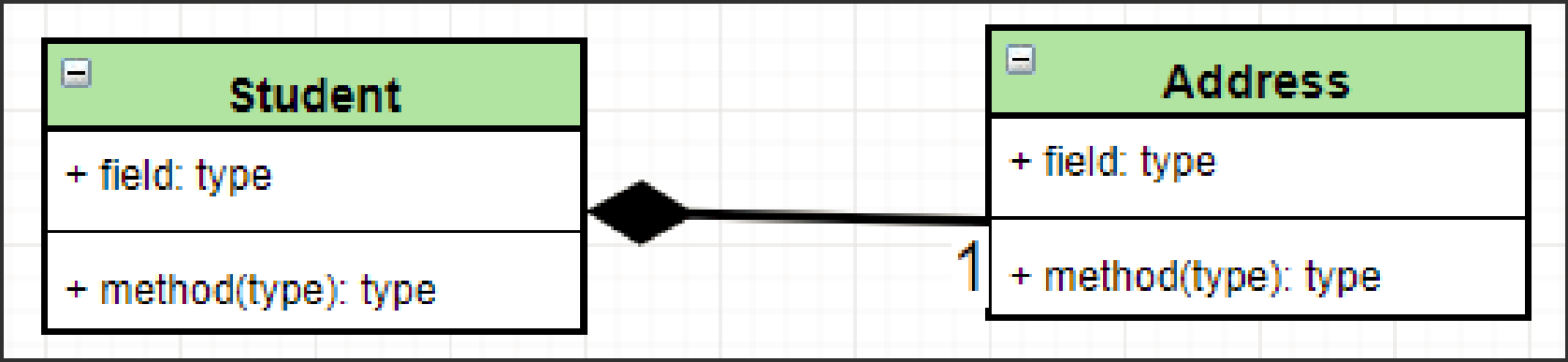
Hibernate: select tutor0_.id as id1_3_0_, tutor0_.name as name2_3_0_, tutor0_.salary as salary3_3_0_, tutor0_.staffID as staffID4_3_0_ from Tutor tutor0_ where tutor0_.id=?

Hibernate: select supervisio0_.TUTOR_FK as TUTOR_FK4_0_0_, supervisio0_.dbID as dbID1_0_0_, supervisio0_.dbID as dbID1_0_1_, supervisio0_.enrollmentID as enrollme2_0_1_, supervisio0_.name as name3_0_1_, supervisio0_.TUTOR_FK as TUTOR_FK4_0_1_ from Student supervisio0_ where supervisio0_.TUTOR_FK=?

Hibernate: delete from Student where dbID=?
Hibernate: delete from Student where dbID=?
Hibernate: delete from Student where dbID=?
Hibernate: delete from Tutor where id=?

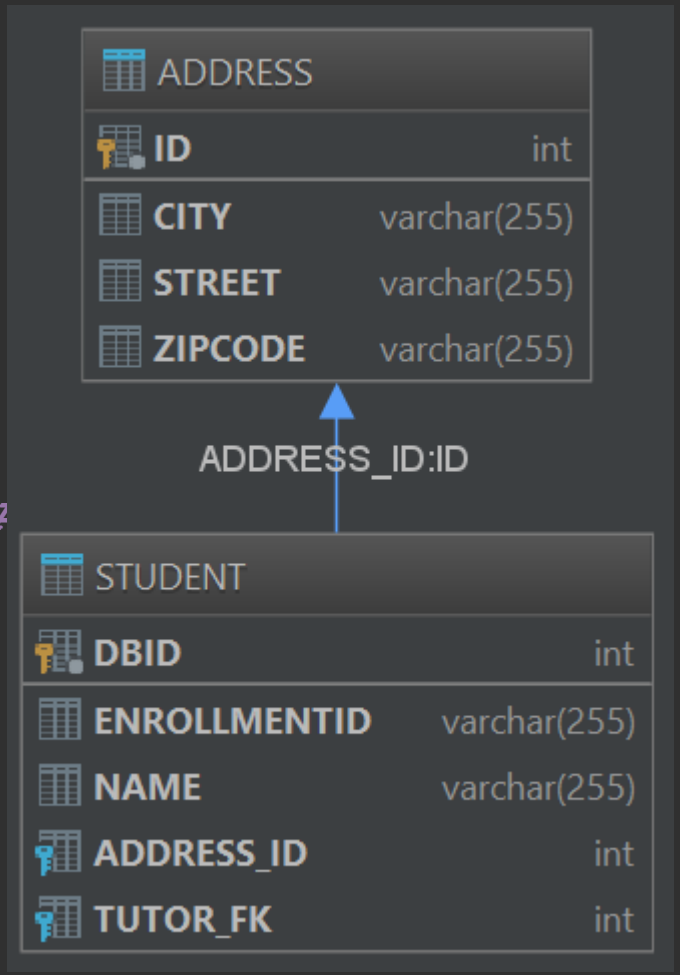
to usuwamy kaskadowo...

...a gdybyśmy chcieli



...oczywiście możemy zrobić to „klasycznie” czyli.....

```
@Entity
public class Address {
    @Id
    @GeneratedValue(strategy = GenerationType.A
    private int id;
    private String street;
    private String city;
    private String zipCode;

 In student class
 @OneToOne(cascade = CascadeType.PERSIST)
 private Address address;
```



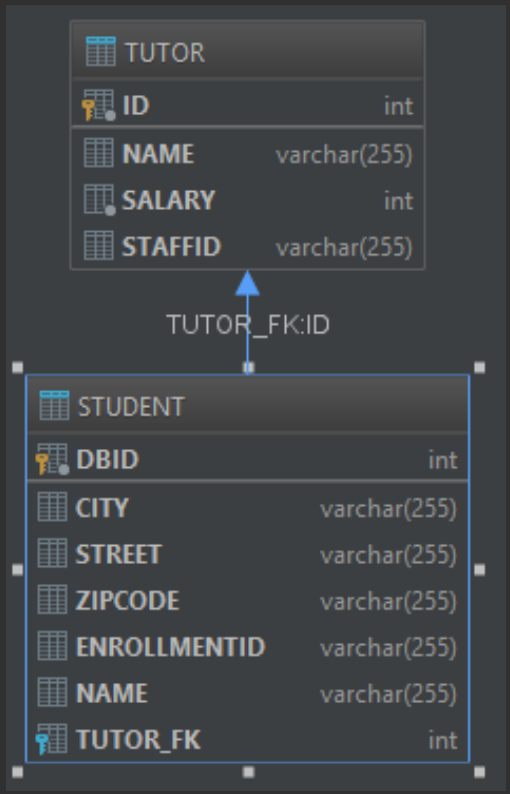
	DBID	ENROLLMENTID	NAME	ADDRESS_ID	TUTOR_FK
1	1	Rys-Mys-2017	Rysio Mysio	2	<null>

	ID	CITY	STREET	ZIPCODE
1	2	Mysioow	Rysiowa	20-059

….albo mozemy „wbudować” adres do tabeli studentów….

```
@Embeddable
public class
Address {
```

```
@Embedded
private Address address;
```



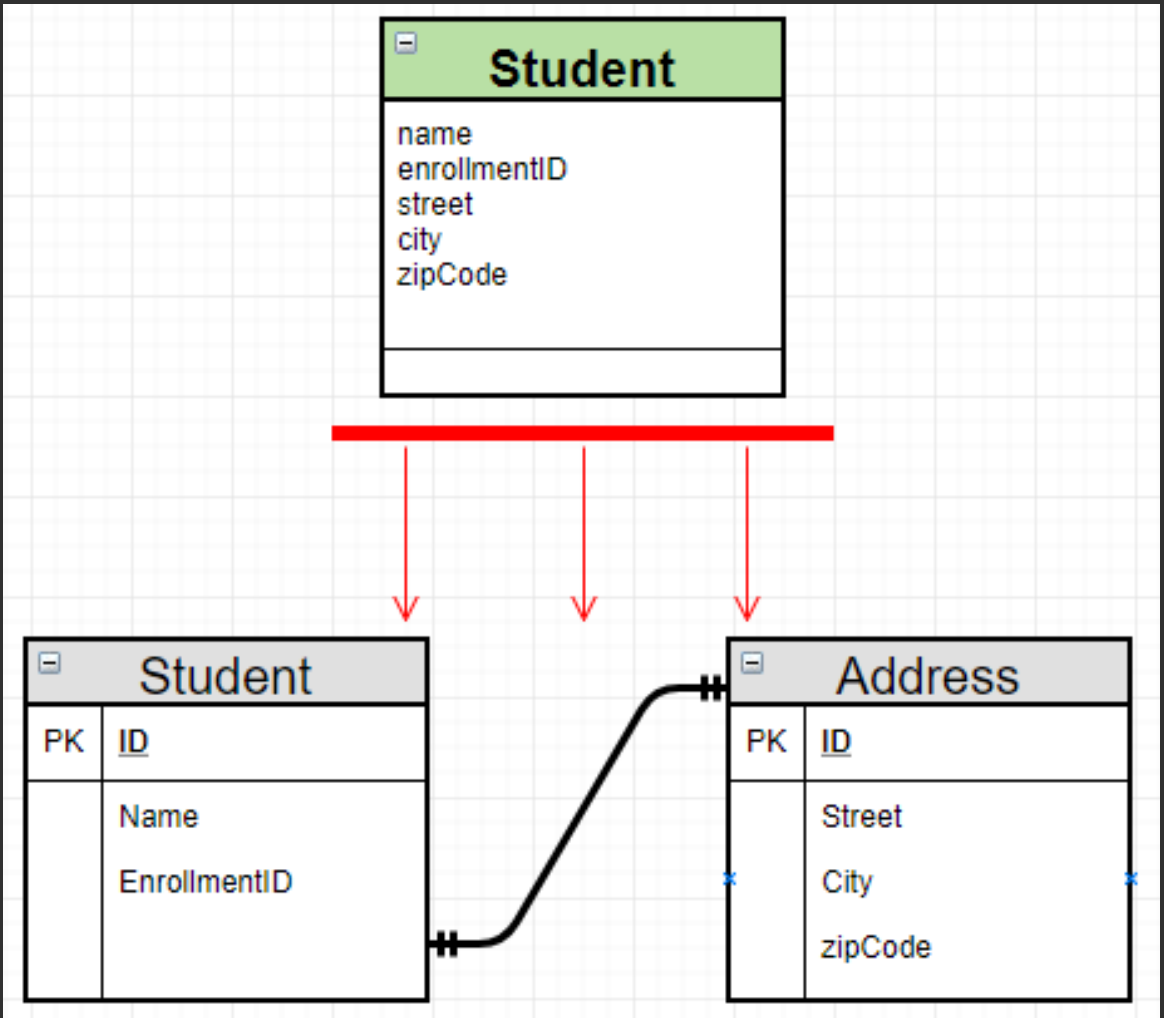
Hibernate: create table Student (dbID integer not null, city varchar(255), street varchar(255) zipCode varchar(255), enrollmentID varchar(255), name varchar(255), TUTOR_FK integer, primary key (dbID))

Hibernate: insert into Student (city, street, zipCode, enrollmentID, name, TUTOR_FK, dbID) values (?, ?, ?, ?, ?, ?, ?)

	DBID	CITY	STREET	ZIPCODE	ENROLLMENTID	NAME	TUTOR_FK
1	1	Mysioow	Rysiowa	20-059	Rys-Mys-2017	Rysio Mysio	<null>

a gdybyśmy chcieli...

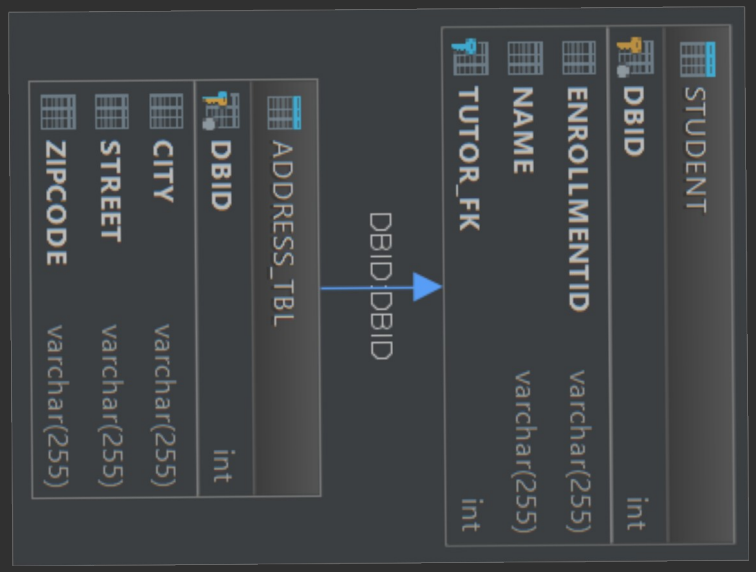
...w drugą strone: 1 klasa mapowana do dwóch tabel….

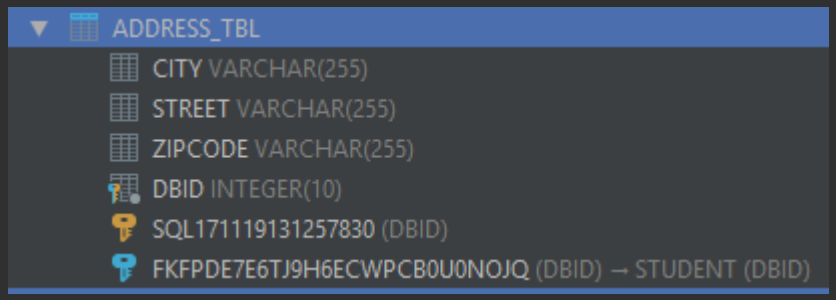


```java
@Entity
@SecondaryTable(name="ADDRESS_TBL")
public class Student {
//Required by Hibernate
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int dbID;
    @Column(table = "ADDRESS_TBL")
    private String street;
    @Column(table = "ADDRESS_TBL")
    private String city;
    @Column(table = "ADDRESS_TBL")
    private String zipCode;


    public Student(String name, String enrollmentID,
                   String street, String city, String zipCode)
    {
        this.name = name;
        this.enrollmentID = enrollmentID;
        this.street = street;
        this.city = city;
        this.zipCode = zipCode;
    }
```

Hibernate: create table Student (dbID integer not null, enrollmentID varchar(255), name varchar(255), TUTOR_FK integer, primary key (dbID))

Hibernate: alter table ADDRESS_TBL add constraint FKfpde7e6tj9h6ecwpcb0u0nojq foreign key (dbID) references Student





Hibernate: select student0_.dbID as dbID1_1_0_, student0_.enrollmentID as enrollme2_1_0_, student0_.name as name3_1_0_, student0_.TUTOR_FK as TUTOR_FK4_1_0_, student0_1_.city as city1_0_0_, student0_1_.street as street2_0_0_, student0_1_.zipCode as zipCode3_0_0_, tutor1_.id as id1_4_1_, tutor1_.name as name2_4_1_, tutor1_.salary as salary3_4_1_, tutor1_.staffID as staffID4_4_1_ from Student student0_ left outer join ADDRESS_TBL student0_1_ on student0_.dbID=student0_1_.dbID left outer join Tutor tutor1_ on student0_.TUTOR_FK=tutor1_.id where student0_.dbID=?

Hibernate: insert into Student (enrollmentID, name, TUTOR_FK, dbID) values (?, ?, ?, ?)
Hibernate: insert into ADDRESS_TBL (city, street, zipCode, dbID) values (?, ?, ?, ?)

	DBID	ENROLLMENTID	NAME	TUTOR_FK
1	1	Rys-Mys-2017	Rysio Mysio	<null>

	CITY	STREET	ZIPCODE	DBID
1	Mysioow	Rysiowa	20-059	1

```
Student studentFromDatabase = em.find(Student.class,1);
System.out.println(studentFromDatabase);
```

Hibernate: select student0_.dbID as dbID1_1_0_, student0_.enrollmentID as enrollme2_1_0_, student0_.name as name3_1_0_, student0_.TUTOR_FK as TUTOR_FK4_1_0_, student0_1_.city as city1_0_0_, student0_1_.street as street2_0_0_, student0_1_.zipCode as zipCode3_0_0_, tutor1_.id as id1_4_1_, tutor1_.name as name2_4_1_, tutor1_.salary as salary3_4_1_, tutor1_.staffID as staffID4_4_1_
from Student student0_ left outer join ADDRESS_TBL student0_1_
on student0_.dbID=student0_1_.dbID left outer join Tutor tutor1_ on student0_.TUTOR_FK=tutor1_.id where student0_.dbID=?

Student{enrollmentID='Rys-Mys-2017', name='Rysio Mysio', street='Rysiowa', city='Mysioow', zipCode='20-059'}

HQL

	DBID	ENROLLMENTID	NAME	TUTOR_FK
1	8	EN-1-2010	Studus Studowski	4
2	7	2-GRA-2009	Studek Studynski	4
3	9	3-PER-2009	Stutka Stutkowska	6

```
Query q = em.createQuery("from Student ");
List<Student> allStudents = q.getResultList();
for (Student stu: allStudents) {
    System.out.println(stu);
}
```

Hibernate: select student0_.dbID as dbID1_1_, student0_.enrollmentID as enrollme2_1_, student0_.name as name3_1_, student0_.TUTOR_FK as TUTOR_FK4_1_, student0_1_.city as city1_0_, student0_1_.street as street2_0_, student0_1_.zipCode as zipCode3_0_ from Student student0_ left outer join ADDRESS_TBL student0_1_ on student0_.dbID=student0_1_.dbID

Hibernate: select tutor0_.id as id1_4_0_, tutor0_.name as name2_4_0_, tutor0_.salary as salary3_4_0_, tutor0_.staffID as staffID4_4_0_ from Tutor tutor0_ where tutor0_.id=?
Hibernate: select tutor0_.id as id1_4_0_, tutor0_.name as name2_4_0_, tutor0_.salary as salary3_4_0_, tutor0_.staffID as staffID4_4_0_
from Tutor tutor0_ where tutor0_.id=?
Student{enrollmentID='EN-1-2010', name='Studus Studowski', street='ul. Stutek 1', city='Miastutko', zipCode='484848'}
Student{enrollmentID='2-GRA-2009', name='Studek Studynski', street='ul. Stutka 14', city='Wiestutka', zipCode='939393'}
Student{enrollmentID='3-PER-2009', name='Stutka Stutkowska', street='pl. Studusia 16', city='Studusiowo', zipCode='939393'}

```
TypedQuery<Student> q = em.createQuery("from
Student",Student.class);
List<Student> allStudents = q.getResultList();
for (Student stu: allStudents) {
    System.out.println(stu);
}
```

```
Hibernate: select student0_.dbID as dbID1_1_, student0_.enrollmentID as enrollme2_1_,
student0_.name as name3_1_, student0_.TUTOR_FK as TUTOR_FK4_1_, student0_1_.city
as city1_0_, student0_1_.street as street2_0_, student0_1_.zipCode as zipCode3_0_
from Student student0_ left outer join ADDRESS_TBL student0_1_ on
student0_.dbID=student0_1_.dbID

Hibernate: select tutor0_.id as id1_4_0_, tutor0_.name as name2_4_0_, tutor0_.salary as
salary3_4_0_, tutor0_.staffID as staffID4_4_0_ from Tutor tutor0_ where tutor0_.id=?
Hibernate: select tutor0_.id as id1_4_0_, tutor0_.name as name2_4_0_, tutor0_.salary as
salary3_4_0_, tutor0_.staffID as staffID4_4_0_ from Tutor tutor0_ where tutor0_.id=?

Student{enrollmentID='EN-1-2010', name='Studus Studowski', street='ul. Stutek 1',
city='Miastutko', zipCode='484848'}
Student{enrollmentID='2-GRA-2009', name='Studek Studynski', street='ul. Stutka 14',
city='Wiestutka', zipCode='939393'}
Student{enrollmentID='3-PER-2009', name='Stutka Stutkowska', street='pl. Studusia 16',
city='Studusiowo', zipCode='939393'}
```

```
TypedQuery<Student> q = em.createQuery("from Student as student"
      + " where student.name='Studus Studowski'",Student.class);
List<Student> allStudents = q.getResultList();
for (Student stu: allStudents) {
    System.out.println(stu);
}
```

```
Hibernate: select student0_.dbID as dbID1_1_, student0_.enrollmentID as enrollme2_1_,
student0_.name as name3_1_, student0_.TUTOR_FK as TUTOR_FK4_1_,
student0_1_.city as city1_0_, student0_1_.street as street2_0_, student0_1_.zipCode as
zipCode3_0_
from Student student0_ left outer join ADDRESS_TBL student0_1_ on
student0_.dbID=student0_1_.dbID where student0_.name='Studus Studowski'

Hibernate: select tutor0_.id as id1_4_0_, tutor0_.name as name2_4_0_, tutor0_.salary as
salary3_4_0_, tutor0_.staffID as staffID4_4_0_ from Tutor tutor0_ where tutor0_.id=?

Student{enrollmentID='EN-1-2010', name='Studus Studowski', street='ul. Stutek 1',
city='Miastutko', zipCode='484848'}
```

```
TypedQuery<Student> q = em.createQuery("from Student as student"
            + " where student.name like 'Stud%'",Student.class);
List<Student> allStudents = q.getResultList();
for (Student stu: allStudents) {
    System.out.println(stu);
}
```

```
Hibernate: select student0_.dbID as dbID1_1_, student0_.enrollmentID as enrollme2_1_,
student0_.name as name3_1_, student0_.TUTOR_FK as TUTOR_FK4_1_, student0_1_.city
as city1_0_, student0_1_.street as street2_0_, student0_1_.zipCode as zipCode3_0_ from
Student student0_ left outer join ADDRESS_TBL student0_1_ on
student0_.dbID=student0_1_.dbID where student0_.name like 'Stud%'

Hibernate: select tutor0_.id as id1_4_0_, tutor0_.name as name2_4_0_, tutor0_.salary as
salary3_4_0_, tutor0_.staffID as staffID4_4_0_ from Tutor tutor0_ where tutor0_.id=?

Student{enrollmentID='EN-1-2010', name='Studus Studowski', street='ul. Stutek 1',
city='Miastutko', zipCode='484848'}
Student{enrollmentID='2-GRA-2009', name='Studek Studynski', street='ul. Stutka 14',
city='Wiestutka', zipCode='939393'}
```

```
TypedQuery<Student> q = em.createQuery("from Student as student"
   + " where student.enrollmentID='EN-1-2010'", Student.class);

Student foundStudent = q.getSingleResult();
System.out.println(foundStudent);
```

```
Hibernate: select student0_.dbID as dbID1_1_, student0_.enrollmentID as enrollme2_1_,
student0_.name as name3_1_, student0_.TUTOR_FK as TUTOR_FK4_1_, student0_1_.city
as city1_0_, student0_1_.street as street2_0_, student0_1_.zipCode as zipCode3_0_
from Student student0_ left outer join ADDRESS_TBL student0_1_ on
student0_.dbID=student0_1_.dbID where student0_.enrollmentID='EN-1-2010'

Hibernate: select tutor0_.id as id1_4_0_, tutor0_.name as name2_4_0_, tutor0_.salary as
salary3_4_0_, tutor0_.staffID as staffID4_4_0_ from Tutor tutor0_ where tutor0_.id=?

Student{enrollmentID='EN-1-2010', name='Studus Studowski', street='ul. Stutek 1',
city='Miastutko', zipCode='484848'}
```

```
String requiredName = "Studus Studowski";
Query q = em.createQuery("from Student as student where
student.name='"
+requiredName+"'");
List<Student> result = q.getResultList();
```

Tak niedobrze

- **Raz: bajzel**
- **Dwa SQLInjection**

Tak dobrze

```
String requiredStudentName = "Studus Studowski";
TypedQuery<Student> q = em.createQuery("from Student as student"
+ " where lower(student.name)=:studentName", Student.class);

q.setParameter("studentName", requiredStudentName);

List<Student> allStudents = q.getResultList();
for (Student stu: allStudents) {
    System.out.println(stu);
}
```

```
Hibernate: select student0_.dbID as dbID1_1_, student0_.enrollmentID as enrollme2_1_,
student0_.name as name3_1_, student0_.TUTOR_FK as TUTOR_FK4_1_, student0_1_.city
as city1_0_, student0_1_.street as street2_0_, student0_1_.zipCode as zipCode3_0_
from Student student0_ left outer join ADDRESS_TBL student0_1_ on
student0_.dbID=student0_1_.dbID where student0_.name=?

Hibernate: select tutor0_.id as id1_4_0_, tutor0_.name as name2_4_0_, tutor0_.salary as
salary3_4_0_, tutor0_.staffID as staffID4_4_0_ from Tutor tutor0_ where tutor0_.id=?

Student{enrollmentID='EN-1-2010', name='Studus Studowski', street='ul. Stutek 1',
city='Miastutko', zipCode='484848'}
```

	ID	NAME	SALARY	STAFFID
1	4	Tutus Tutowski	2939393	ABC123
2	5	Tutek Tutynski	0	DEF456
3	6	Tutka Tutawska	0	GHI678

	DBID	ENROLLMENTID	NAME	TUTOR_FK
1	8	EN-1-2010	Studus Studowski	4
2	7	2-GRA-2009	Studek Studynski	4
3	9	3-PER-2009	Stutka Stutkowska	6

```
Tutor tutor = em.find(Tutor.class,4);

 TypedQuery<Student> q = em.createQuery("from Student as student" +
         " where student.tutor=:tutor", Student.class);
 q.setParameter("tutor", tutor);
```

Hibernate: select tutor0_.id as id1_4_0_, tutor0_.name as name2_4_0_, tutor0_.salary as salary3_4_0_, tutor0_.staffID as staffID4_4_0_ from Tutor tutor0_ where tutor0_.id=?

Hibernate: select student0_.dbID as dbID1_1_, student0_.enrollmentID as enrollme2_1_, student0_.name as name3_1_, student0_.TUTOR_FK as TUTOR_FK4_1_, student0_1_.city as city1_0_, student0_1_.street as street2_0_, student0_1_.zipCode as zipCode3_0_ from Student student0_ left outer join ADDRESS_TBL student0_1_ on student0_.dbID=student0_1_.dbID where student0_.TUTOR_FK=?

Student{enrollmentID='EN-1-2010', name='Studus Studowski', street='ul. Stutek 1', city='Miastutko', zipCode='484848'}
Student{enrollmentID='2-GRA-2009', name='Studek Studynski', street='ul. Stutka 14', city='Wiestutka', zipCode='939393'}

```
TypedQuery<Student> q = em.createQuery("from Student as
student" +
        " where student.tutor.name=:name", Student.class);
q.setParameter("name", "Tutus Tutowski");
```

Hibernate: select student0_.dbID as dbID1_1_, student0_.enrollmentID as enrollme2_1_, student0_.name as name3_1_, student0_.TUTOR_FK as TUTOR_FK4_1_, student0_1_.city as city1_0_, student0_1_.street as street2_0_, student0_1_.zipCode as zipCode3_0_ from Student student0_ left outer join ADDRESS_TBL student0_1_ on student0_.dbID=student0_1_.dbID, Tutor tutor1_ where student0_.TUTOR_FK=tutor1_.id and tutor1_.name=?

Hibernate: select tutor0_.id as id1_4_0_, tutor0_.name as name2_4_0_, tutor0_.salary as salary3_4_0_, tutor0_.staffID as staffID4_4_0_ from Tutor tutor0_ where tutor0_.id=?

Student{enrollmentID='EN-1-2010', name='Studus Studowski', street='ul. Stutek 1', city='Miastutko', zipCode='484848'}
Student{enrollmentID='2-GRA-2009', name='Studek Studynski', street='ul. Stutka 14', city='Wiestutka', zipCode='939393'}

```
TypedQuery<Tutor> q = em.createQuery("from Tutor as tutor" +
        " where tutor.supervisionGroup is empty ",
Tutor.class);
```

Hibernate: select tutor0_.id as id1_4_, tutor0_.name as name2_4_, tutor0_.salary as salary3_4_, tutor0_.staffID as staffID4_4_ from Tutor tutor0_ where not (exists (select supervisio1_.dbID from Student supervisio1_ where tutor0_.id=supervisio1_.TUTOR_FK))
Tutor{name='Tutek Tutynski'}

```
Subject history = em.find(Subject.class,3);
TypedQuery<Tutor> q = em.createQuery("from Tutor as tutor" +
        " where :subject member of tutor.subjectsQualifiedToTeach
", Tutor.class);
q.setParameter("subject",history);
```

```
Hibernate: select subject0_.id as id1_2_0_, subject0_.numberOfSemester as
numberOf2_2_0_, subject0_.subjectName as subjectN3_2_0_
    from Subject subject0_ where subject0_.id=?

Hibernate: select tutor0_.id as id1_4_, tutor0_.name as name2_4_, tutor0_.salary as
salary3_4_, tutor0_.staffID as staffID4_4_ from Tutor tutor0_
where ? in (select subjectsqu1_.subjectsQualifiedToTeach_id from Subject_Tutor
subjectsqu1_ where tutor0_.id=subjectsqu1_.qualifiedTutors_id)

Tutor{name='Tutka Tutawska'}
```

```
Subject science = em.find(Subject.class,2);
TypedQuery<Student> q = em.createQuery("from Student as student" +
        " where :subject member of
student.tutor.subjectsQualifiedToTeach ", Student.class);
q.setParameter("subject",science);
```

```
Hibernate: select subject0_.id as id1_2_0_, subject0_.numberOfSemester as
numberOf2_2_0_, subject0_.subjectName as subjectN3_2_0_ from Subject subject0_ where
subject0_.id=?

Hibernate: select student0_.dbID as dbID1_1_, student0_.enrollmentID as enrollme2_1_,
student0_.name as name3_1_, student0_.TUTOR_FK as TUTOR_FK4_1_, student0_1_.city
as city1_0_, student0_1_.street as street2_0_, student0_1_.zipCode as zipCode3_0_
from Student student0_ left outer join ADDRESS_TBL student0_1_ on
student0_.dbID=student0_1_.dbID, Tutor tutor1_ where student0_.TUTOR_FK=tutor1_.id
and (? in (select subjectsqu2_.subjectsQualifiedToTeach_id from Subject_Tutor
subjectsqu2_ where tutor1_.id=subjectsqu2_.qualifiedTutors_id))

Hibernate: select tutor0_.id as id1_4_0_, tutor0_.name as name2_4_0_, tutor0_.salary as
salary3_4_0_, tutor0_.staffID as staffID4_4_0_ from Tutor tutor0_ where tutor0_.id=?

Student{enrollmentID='EN-1-2010', name='Studus Studowski', street='ul. Stutek 1',
city='Miastutko', zipCode='484848'}
Student{enrollmentID='2-GRA-2009', name='Studek Studynski', street='ul. Stutka 14',
city='Wiestutka', zipCode='939393'}
```

```
Query q = em.createQuery("from Tutor as tutor" +
        " join tutor.supervisionGroup as student where
student.city='Miastutko' ");

List<Object[]> allTutors = q.getResultList();
for (Object[] next: allTutors) {
    System.out.println(next[0]+"------------"+next[1]);
}
```

```
Hibernate: select tutor0_.id as id1_4_0_, supervisio1_.dbID as dbID1_1_1_,
tutor0_.name as name2_4_0_, tutor0_.salary as salary3_4_0_, tutor0_.staffID as
staffID4_4_0_, supervisio1_.enrollmentID as enrollme2_1_1_, supervisio1_.name as
name3_1_1_, supervisio1_.TUTOR_FK as TUTOR_FK4_1_1_, supervisio1_1_.city as
city1_0_1_, supervisio1_1_.street as street2_0_1_, supervisio1_1_.zipCode as
zipCode3_0_1_
from Tutor tutor0_ inner join Student supervisio1_ on
tutor0_.id=supervisio1_.TUTOR_FK
left outer join ADDRESS_TBL supervisio1_1_ on
supervisio1_.dbID=supervisio1_1_.dbID where supervisio1_1_.city='Miastutko'

Tutor{name='Tutus Tutowski'}-----------Student{enrollmentID='EN-1-2010', name='Studus
Studowski', street='ul. Stutek 1', city='Miastutko', zipCode='484848'}
```

```
Query q = em.createQuery("select Tutor from Tutor as tutor" +
        " join tutor.supervisionGroup as student where
student.city='Miastutko' ");

List<Tutor> allTutors = q.getResultList();
for (Tutor next: allTutors) {
    System.out.println(next);
}
```

```
Hibernate: select tutor0_.id as id1_4_, tutor0_.name as name2_4_, tutor0_.salary as
salary3_4_, tutor0_.staffID as staffID4_4_
from Tutor tutor0_ inner join Student supervisio1_ on tutor0_.id=supervisio1_.TUTOR_FK
left outer join ADDRESS_TBL supervisio1_1_ on supervisio1_.dbID=supervisio1_1_.dbID
where supervisio1_1_.city='Miastutko'

Tutor{name='Tutus Tutowski'}
```

FluentAPI

```
String requiredName = "Studus Studowski";
List<Student> results = em.createQuery("from Student as student" +
        " where student.name = :name")
        .setParameter("name",requiredName)
        .getResultList();
```

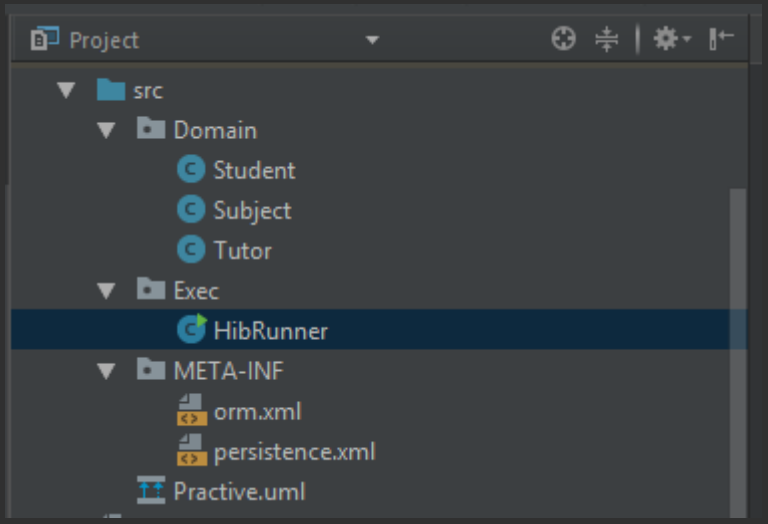
Hibernate: select student0_.dbID as dbID1_1_, student0_.enrollmentID as enrollme2_1_, student0_.name as name3_1_, student0_.TUTOR_FK as TUTOR_FK4_1_, student0_1_.city as city1_0_, student0_1_.street as street2_0_, student0_1_.zipCode as zipCode3_0_ from Student student0_ left outer join ADDRESS_TBL student0_1_ on student0_.dbID=student0_1_.dbID where student0_.name=?

Hibernate: select tutor0_.id as id1_4_0_, tutor0_.name as name2_4_0_, tutor0_.salary as salary3_4_0_, tutor0_.staffID as staffID4_4_0_ from Tutor tutor0_ where tutor0_.id=?

Student{enrollmentID='EN-1-2010', name='Studus Studowski', street='ul. Stutek 1', city='Miastutko', zipCode='484848'}

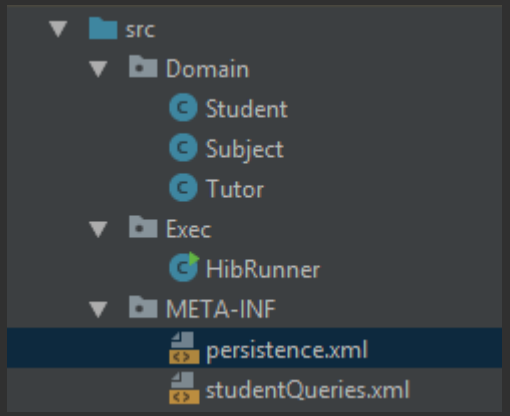
```
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
    http://java.sun.com/xml/ns/persistence/orm_2_0.xsd"
                 version="2.0">

<named-query name="searchByName">
    <query>from Student as student where student.name=:name</query>
</named-query>

</entity-mappings>
```

```
String requiredName = "Studus Studowski";
List<Student> results =
em.createNamedQuery("searchByName")
        .setParameter("name",requiredName)
        .getResultList();
```



```
<persistence-unit name="myDatabaseConfig"
                  transaction-type="RESOURCE_LOCAL">
    <mapping-file>META-INF/studentQueries.xml</mapping-file>
    <properties>
        <property name="hibernate.connection.driver_class"
value="org.apache.derby.jdbc.ClientDriver"/>
```

```
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
    http://java.sun.com/xml/ns/persistence/orm_2_0.xsd"
                 version="2.0">

<named-query name="searchByName">
    <query>from Student as student where student.name=:name</query>
</named-query>

</entity-mappings>
```



	DBID	ENROLLMENTID	NAME	TUTOR_FK
1	8	EN-1-2010	Studus Studowski	4
2	7	2-GRA-2009	Studek Studynski	4
3	9	3-PER-2009	Stutka Stutkowska	6

```
String requiredName = "Studus Studowski";
Long numberOfStudents = (long)em.createQuery
        ("select count(student) from Student as
student").getSingleResult();

System.out.println(numberOfStudents);
```

Hibernate: select count(student0_.dbID) as col_0_0_ from Student student0_ left outer join ADDRESS_TBL student0_1_ on student0_.dbID=student0_1_.dbID

3

Dostępne także max min sum avg

...a gdybyśmy chcieli podwoić wynagrodzenie wszystkim Tutorom....

```
List<Tutor> allTutors = em.createQuery("from Tutor
").getResultList();
for (Tutor next: allTutors) {
    next.doubleSalary();
}
```

```
public void doubleSalary() {
    this.salary = this.salary * 2;
}
```

Hibernate: select tutor0_.id as id1_4_, tutor0_.name as name2_4_, tutor0_.salary as salary3_4_, tutor0_.staffID as staffID4_4_ from Tutor tutor0_

Hibernate: update Tutor set name=?, salary=?, staffID=? where id=?
Hibernate: update Tutor set name=?, salary=?, staffID=? where id=?
Hibernate: update Tutor set name=?, salary=?, staffID=? where id=?

	ID	NAME	SALARY	STAFFID
1	4	Tutus Tutowski	10000	ABC123
2	5	Tutek Tutynski	15000	DEF456
3	6	Tutka Tutawska	20000	GHI678

Możemy pojedynczo tak jak wcześniej, albo „zbiorowo” jak poniżej

```
em.createQuery
        ("update Tutor as tutor set
tutor.salary=tutor.salary*2").executeUpdate();
```

Hibernate: update Tutor set salary=salary*2

	ID	NAME	SALARY	STAFFID
1	4	Tutus Tutowski	40000	ABC123
2	5	Tutek Tutynski	60000	DEF456
3	6	Tutka Tutawska	80000	GHI678

Dostępne także delete oraz insert into

Nie lubisz HQL'a – użyj native SQL'a

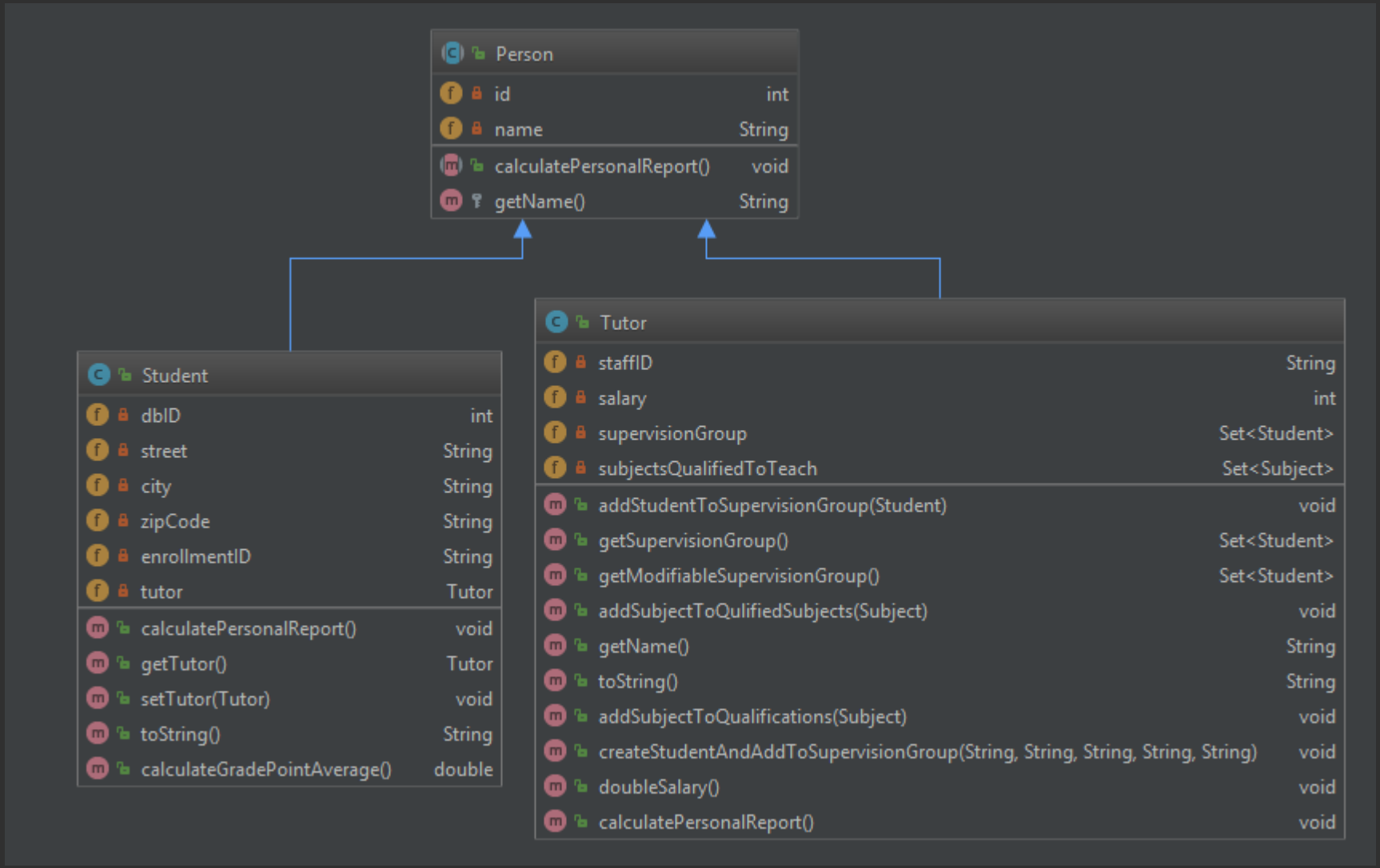
```
List<Object[]> result = em.createNativeQuery(
        "select s.name, s.enrollmentid from student as s")
        .getResultList();
    for (Object[] next: result ) {
        System.out.println(next[0]+"-----"+next[1]);
```

```
Hibernate: select s.name, s.enrollmentid from student as s
Studus Studowski-----EN-1-2010
Studek Studynski-----2-GRA-2009
Stutka Stutkowska-----3-PER-2009
```

...albo JPQL

- Troche inne nazwy metod
- Zapytania niemal identyczne

a gdybyśmy chcieli...



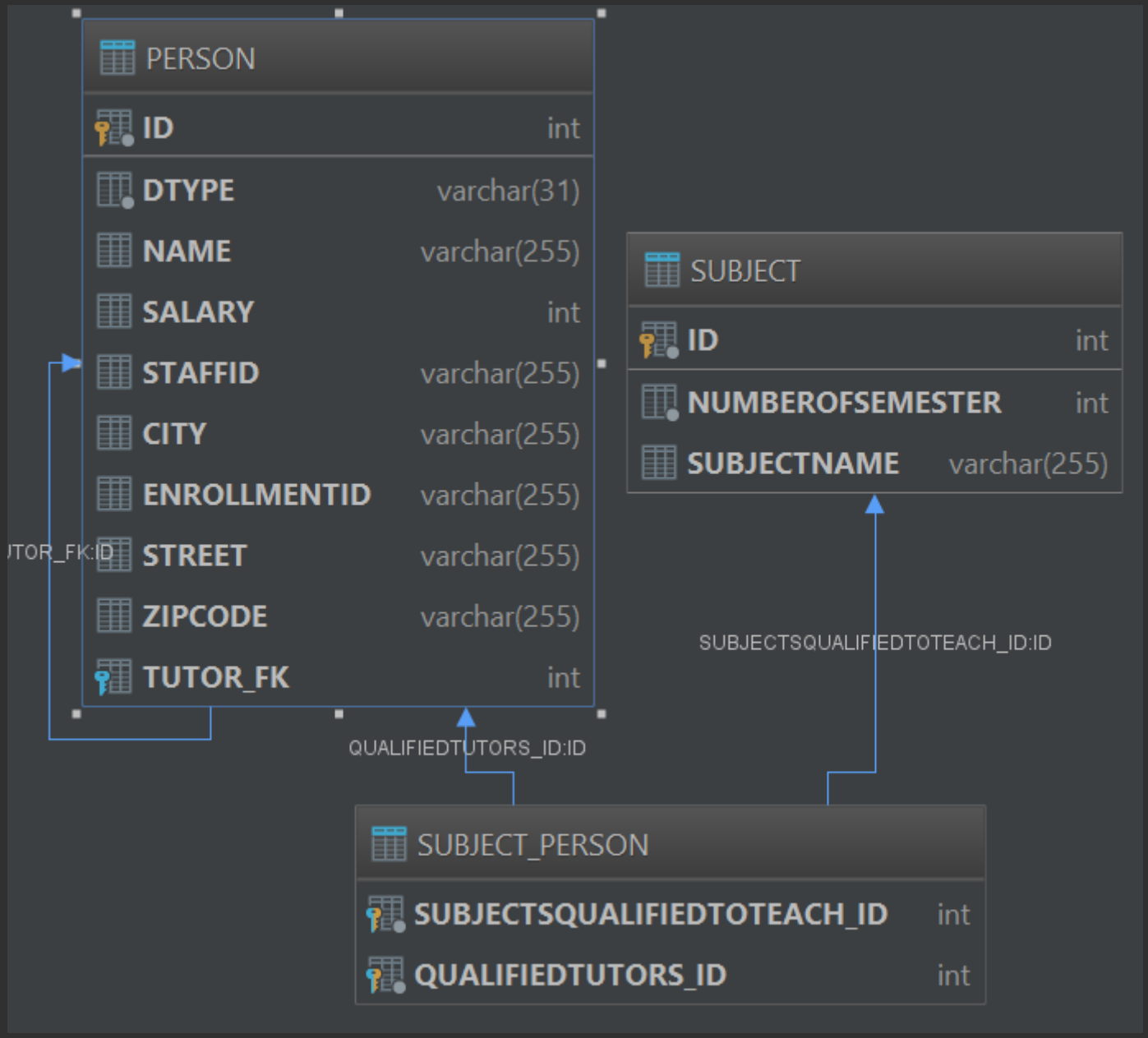
```java
@Entity
@Inheritance(strategy= InheritanceType.SINGLE_TABLE)
public abstract class Person {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    private String name;
    public abstract void calculatePersonalReport();

    public Person(String name) {
        this.name = name;
    }

    public Person() {
    }

    protected String getName(){
        return this.name;
    }
}
```

Jedna tabela na całą hierarchię..

```java
@Entity
public class Student extends Person {
    private String street;
    private String city;
    private String zipCode;
    private String enrollmentID;
    @ManyToOne(cascade = CascadeType.PERSIST)
    @JoinColumn(name = "TUTOR_FK")
    private Tutor tutor;
```

```java
@Entity
public class Tutor extends Person {
    private String staffID;
    private int salary;
    @OneToMany(mappedBy = "tutor", cascade = {CascadeType.PERSIST,CascadeType.REMOVE})
    private Set<Student> supervisionGroup;

    @ManyToMany(mappedBy = "qualifiedTutors")
    private Set<Subject> subjectsQualifiedToTeach;
```

```
Hibernate: create table Person (DTYPE varchar(31) not null, id integer not null, name varchar(255), salary integer, staffID varchar(255), city varchar(255), enrollmentID varchar(255), street varchar(255), zipCode varchar(255), TUTOR_FK integer, primary key (id))

Hibernate: create table Subject (id integer not null, numberOfSemester integer not null, subjectName varchar(255), primary key (id))

Hibernate: create table Subject_Person (subjectsQualifiedToTeach_id integer not null, qualifiedTutors_id integer not null, primary key (subjectsQualifiedToTeach_id, qualifiedTutors_id))

Hibernate: alter table Person add constraint FKgp5cxpfy0lgpvfpmrdbj6xyjo foreign key (TUTOR_FK) references Person

Hibernate: alter table Subject_Person add constraint FKbgkltv9ulqvuvtkdrbqly6iu2 foreign key (qualifiedTutors_id) references Person

Hibernate: alter table Subject_Person add constraint FKesf8o7gn0psq98rg65e25c4qa foreign key (subjectsQualifiedToTeach_id) references Subject
```


PERSON
ID int
DTYPE varchar(31)
NAME varchar(255)
SALARY int
STAFFID varchar(255)
CITY varchar(255)
ENROLLMENTID varchar(255)
STREET varchar(255)
ZIPCODE varchar(255)
TUTOR_FK int
TOR_FK:ID
SUBJECT
ID int
NUMBEROFSEMESTER int
SUBJECTNAME varchar(255)
SUBJECTSQUALIFIEDTOTEACH_ID:ID
QUALIFIEDTUTORS_ID:ID
SUBJECT_PERSON
SUBJECTSQUALIFIEDTOTEACH_ID int
QUALIFIEDTUTORS_ID int

Hibernate: insert into Person (name, city, enrollmentID, street, TUTOR_FK, zipCode, DTYPE, id) values (?, ?, ?, ?, ?, ?, 'Student', ?)

Hibernate: insert into Person (name, salary, staffID, DTYPE, id) values (?, ?, ?, 'Tutor', ?)

	DTYPE	ID	NAME	SALARY	STAFFID	CITY	ENROLLMENTID	STREET	ZIPCODE	TUTOR_FK
1	Student	1	Stutek Studzinski	<null>	<null>	Stutkowo	ENR-1278	Stutkowa 7	345676	<null>
2	Tutor	2	Tutek Tucinski	20000	ST-765	<null>	<null>	<null>	<null>	<null>

```
public class Tutor extends Person {
    @Column(unique = true,nullable = false)
    private String staffID;
```

Caused by: java.sql.SQLIntegrityConstraintViolationException: Column 'STAFFID' cannot accept a NULL value.

```
List<Person> allPeople = em.createQuery("from
Person").getResultList();
for (Person next: allPeople) {
    next.calculatePersonalReport();
}
```

Hibernate: select person0_.id as id2_0_, person0_.name as name3_0_, person0_.salary as salary4_0_, person0_.staffID as staffID5_0_, person0_.city as city6_0_, person0_.enrollmentID as enrollme7_0_, person0_.street as street8_0_, person0_.TUTOR_FK as TUTOR_F10_0_, person0_.zipCode as zipCode9_0_, person0_.DTYPE as DTYPE1_0_ from Person person0_

Calulating report for Student: Stutek Studzinski
Calulating report for Tutor: Tutek Tucinski

```
List<Student> allPeople = em.createQuery("from
Student").getResultList();
for (Student next: allPeople) {
    next.calculatePersonalReport();
}
```

Hibernate: select student0_.id as id2_0_, student0_.name as name3_0_, student0_.city as city6_0_, student0_.enrollmentID as enrollme7_0_, student0_.street as street8_0_, student0_.TUTOR_FK as TUTOR_F10_0_, student0_.zipCode as zipCode9_0_ from Person student0_ where student0_.DTYPE='Student'
Calulating report for Student: Stutek Studzinski

```java
@Entity
@Inheritance(strategy= InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="PERSON_TYPE")
public abstract class Person {
```

	PERSON_TYPE	ID	NAME	SALARY	STAFFID	CITY	ENROLLMENTID	STREET	ZIP
1	Student	1	Stutek Studzinski	<null>	<null>	Stutkowo	ENR-1278	Stutkowa 7	34567
2	Tutor	2	Tutek Tucinski	20000	ST-765	<null>	<null>	<null>	<null

```java
@Entity
@DiscriminatorValue(value = "T")
public class Tutor extends Person {
```

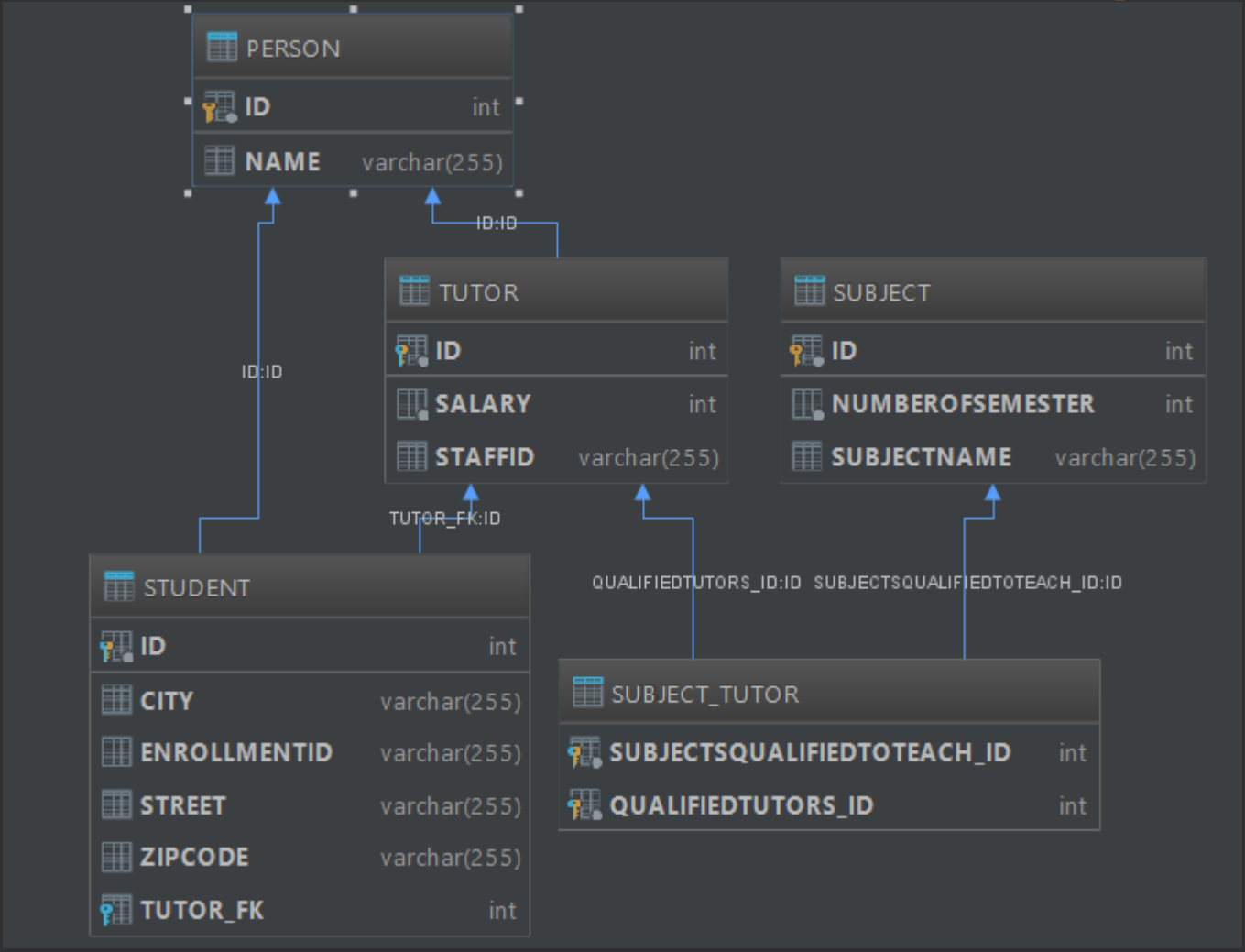
```java
@Entity
@DiscriminatorValue(value = "S")
public class Student extends
Person {
```

	PERSON_TYPE	ID	NAME	SALARY	STAFFID	CITY	ENROLLMENTID	STREET
1	S	5	Stutek Studzinski	<null>	<null>	Stutkowo	ENR-1278	Stutkowa
2	T	6	Tutek Tucinski	20000	ST-765	<null>	<null>	<null>

```
@Entity
@Inheritance(strategy= InheritanceType.JOINED)
public abstract class Person {
```

```
@Entity
public class Student extends Person {
    private String street;

@Entity
public class Tutor extends Person {
    private String staffID;
```

Tabele łączone



```
List<Person> allPeople = em.createQuery("from Person").getResultList();
for (Person next: allPeople) {
    next.calculatePersonalReport();
}
```

```
Hibernate: insert into Person (name, id) values (?, ?)
Hibernate: insert into Student (city, enrollmentID, street, TUTOR_FK, zipCode, id) values
(?, ?, ?, ?, ?, ?)

Hibernate: insert into Person (name, id) values (?, ?)
Hibernate: insert into Tutor (salary, staffID, id) values (?, ?, ?)

Hibernate: select person0_.id as id1_0_, person0_.name as name2_0_, person0_1_.salary a
salary1_4_, person0_1_.staffID as staffID2_4_, person0_2_.city as city1_1_,
person0_2_.enrollmentID as enrollme2_1_, person0_2_.street as street3_1_,
person0_2_.TUTOR_FK as TUTOR_FK6_1_, person0_2_.zipCode as zipCode4_1_,
case
when person0_1_.id is not null then 1
when person0_2_.id is not null then 2
when person0_.id is not null then 0 else -1 end
 as clazz_ from Person person0_ left outer join Tutor person0_1_ on
person0_.id=person0_1_.id left outer join Student person0_2_ on
person0_.id=person0_2_.id

Calulating report for Student: Stutek Studzinski
Calulating report for Tutor: Tutek Tucinski
```

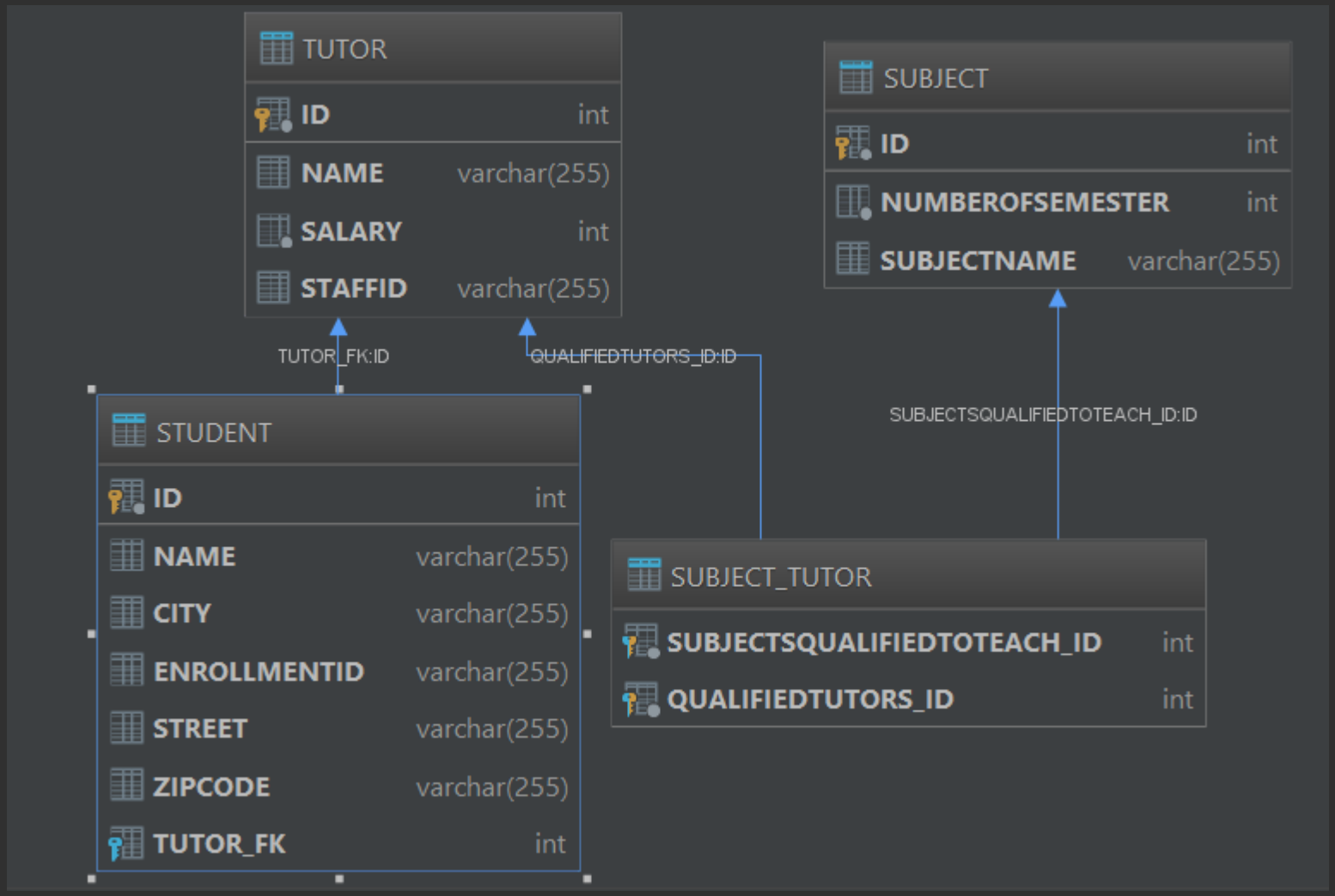
```
List<Student> allPeople = em.createQuery("from
Student").getResultList();
for (Student next: allPeople) {
    next.calculatePersonalReport();
}
```

Hibernate: select student0_.id as id1_0_, student0_1_.name as name2_0_, student0_.city as city1_1_, student0_.enrollmentID as enrollme2_1_, student0_.street as street3_1_, student0_.TUTOR_FK as TUTOR_FK6_1_, student0_.zipCode as zipCode4_1_ from Student student0_ inner join Person student0_1_ on student0_.id=student0_1_.id

Calulating report for Student: Stutek Studzinski

```
@Entity
@Inheritance(strategy= InheritanceType.TABLE_PER_CLASS)
public abstract class Person {
```

Jedna tabela na konkretną klasę



```
List<Person> allPeople = em.createQuery("from
Person").getResultList();
for (Person next: allPeople) {
    next.calculatePersonalReport();
}
```

Hibernate: select person0_.id as id1_0_, person0_.name as name2_0_, person0_.salary as salary1_4_, person0_.staffID as staffID2_4_, person0_.city as city1_1_, person0_.enrollmentID as enrollme2_1_, person0_.street as street3_1_, person0_.TUTOR_FK as TUTOR_FK5_1_, person0_.zipCode as zipCode4_1_, person0_.clazz_ as clazz_
from (select id, name, salary, staffID, nullif('x', 'x') as city, nullif('x', 'x') as enrollmentID, nullif('x', 'x') as street, nullif('x', 'x') as zipCode, nullif(0, 0) as TUTOR_FK, 1 as clazz_
from Tutor union all
select id, name, nullif(0, 0) as salary, nullif('x', 'x') as staffID, city, enrollmentID, street, zipCode, TUTOR_FK, 2 as clazz_ from Student) person0_
Calulating report for Tutor: Tutek Tucinski
Calulating report for Student: Stutek Studzinski

	ID	NAME	CITY	ENROLLMENTID	STREET	ZIPCODE	TUTOR_FK
1	1	Stutek Studzinski	Stutkowo	ENR-1278	Stutkowa 7	345676	<null>

	ID	NAME	SALARY	STAFFID
1	2	Tutek Tucinski	20000	ST-765

Strategie mapowania hierarchii dziedziczenia – za i przeciw

- **Jedna tabela na całą hierarchię**
  - **Niezła w kontekście polimorfizmu**
  - **Niezła w kontekście dostępu do klas dziedziczących**
  - **Dramat w kontekście schematu bazy**

- **Tabele łączone**
  - **Gorsza w kontekście polimorfizmu**
  - **Gorsza w kontekście dostępu do klas dziedziczących**
  - **Zdecydowanie lepiej w kontekście schematu bazy**

- **Jedna tabela per konkretna klasa**
  - **Słaba w kontekście polimorfizmu (unia)**
  - **Niezła w kontekście dostępu do konkretnych**
  - **Słabo w kontekście schematu bazy**

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

    Tutor tutor = new Tutor("ST-765","Tutek Tucinski",20000);
    em.persist(tutor);

    //thinking....thinking....thinking....

    tutor.setName("Tutus Tutowski");
}
```

Jeśli w między czasie inny proces zmieni zawartość rekordu opisującego Tutka Tucinskiego to:

- **Update'ujemy Tutka Tucinskiego czy inną wartość?**
- **Mamy szanse o tym wiedziec?**

Domyślnie:

- **Updatujemy inną wartość (chyba że zamkniemy to w jednej transakcji – co nie jest najlepszym rozwiązaniem)**
- **Nie mamy szansy się o tym dowiedzieć**

..ale...

```
@Entity
@Inheritance(strategy= InheritanceType.TABLE_PER_CLASS)
public abstract class Person {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    @Version
    private int version;
```

Hibernate: create table Student (id integer not null, name varchar(255), version integer not null, enrollmentID varchar(255), TUTOR_FK integer, primary key (id))

ibernate: create table Tutor (id integer not null, name varchar(255), version integer not null, salary integer not null, staffID varchar(255), primary key (id))

Hibernate: insert into Tutor (name, version, salary, staffID, id) values (?, ?, ?, ?, ?)
Hibernate: update Tutor set name=?, version=?, salary=?, staffID=? where id=? and version=?

	ID	NAME	VERSION	SALARY	STAFFID
1	1	Tutek Tucinski	0	20000	ST-765

```
etx = em.getTransaction();
etx.begin();
tutor = em.merge(tutor);
tutor.setName("Tutek Tutynski");
System.out.println("...and here the version is"+tutor.getVersion());
etx.commit();
```

….gdyby po drodze inny proces podbił version na wartość inna niż nam znana…..

Exception in thread "main" javax.persistence.OptimisticLockException: Row was updated or deleted by another transaction (or unsaved-value mapping was incorrect) : [Domain.Tutor#1]
at org.hibernate.internal.ExceptionConverterImpl.wrapStaleStateException(ExceptionConverterImpl.java:202)

…..innymi słowy OptimisticLocking to nie Locking a Versioning….

```java
List<Object[]> tutorNamesAndSalaries = em.createQuery
  ("select tutor.name,tutor.salary from Tutor as tutor")
                                        .getResultList();

    for (Object[] next: tutorNamesAndSalaries) {
        System.out.println(next[0]+"  :  "+next[1]);
    }
        //thinking....thinking....thinking....
      long totalSalary = (Long)em.createQuery
          ("select sum(tutor.salary) from Tutor as tutor")
                                        .getSingleResult();
    System.out.println("Total tutor salary is:  "+totalSalary);
```

	ID	NAME	VERSION	SALARY	STAFFID
1	1	Tutus Tutowski	60	20000	ST-765
2	2	Tutek Tutynski	0	20000	ST-900

Hibernate: select tutor0_.name as col_0_0_, tutor0_.salary as col_1_0_ from Tutor tutor0_
Tutus Tutowski : 20000
Tutek Tutynski : 20000

	ID	NAME	VERSION	SALARY	STAFFID
1	1	Tutus Tutowski	60	0	ST-765
2	2	Tutek Tutynski	0	0	ST-900

Hibernate: select sum(tutor0_.salary) as col_0_0_ from Tutor tutor0_
Total tutor salary is: 0

```
List<Object[]> tutorNamesAndSalaries = em.createQuery("select tutor.name,tutor.salary
                                        " + "from Tutor as tutor")
                                        .getResultList();

    for (Object[] next: tutorNamesAndSalaries) {
        System.out.println(next[0]+"  :  "+next[1]);
    }
    long totalSalary = (Long)em.createQuery("select sum(tutor.salary) from Tutor as
                                        tutor")
                                        .setLockMode(LockModeType.PESSIMISTIC_READ)
                                        .getSingleResult();
    System.out.println("Total tutor salary is:  "+totalSalary);
```

	ID	NAME	VERSION	SALARY	STAFFID
1	1	Tutus Tutowski	60	20000	ST-765
2	2	Tutek Tutynski	0	20000	ST-900

Hibernate: select tutor0_.name as col_0_0_, tutor0_.salary as col_1_0_ from Tutor tutor0_ for read only with rs
Tutus Tutowski : 20000
Tutek Tutynski : 20000

	ID	NAME	VERSION	SALARY	STAFFID
1	1	Tutus Tutowski	60	0	ST-765
2	2	Tutek Tutynski	0	0	ST-900

Processing...

Hibernate: select sum(tutor0_.salary) as col_0_0_ from Tutor tutor0_
Total tutor salary is: 40000