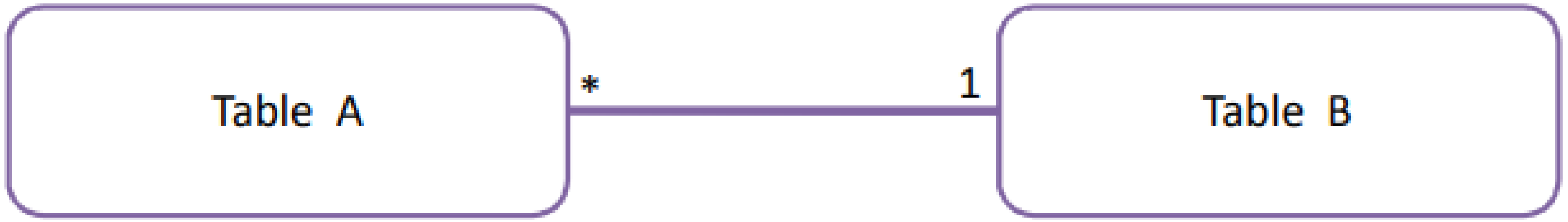
HIBERNATE

- Many-to-One
- Many-to-Many
- Bidirectional mapping
- Use of mappedBy property

Many-to-one mapping



- Table A has many-to-one relationship with Table B
  - If many records of Table A can be linked to a single record of Table B
- Table B has one-to-many relationship with Table A
  - If a single record in Table B is linked with 1..many records of Table A.

Java Mapping data types for Relations: Collection Mapping

- If an entity or a class has collection of values for a particular variable, then we can map those values using any one of the Java collection interfaces.
- Hibernate can persist instances of the following types:
  - java.util.Map,
  - java.util.Set,
  - java.util.SortedMap,
  - java.util.SortedSet,
  - java.util.List, and
  - any array of persistent entities or values.

Collection Mapping

Java Collection type	Mapping tag element (in XML file) and its initialization
java.util.Set	<set> initialized using java.util.HashSet
java.util.SortedSet	<set> Initialized using java.util.TreeSet sort attribute can assume comparator or natural order
java.util.List	<list> initialized using java.util.ArrayList
java.util.Collection	<bag> or <ibag> initialized using java.util.ArrayList
java.util.Map	<map> initialized using java.util.HashMap
java.util.SortedMap	<map> initialized using java.util.TreeMap sort attribute can assume comparator or natural order

Example: registration of participants in some competition

Student

registration_id	student_name	college_id
1	Kamal	1
2	Abhishek	2
3	Radha	3
4	Kisan	3

StudentAddress

college_id	college_address
1	L D Eng. College, Ahmedabad
2	Niram University, Ahmedabad
3	D D University, Nadiad

Not Null column. It has a foreign key constraint with college_id column of StudentAddress table

Configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration
    DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
 <session-factory>
  <property
   name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
  <property
   name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
  <property
   name="hibernate.connection.url">jdbc:mysql://localhost:3306/ddu?zeroDateTime
   Behavior=convertToNull</property>
  <property name="hibernate.connection.username">root</property>
  <mapping class="hibernate.relations.many2one.Student"/>
  <mapping class="hibernate.relations.many2one.StudentAddress"/>
 </session-factory>
</hibernate-configuration>
```

How to create tables?

Two options

- Create tables using SQL queries
- Use hibernate.hbm2ddl.auto property with create as value

Student class

```
@Entity
@Table(name = "STUDENT")
public class Student {
   @Id
   @GeneratedValue
   private int student_id;
   private String student_name;

   public int getStudent_id() {
      return student_id;
   }
   @ManyToOne(cascade=CascadeType.ALL)
   private StudentAddress studentAddress;
```

It tells that many objects (rows) of Student are associated with one object (row) of StudentAddress

Student class

```
  public StudentAddress getStudentAddress() {
      return studentAddress;
   }
   public void setStudentAddress(StudentAddress studentAddress) {
      this.studentAddress = studentAddress;
   }
   public String getStudent_name() {
      return student_name;
   }
   public void setStudent_name(String student_name) {
      this.student_name = student_name;
   }
}
```

StudentAddress class

```
@Entity @Table(name="STUDENTADDRESS")
public class StudentAddress {
   @Id @GeneratedValue
   private int college_id;
   private String college_address;
   public int getCollege_id() {
      return college_id;
   }
 public String getCollege_address() {
      return college_address;
   }
   public void setCollege_address(String college_address) {
      this.college_address = college_address;
   }
}
```

Main class

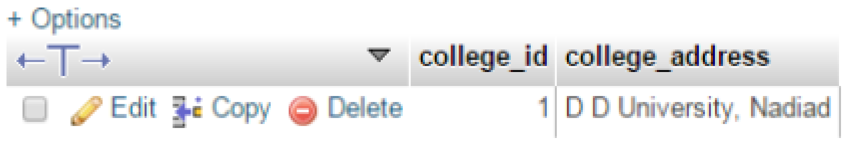
```
package hibernate.relations.many2one;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;
public class Main {
   public static void main(String[] args) {
      StudentAddress studentAddress=new StudentAddress();
      studentAddress.setCollege_address("D D University, Nadiad");

      Student student1=new Student();
      student1.setStudent_name("Kisan");
      student1.setStudentAddress(studentAddress);
```

Main class

```
    Student student2=new Student();
     student2.setStudent_name("Radha");
     student2.setStudentAddress(studentAddress);
SessionFactory sessionFactory= new
   AnnotationConfiguration().configure().buildSessionFactory();
     Session session=sessionFactory.openSession();
     session.beginTransaction();
     session.save(student1);
     session.save(student2);
     session.getTransaction().commit();
     session.close();
     sessionFactory.close();
  }
}
```

Records in the tables

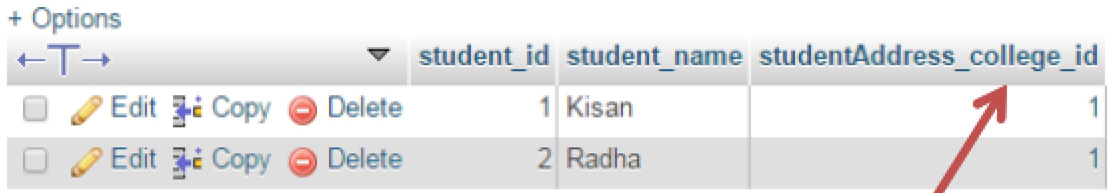
- STUDENTADDRESS Table:



+ Options

	college_id	college_address
Edit Copy Delete	1	D D University, Nadiad

- STUDENT Table:



+ Options

	student_id	student_name	studentAddress_college_id
Edit Copy Delete	1	Kisan	1
Edit Copy Delete	2	Radha	1

Due to the following code in Student entity:

```
@ManyToOne(cascade=CascadeType.ALL)
  private StudentAddress studentAddress;
```

Column automatically added.
In column name studentAddress _college_id,
studentAddress is the name of the field of type StudentAddress and college_id is the Id (column) of StudentAddress

Many-to-one Bidirectional mapping

- Instead of saving students object, we want to write code for adding studentAddress object and associated student objects should get saved automatically.
- We need to add the following code in StudentAddress class

```
@OneToMany(cascade=CascadeType.ALL, mappedBy="studentAddress")
private Set<Student> students = new HashSet<Student>(0);
public Set<Student> getStudents() {
   return students;
}
public void setStudents(Set<Student> students) {
   this.students = students;
}
```

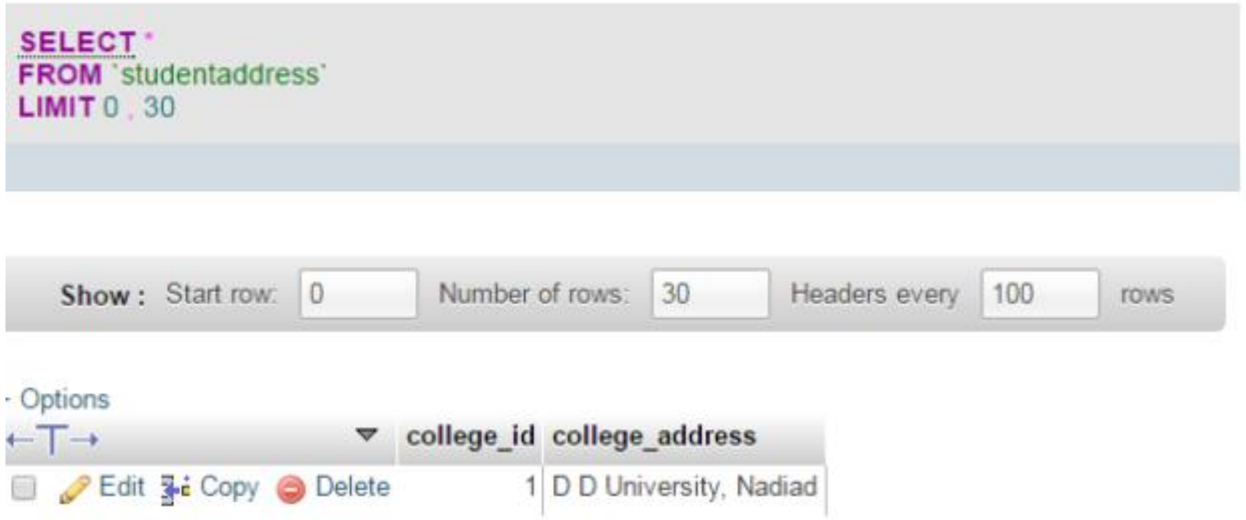
- We need to understand the use of mappedBy

Many-to-one Bidirectional mapping

- Add following code in Main class

```
(studentAddress.getStudents()).add(student1);
 (studentAddress.getStudents()).add(student2);
 SessionFactory sessionFactory= new
AnnotationConfiguration().configure().buildSessionFactory();
…
 session.save(studentAddress);
 session.getTransaction().commit();
 session.close();
…
```

We get the same result

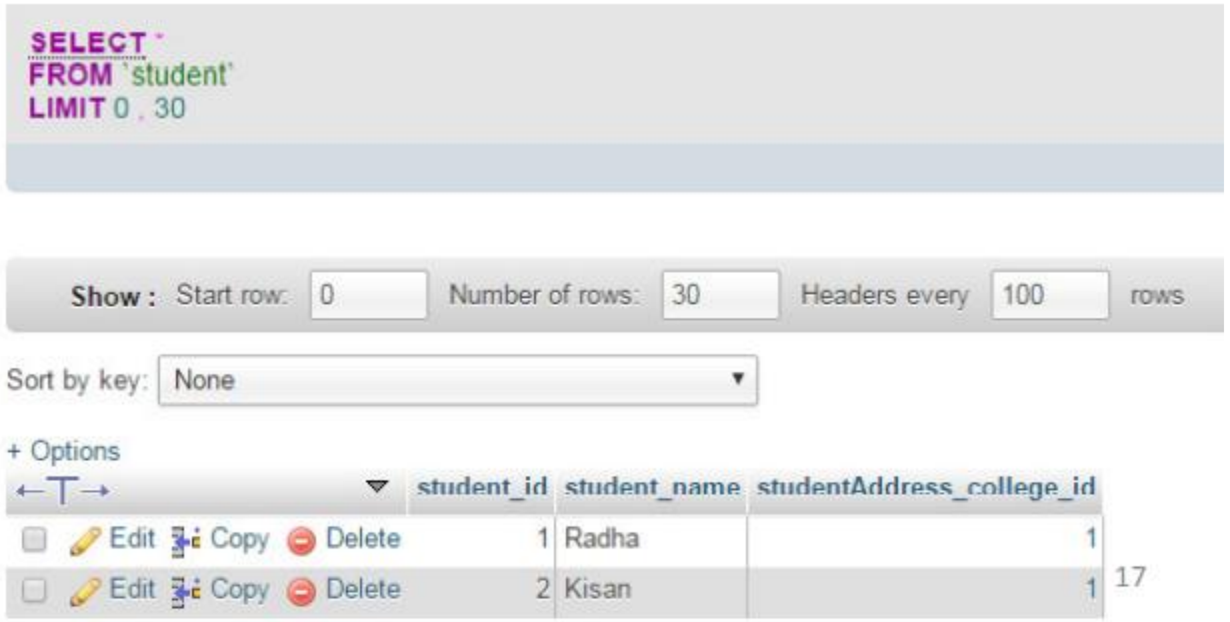


```
SELECT *
FROM `studentaddress`
LIMIT 0 , 30
```

Show : Start row: 0 Number of rows: 30 Headers every 100 rows

- Options

	college_id	college_address
Edit Copy Delete	1	D D University, Nadiad



```
SELECT *
FROM `student`
LIMIT 0 , 30
```

Show : Start row: 0 Number of rows: 30 Headers every 100 rows

Sort by key: None

+ Options

	student_id	student_name	studentAddress_college_id
Edit Copy Delete	1	Radha	1
Edit Copy Delete	2	Kisan	1

Why to use mappedBy property in Bidirectional mapping?

- Lets understand by removing that property
- Remove mappedBy property from StudentAddress class

Instead of

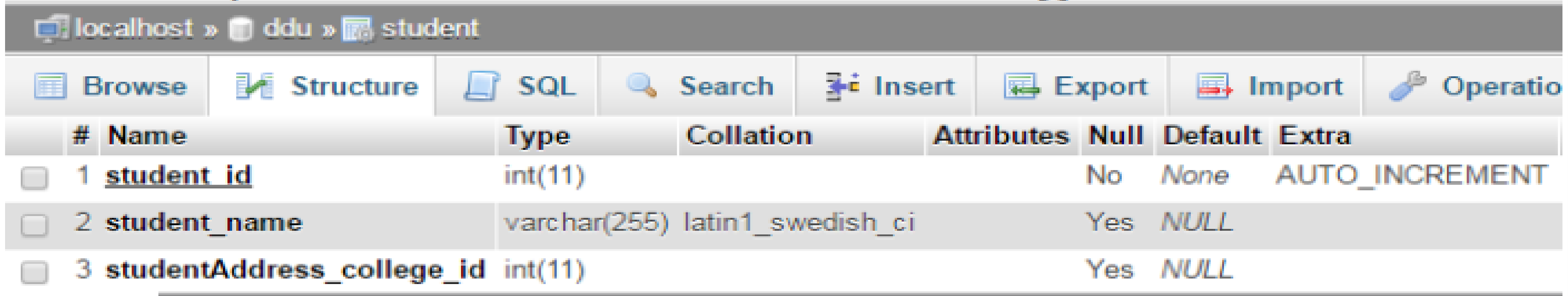
```
@OneToMany(cascade=CascadeType.ALL, mappedBy="studentAddress")
```

Write the following:

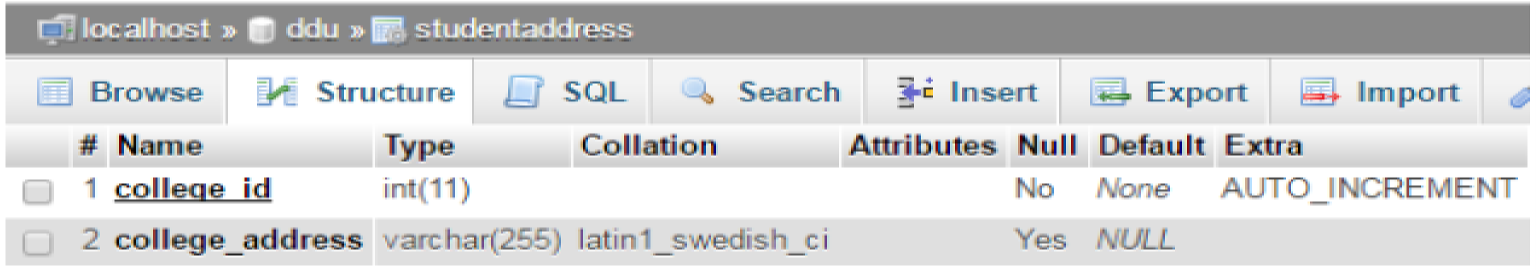
```
  @OneToMany(cascade=CascadeType.ALL)
  private Set<Student> students = new HashSet<Student>(0);
```

Removing mappedBy property in Bidirectional mapping

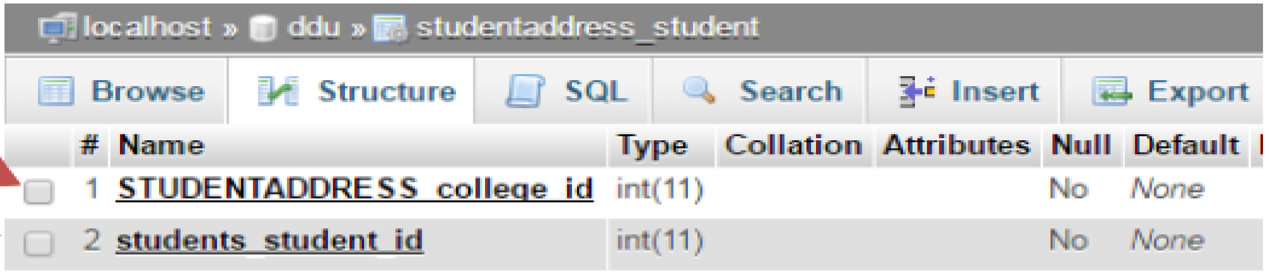
- Instead of two tables, hibernate creates following three tables


localhost » ddu » student

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	student_id	int(11)			No	None	AUTO_INCREMENT
2	student_name	varchar(255)	latin1_swedish_ci		Yes	NULL	
3	studentAddress_college_id	int(11)			Yes	NULL	


localhost » ddu » studentaddress

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	college_id	int(11)			No	None	AUTO_INCREMENT
2	college_address	varchar(255)	latin1_swedish_ci		Yes	NULL	


localhost » ddu » studentaddress_student

#	Name	Type	Collation	Attributes	Null	Default
1	STUDENTADDRESS_college_id	int(11)			No	None
2	students_student_id	int(11)			No	None

Due to:

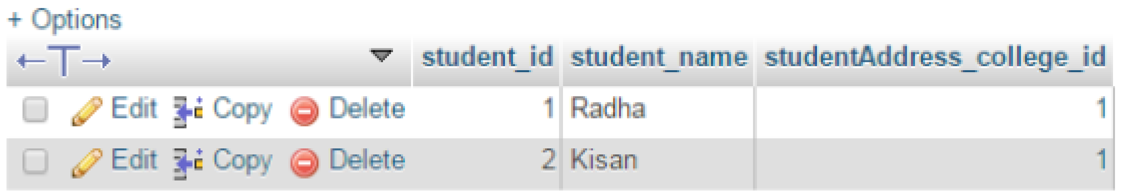
```
@Table(name="STUDENTADDRESS")
public class StudentAddress {
    @Id
    @GeneratedValue
    private int college_id;
```

Due to:

```
private Set<Student> students = new HashSet<Student>(0);
```

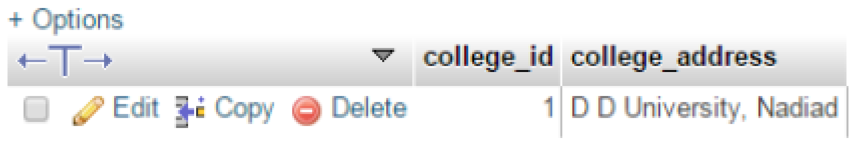
Removing mappedBy property in Bidirectional mapping

- student



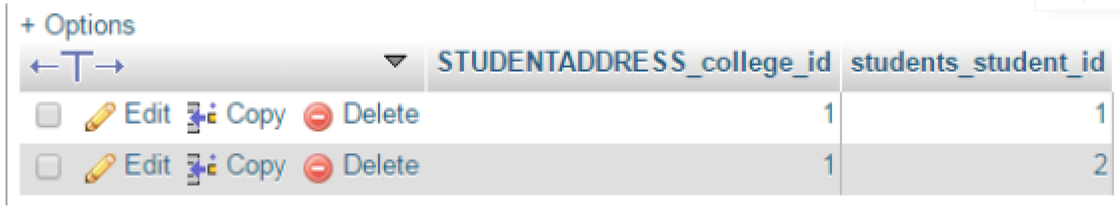
student_id	student_name	studentAddress_college_id
1	Radha	1
2	Kisan	1

- studentaddress



college_id	college_address
1	D D University, Nadiad

- studentaddress_student



STUDENTADDRESS_college_id	students_student_id
1	1
1	2

Why we need mappedBy property in Bidirectional mapping?

- If Student class already has mapping column, why we need third table?
- In bidirectional mapping, we need to specify which side hibernate should consider for mapping the two tables.
- After reading the following lines in Student class

```
@ManyToOne(cascade=CascadeType.ALL)
 private StudentAddress studentAddress;
```

  - Hibernate added a column studentAddress_college_id in student table
- After reading the following lines in StudentAddress class

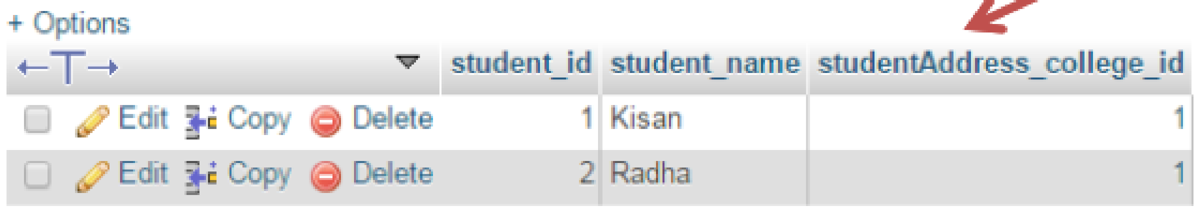
```
@OneToMany(cascade=CascadeType.ALL)
 private Set<Student> students = new HashSet<Student>(0);
```

  - Hibernate created a new table for this mapping (as if it adds a single column to the table, a single column can’t hold many student ids)

Why we need mappedBy property in Bidirectional mapping?

- If we want hibernate to do mapping task only once, not twice, we specify mappedBy property.
- For bidirectional relationships, we have a concept of ownership of a relation.
  - It indicates who is the owner of this relationship?
- In our example, Student class is the owner of the relation.

Student Table:



+ Options

←T→	student_id	student_name	studentAddress_college_id
Edit Copy Delete	1	Kisan	1
Edit Copy Delete	2	Radha	1

Why we need mappedBy property in Bidirectional mapping?

- In StudentAddress class, we have written the following:

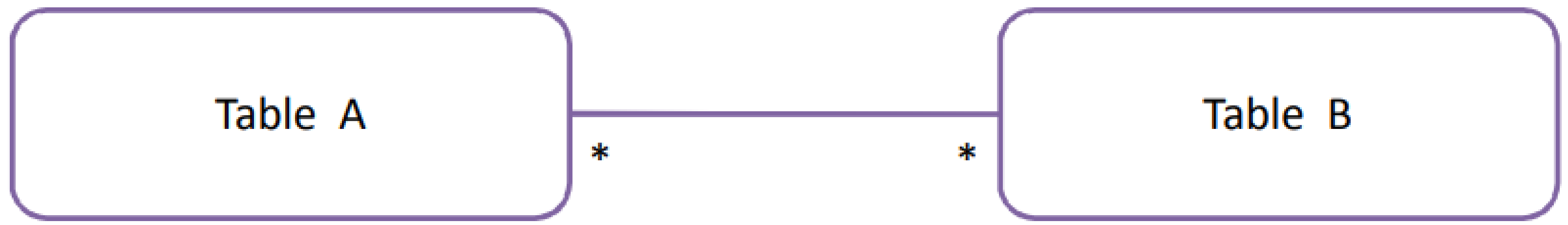
```
@OneToMany(cascade=CascadeType.ALL, mappedBy="studentAddress")
private Set<Student> students = new HashSet<Student>(0);
```

- In Student class, we have written the following:

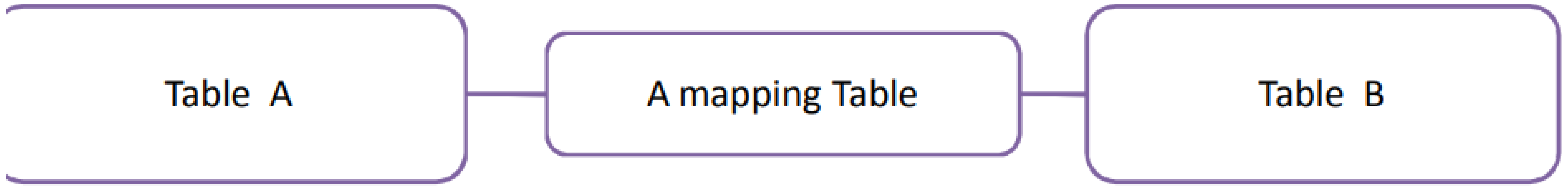
```
@ManyToOne(cascade=CascadeType.ALL)
private StudentAddress studentAddress;
```

- Student class is owning the relation via added column
- mappedBy attribute are always put (annotated) on the inverse side of relation ship and specifies with it's attribute value
  - Student is the owner and inverse side is StudentAddress
- StudentAddress class (in which we specify mappedBy property) is not owning the relation.

Many to many mapping



- If many records of Table A can be linked with many records in Table B then,
  - Table A has many-to-many relationship with Table B.
  - Table B has many-to-many relationship with Table A.
- Such relation can be created by a third table



Many to many mapping : Example

student_id	student_name
1	Kamal
2	Radha
3	Kisan

student

certification_id	certification_name
1	Red Hat Certification
2	Java Certification
3	Oracle Database Certification

studentcertification

Mapping Table

student_id	certification_id
1	2
1	3
2	1
3	3

student_studentcertification

Many-to-Many mapping example (Code without annotations for mapping)

- Student class

```
@Entity
@Table(name="STUDENT")
public class Student {
   @Id
   @GeneratedValue
   private int student_id;

   private String student_name;

   public int getStudent_id() {
      return student_id;
   }
```

Many-to-Many mapping example (Code without annotations for mapping)

```
    public String getStudent_name() {
        return student_name;
    }

    public void setStudent_name(String student_name) {
        this.student_name = student_name;
    }

}
```

Many-to-Many mapping example (Code without annotations for mapping)

- StudentCertification class

```
@Entity
@Table(name="STUDENTCERTIFICATION")
public class StudentCertification {
   @Id
   @GeneratedValue
   private int certification_id;
   private String certification_name;

   public int getCertification_id() {
      return certification_id;
   }
```

Many-to-Many mapping example (Code without annotations for mapping)

```
    public String getCertification_name() {
        return certification_name;
    }

    public void setCertification_name(String certification_name) {
        this.certification_name = certification_name;
    }
}
```

Many-to-Many mapping example

- Add the relation in the Student class (Owner class)
- Add the following code in Student class

```
@ManyToMany(cascade=CascadeType.ALL)
   private Set<StudentCertification> studentCertification = new
     HashSet<StudentCertification>(0);

   public Set<StudentCertification> getStudentCertification() {
      return studentCertification;
   }

   public void setStudentCertification(Set<StudentCertification>
     studentCertification) {
      this.studentCertification = studentCertification;
   }
```

Many-to-Many mapping example

- Configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate
   Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-
   configuration-3.0.dtd">
<hibernate-configuration>
 <session-factory>
  <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect
   </property>
  <property
   name="hibernate.connection.driver_class">com.mysql.jdbc.Driver
   </property>
```

Many-to-Many mapping example

```
  <property
    name="hibernate.connection.url">jdbc:mysql://localhost:3306/ddu?zeroD
    ateTimeBehavior=convertToNull</property>
  <property name="hibernate.connection.username">root</property>

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

  <mapping class="hibernate.relations.many2many.Student"/>
  <mapping class="hibernate.relations.many2many.StudentCertification"/>

 </session-factory>
</hibernate-configuration>
```

Many-to-Many mapping example

- Main class

```
public class Main {
public static void main(String[] args) {
    StudentCertification studentCertification1=new StudentCertification();
    studentCertification1.setCertification_name("Red Hat Certification");

    StudentCertification studentCertification2=new StudentCertification();
    studentCertification2.setCertification_name("Java Certification");

    StudentCertification studentCertification3=new StudentCertification();
    studentCertification3.setCertification_name("Oracle Database
  Certification");
```

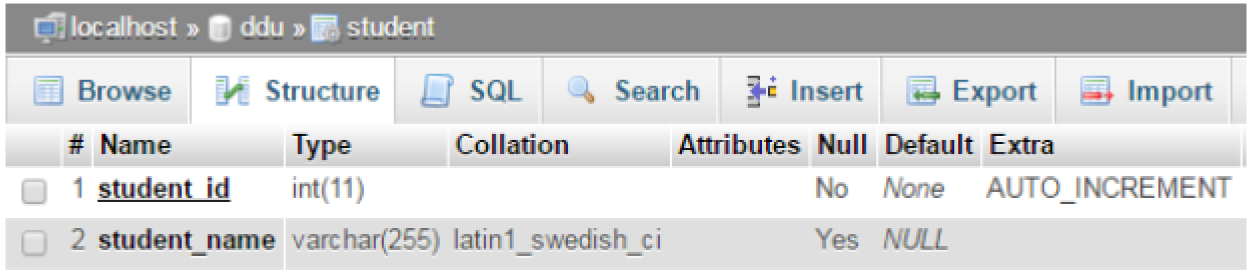
Many-to-Many mapping example

```
Student student1=new Student();
student1.setStudent_name("Kamal");
(student1.getStudentCertification()).add(studentCertification2);
(student1.getStudentCertification()).add(studentCertification3);

Student student2=new Student();
student2.setStudent_name("Radha");
(student2.getStudentCertification()).add(studentCertification1);

Student student3=new Student();
student3.setStudent_name("Kisan");
(student3.getStudentCertification()).add(studentCertification3);
```

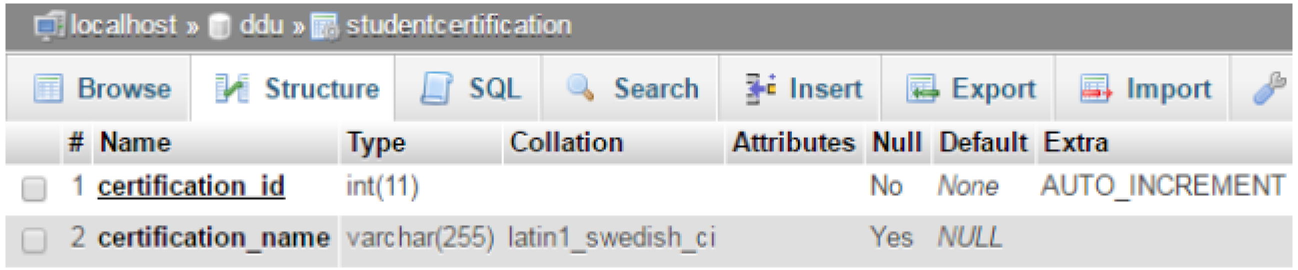
Many-to-Many mapping example

```
 SessionFactory sessionFactory= new
AnnotationConfiguration().configure().buildSessionFactory();
  Session session=sessionFactory.openSession();
  session.beginTransaction();

  session.save(student1); //Student is owner of the relation
  session.save(student2);
  session.save(student3);

  session.getTransaction().commit();
  session.close();
  sessionFactory.close();
}
}
```

Table structures



#	Name	Type	Collation	Attributes	Null	Default	Extra
1	student_id	int(11)			No	None	AUTO_INCREMENT
2	student_name	varchar(255)	latin1_swedish_ci		Yes	NULL	

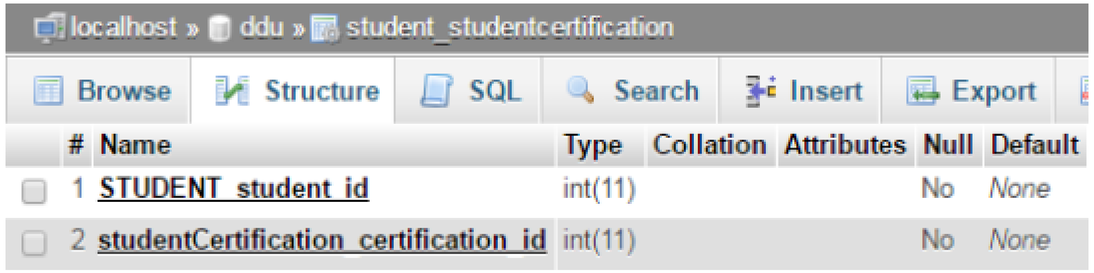


#	Name	Type	Collation	Attributes	Null	Default	Extra
1	certification_id	int(11)			No	None	AUTO_INCREMENT
2	certification_name	varchar(255)	latin1_swedish_ci		Yes	NULL	

Due to the following code:

```
@Entity
@Table(name="STUDENT")
public class Student {
...
@ManyToMany(cascade=CascadeType.ALL)
   private Set<StudentCertification> studentCertification = new
HashSet<StudentCertification>(0);
```

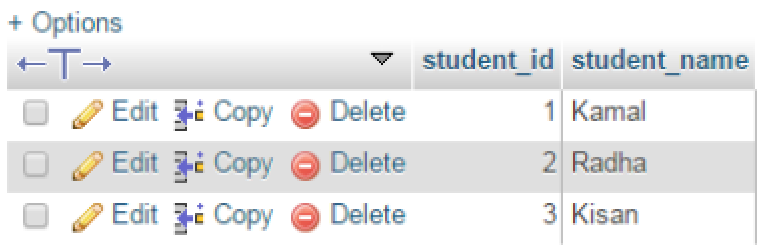
Table is automatically created by hibernate



#	Name	Type	Collation	Attributes	Null	Default
1	STUDENT_student_id	int(11)			No	None
2	studentCertification_certification_id	int(11)			No	None

Records in the tables

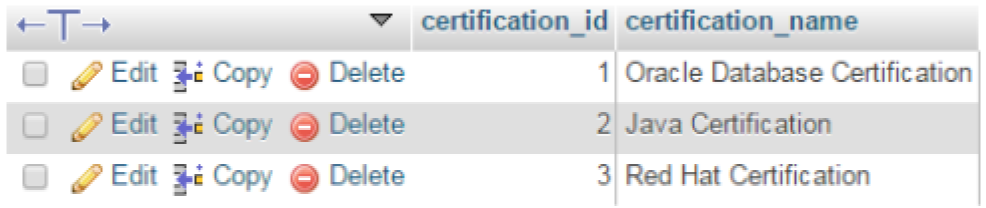
- Student



+ Options

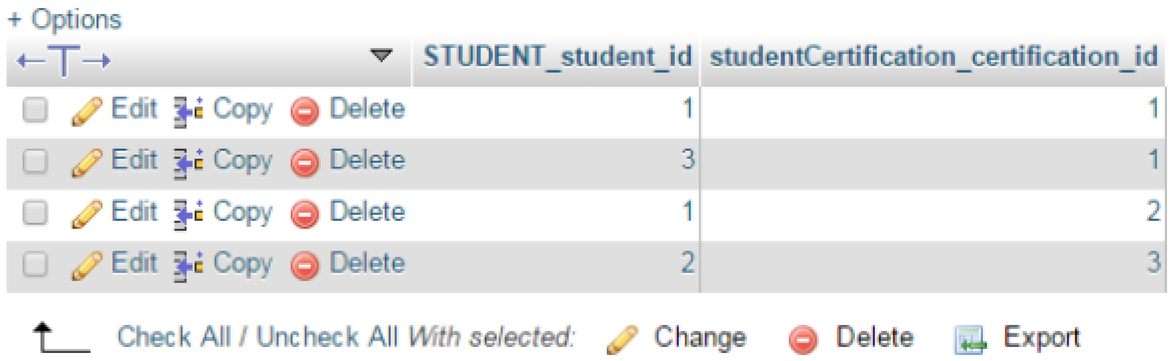
	student_id	student_name
Edit Copy Delete	1	Kamal
Edit Copy Delete	2	Radha
Edit Copy Delete	3	Kisan

- StudentCertification



	certification_id	certification_name
Edit Copy Delete	1	Oracle Database Certification
Edit Copy Delete	2	Java Certification
Edit Copy Delete	3	Red Hat Certification

- student_studentcertification



+ Options

	STUDENT_student_id	studentCertification_certification_id
Edit Copy Delete	1	1
Edit Copy Delete	3	1
Edit Copy Delete	1	2
Edit Copy Delete	2	3

Check All / Uncheck All *With selected:* Change Delete Export

Many-to-Many relation in reverse direction also

- If we want many-to-many relation in reverse direction also
- I.e., if we save StudentCertification objects, related Student objects should also get stored automatically
  - Add @ManyToMany annotation (with mappedBy property) in StudentCertification class

loca
Exp

Option 1: SQL Query

STUDENTADDRESS Table:

college_id	college_address

```
create table STUDENTADDRESS (
  college_id BIGINT NOT NULL AUTO_INCREMENT,
  college_address VARCHAR(30) NOT NULL,
  PRIMARY KEY (college_id)
);
```

Option 1: SQL Query

STUDENT Table

registration_id	student_name	college_id

```
create table STUDENT (
  registration_id BIGINT NOT NULL AUTO_INCREMENT,
  student_name VARCHAR(30) NOT NULL,
   college_id BIGINT NOT NULL,
   PRIMARY KEY (student_id),
   CONSTRAINT student_studentaddress FOREIGN KEY (college_id)
     REFERENCES STUDENTADDRESS (college_id) ON UPDATE CASCADE ON
     DELETE CASCADE
);
```