luv2code

Mapping Sets

Set

Collection of items that contains no duplicates

Use Case for Sets

Use Case for Sets

- Useful when
  - Only interested in membership … for yes/no decisions

Use Case for Sets

- Useful when
  - Only interested in membership … for yes/no decisions
  - Order / sequence is not important

Use Case for Sets

- Useful when
  - Only interested in membership … for yes/no decisions
  - Order / sequence is not important
- Examples

Use Case for Sets

- Useful when
  - Only interested in membership … for yes/no decisions
  - Order / sequence is not important

- Examples
  - Playing cards: Poker … order of cards in your hand does not matter

Use Case for Sets

- Useful when
  - Only interested in membership … for yes/no decisions
  - Order / sequence is not important

- Examples
  - Playing cards: Poker … order of cards in your hand does not matter
  - Membership: Is my name on the guest list?

Student and Images

Student and Images

- A student will have a ***set*** of images (image file names)

Student and Images

- A student will have a ***set*** of images (image file names)
  - We don't care about the order of the images

Student and Images

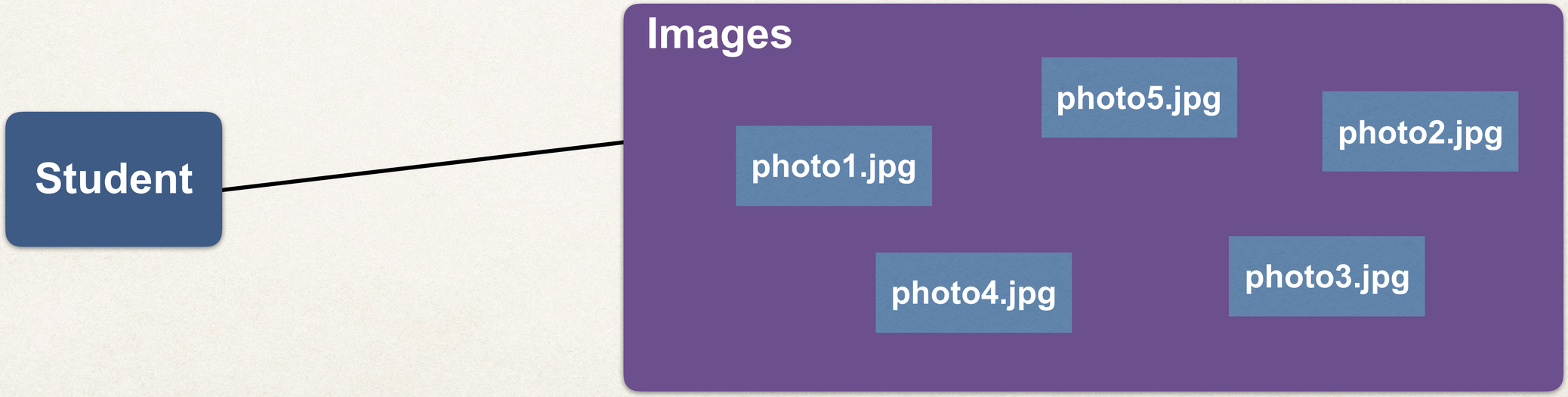
- A student will have a ***set*** of images (image file names)
  - We don't care about the order of the images
  - There should not be any duplicate images

Student and Images

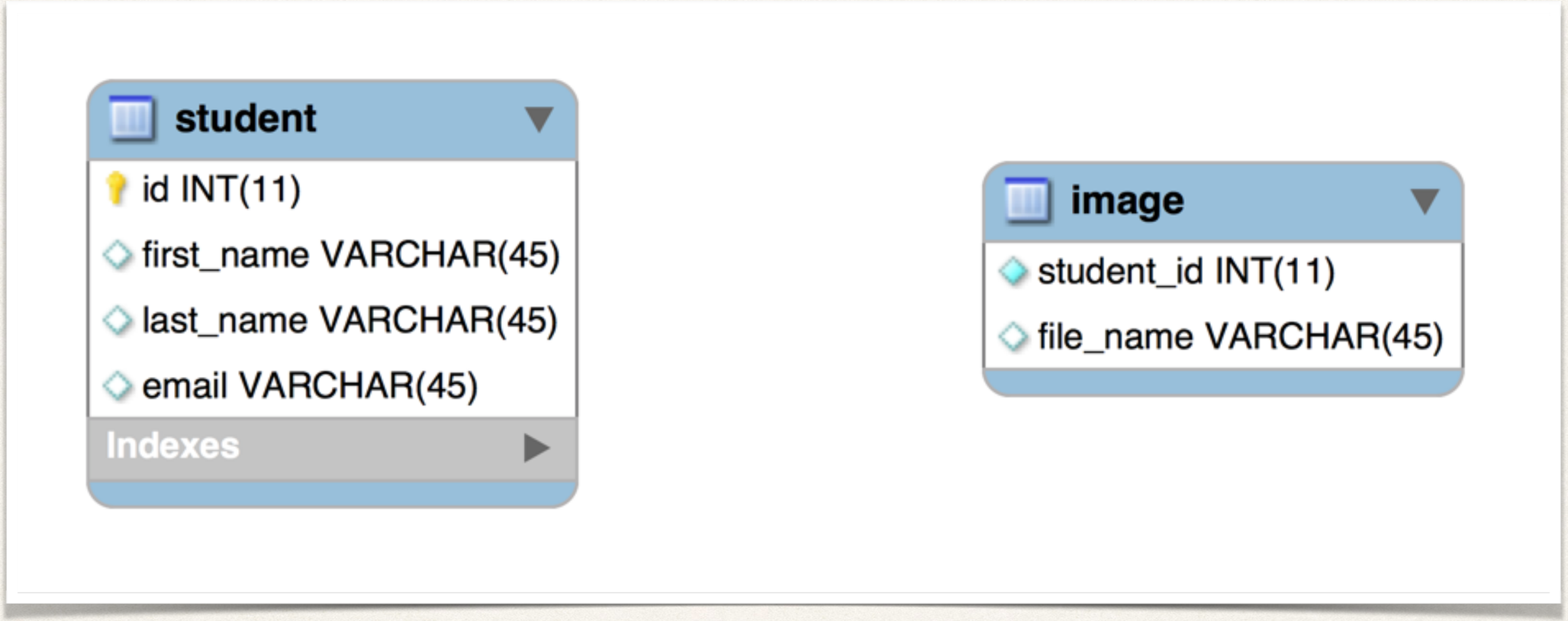
- A student will have a ***set*** of images (image file names)
  - We don't care about the order of the images
  - There should not be any duplicate images

Student and Images

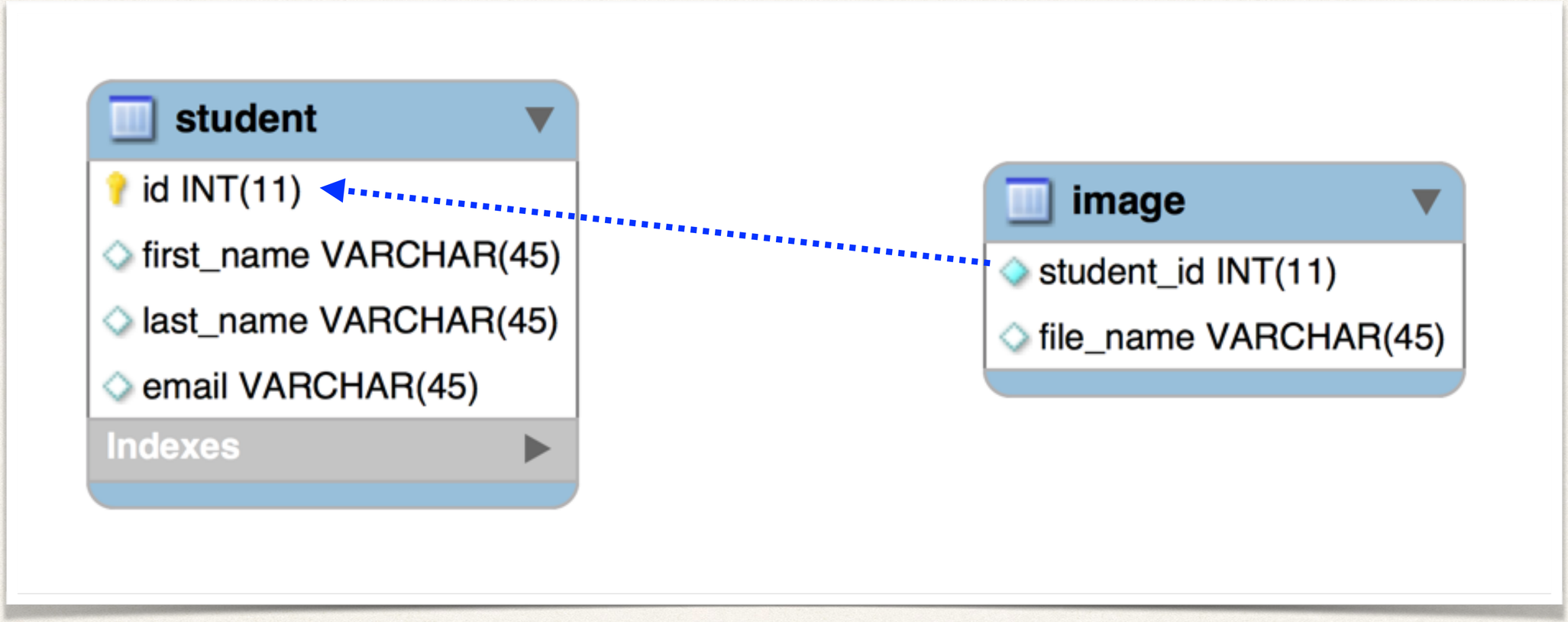
- A student will have a ***set*** of images (image file names)
  - We don't care about the order of the images
  - There should not be any duplicate images



Database Diagram



Database Diagram



Development Process

Step-By-Step

Development Process

Step-By-Step

1. Create database tables

Development Process

Step-By-Step

1. Create database tables

2. Create Student entity

Development Process

Step-By-Step

1. Create database tables

2. Create Student entity

3. Map the element collection

Development Process

Step-By-Step

1. Create database tables

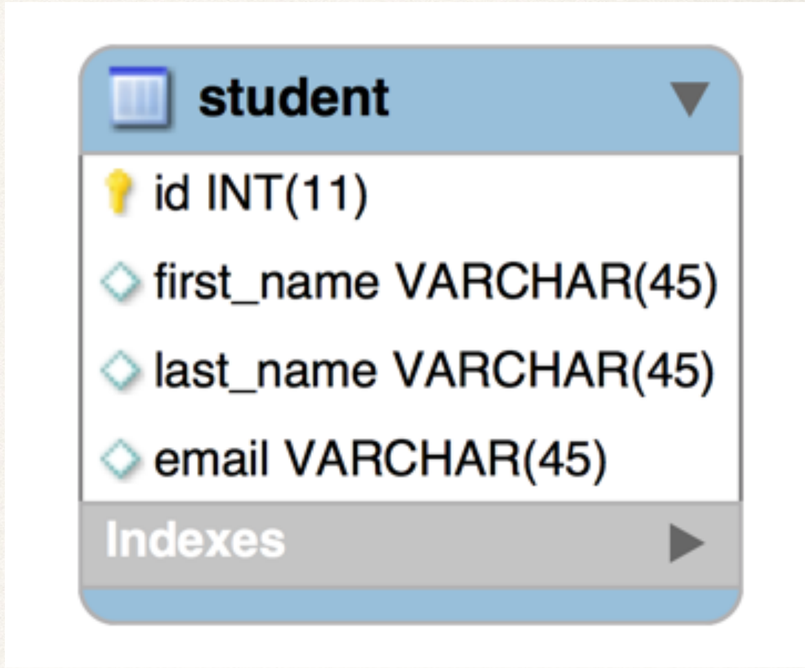
2. Create Student entity

3. Map the element collection

4. Develop the main application

Step 1: Create database tables: `student`

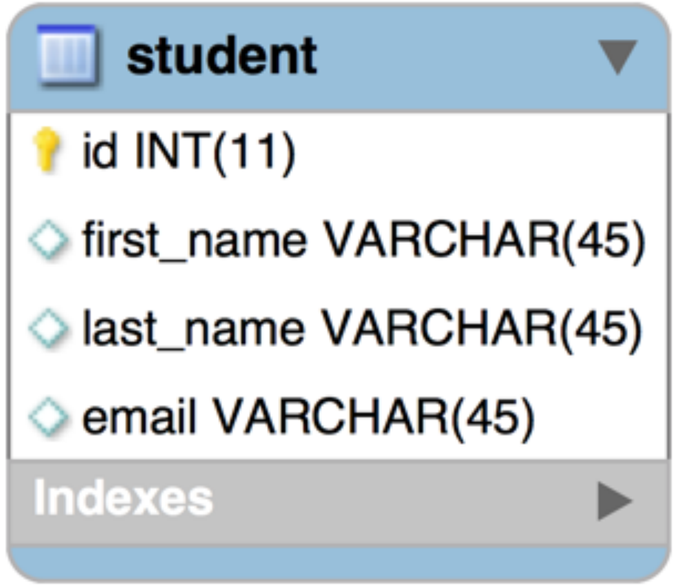
Step 1: Create database tables: `student`



Step 1: Create database tables: `student`

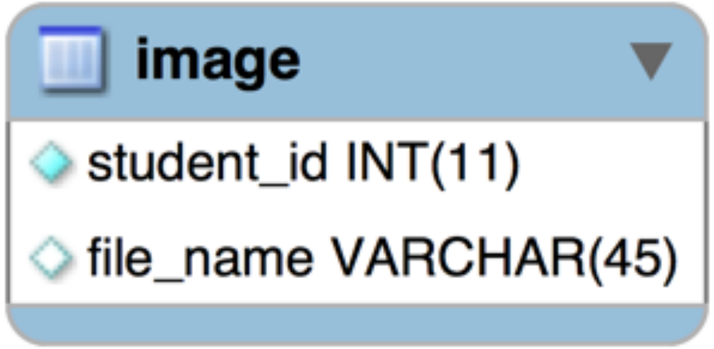
File: hb_student_tracker_student.sql

```
CREATE TABLE student (

  id int(11) NOT NULL AUTO_INCREMENT,
  first_name varchar(45) DEFAULT NULL,
  last_name varchar(45) DEFAULT NULL,
  email varchar(45) DEFAULT NULL,

  PRIMARY KEY (id)
);
…
```



Step 1: Create database tables: `image`

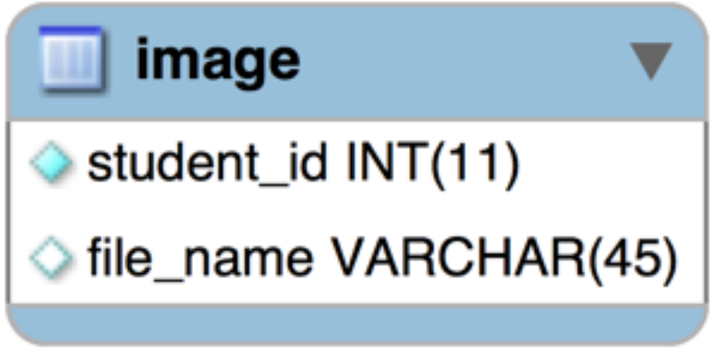
Step 1: Create database tables: `image`



Step 1: Create database tables: `image`

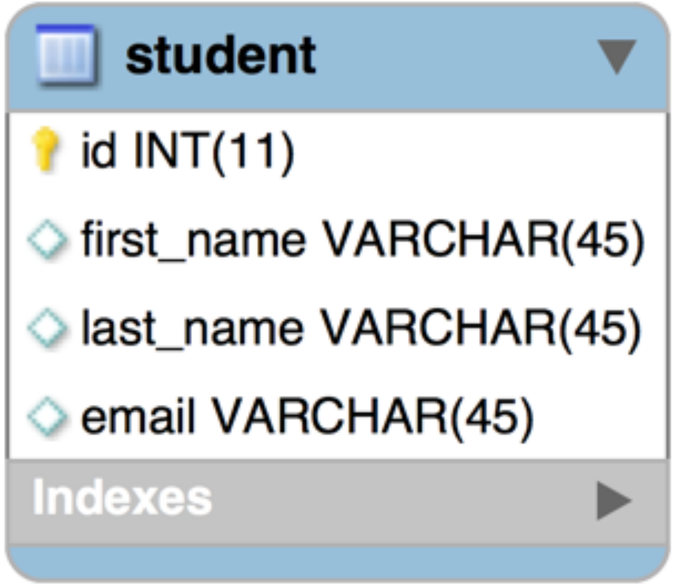
File: hb_student_tracker_student.sql (continued)

```
…
CREATE TABLE image (

  student_id int(11) NOT NULL,
  file_name varchar(45) DEFAULT NULL

);
```



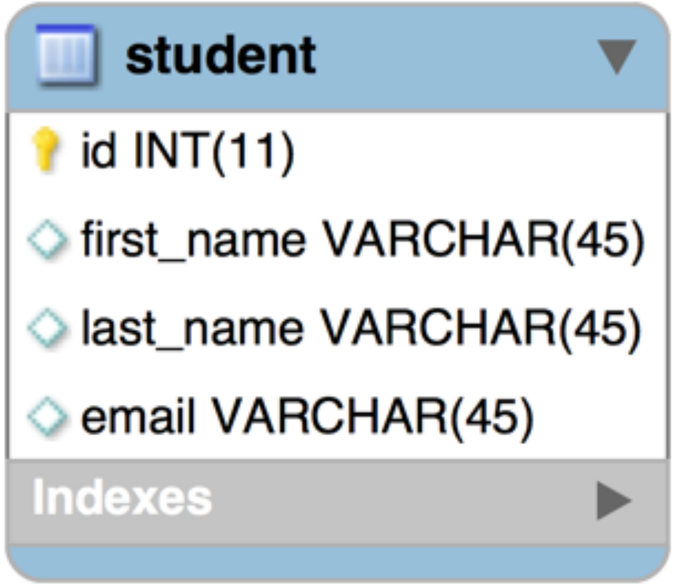
Step 2: Create Student entity

Step 2: Create Student entity



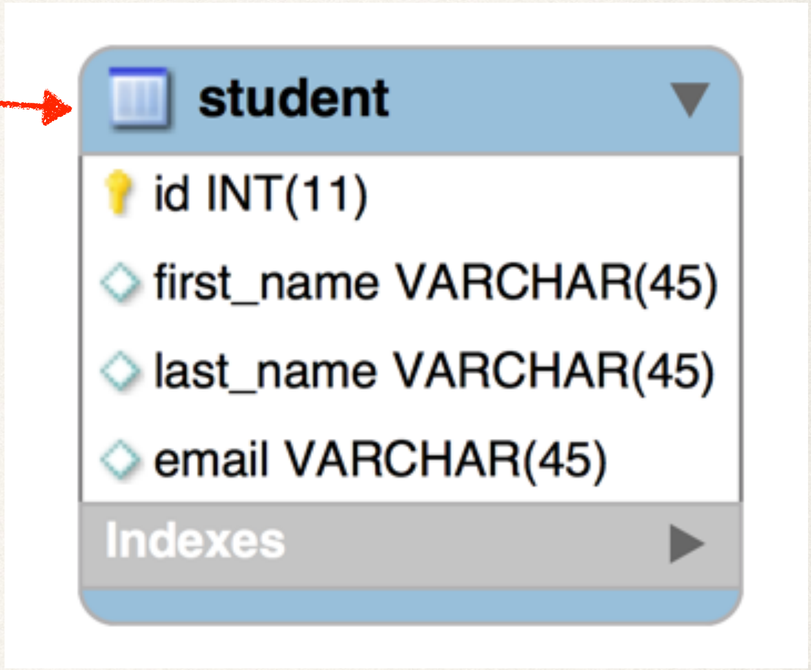
Step 2: Create Student entity

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

  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private int id;

  @Column(name="first_name")
  private String firstName;

  @Column(name="last_name")
  private String lastName;

  @Column(name="email")
  private String email;


  …
  // constructors, getters / setters
}
```



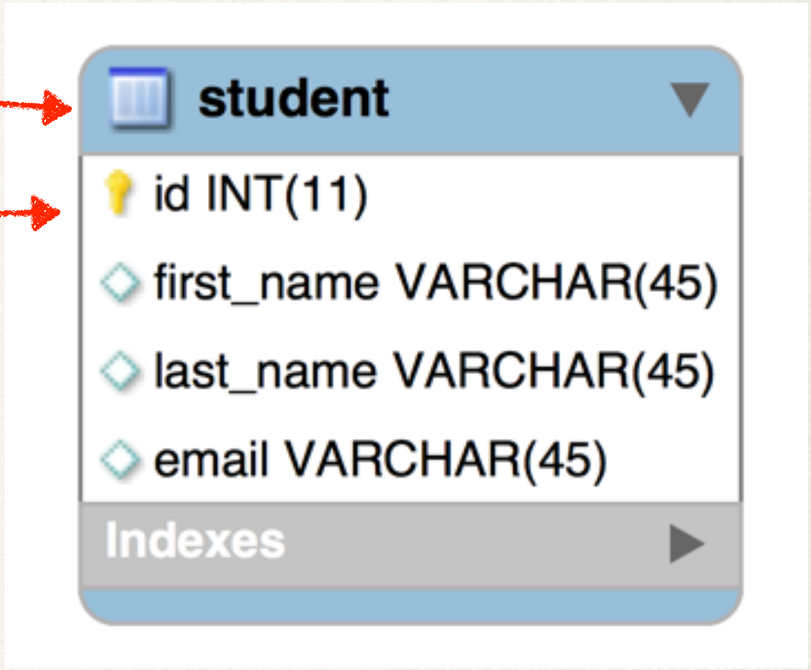
Step 2: Create Student entity

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

  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private int id;

  @Column(name="first_name")
  private String firstName;

  @Column(name="last_name")
  private String lastName;

  @Column(name="email")
  private String email;

  …
  // constructors, getters / setters
}
```



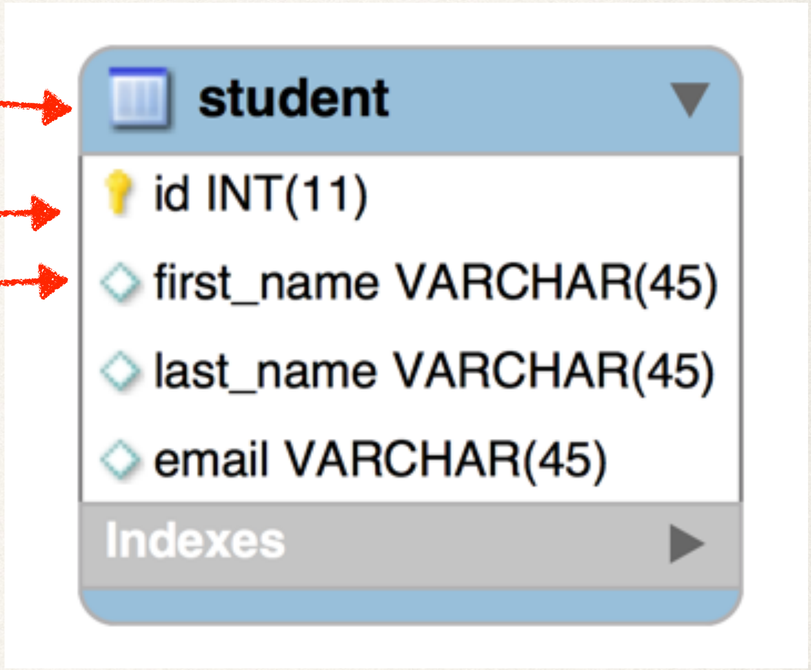
Step 2: Create Student entity

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

  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private int id;

  @Column(name="first_name")
  private String firstName;

  @Column(name="last_name")
  private String lastName;

  @Column(name="email")
  private String email;

  …
  // constructors, getters / setters
}
```



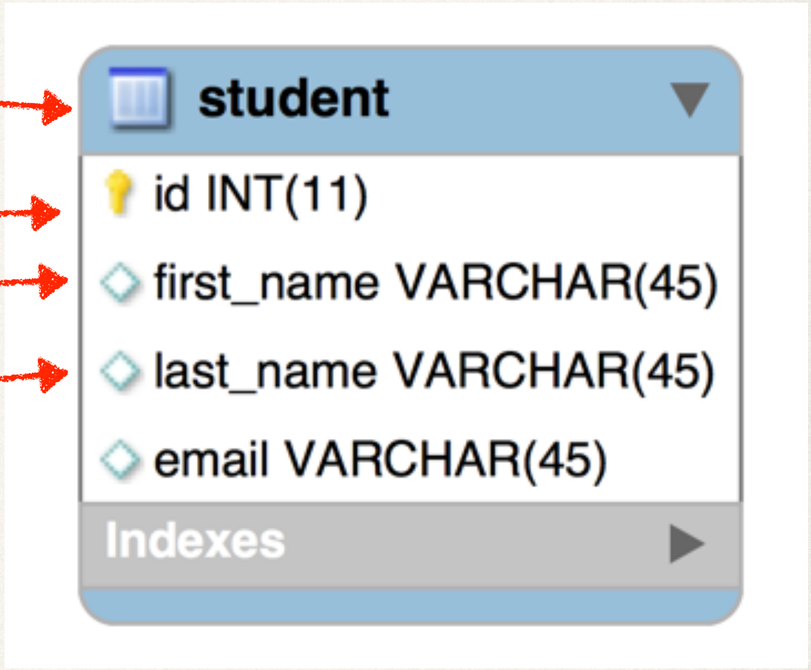
Step 2: Create Student entity

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

  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private int id;

  @Column(name="first_name")
  private String firstName;

  @Column(name="last_name")
  private String lastName;

  @Column(name="email")
  private String email;

  …
  // constructors, getters / setters
}
```



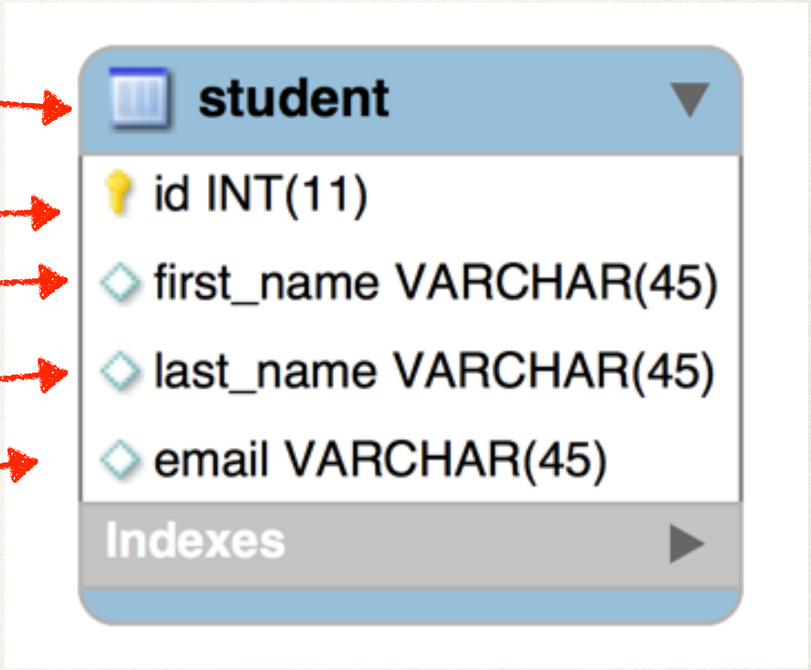
Step 2: Create Student entity

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

  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private int id;

  @Column(name="first_name")
  private String firstName;

  @Column(name="last_name")
  private String lastName;

  @Column(name="email")
  private String email;

  …
  // constructors, getters / setters
}
```



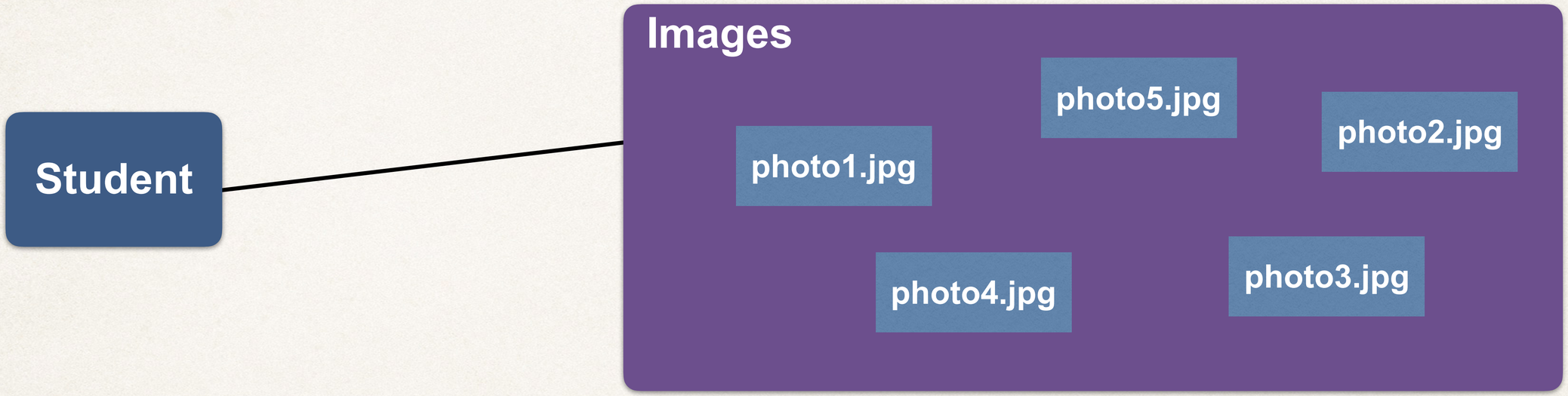
Step 2: Create Student entity

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

  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private int id;

  @Column(name="first_name")
  private String firstName;

  @Column(name="last_name")
  private String lastName;

  @Column(name="email")
  private String email;

  …
  // constructors, getters / setters
}
```

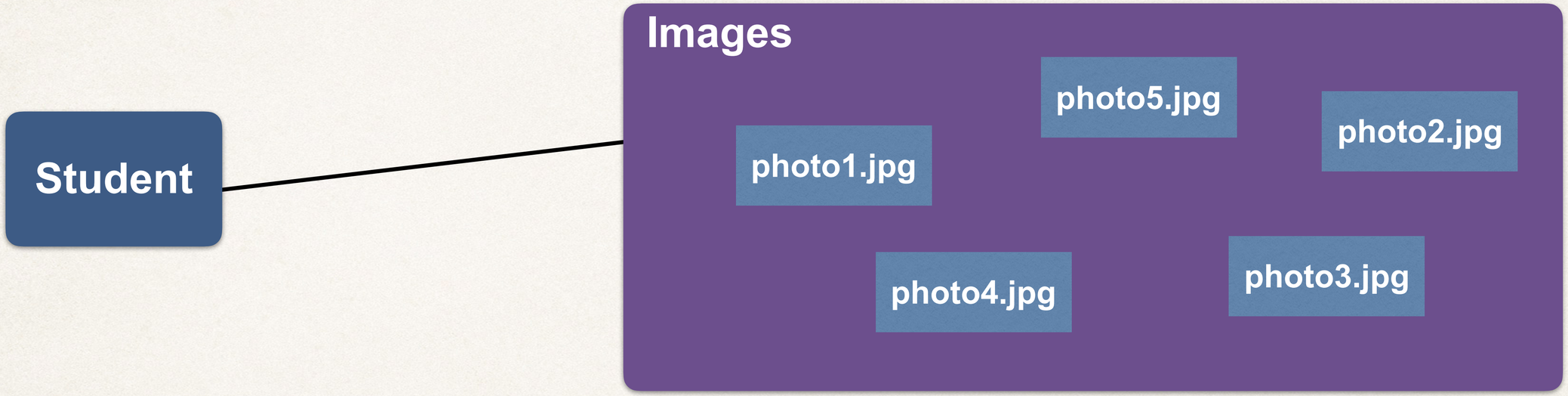


Step 3: Map the element collection

Step 3: Map the element collection



Step 3: Map the element collection




```
private Set<String> images = new HashSet<String>();
```

Annotations to Map Collections

Annotations to Map Collections

Annotation	Description

Annotations to Map Collections

Annotation	Description
`@ElementCollection`	**Declares an element collection mapping. The data for the collection is stored in a separate table.**

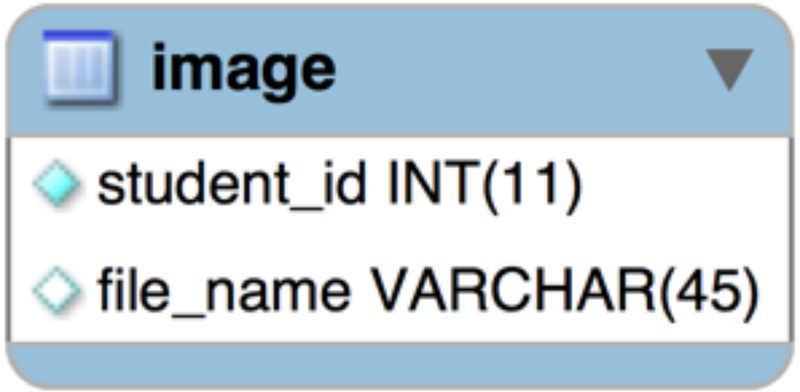
Annotations to Map Collections

Annotation	Description
`@ElementCollection`	**Declares an element collection mapping. The data for the collection is stored in a separate table.**
`@CollectionTable`	**Specifies the name of table that will hold the collection. Also provides the join column to refer to primary table.**

Annotations to Map Collections

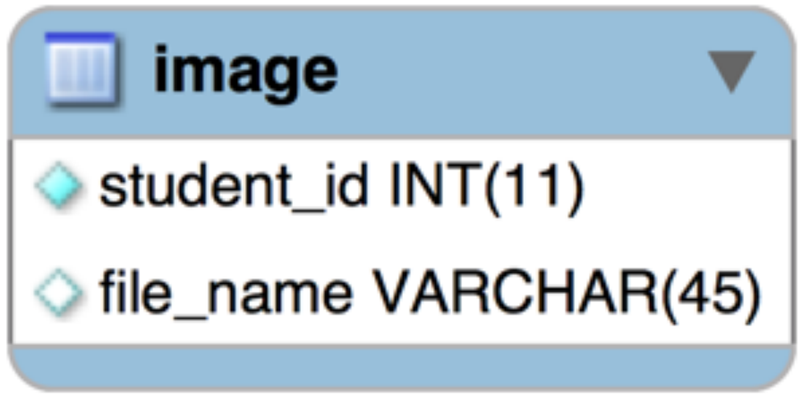
Annotation	Description
`@ElementCollection`	**Declares an element collection mapping. The data for the collection is stored in a separate table.**
`@CollectionTable`	**Specifies the name of table that will hold the collection. Also provides the join column to refer to primary table.**
`@Column`	**The name of the column to map in the collection table.**

Mapping the Collection



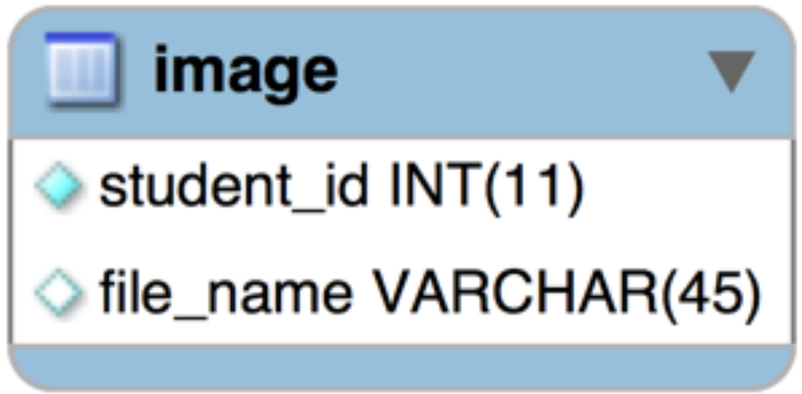
Mapping the Collection

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

  …

  private Set<String> images = new HashSet<String>();

  …
}
```



Mapping the Collection

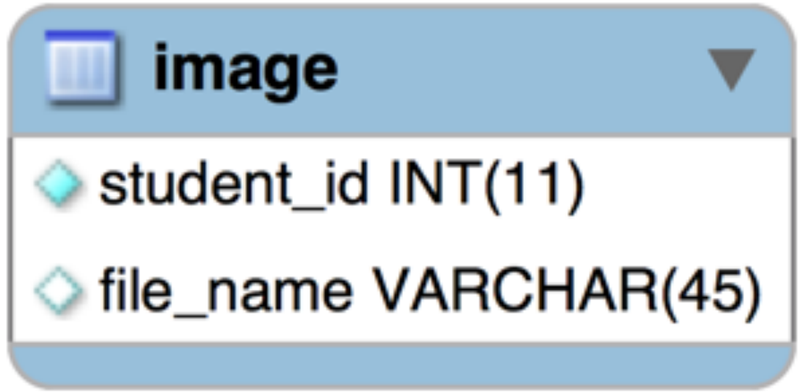
```
@Entity
@Table(name="student")
public class Student {

  …

  private Set<String> images = new HashSet<String>();

  …
}
```



Use a **`Set`** to track image file names

Mapping the Collection

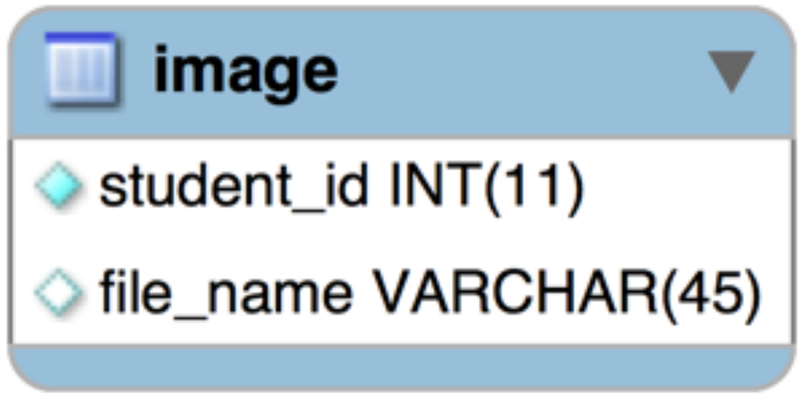
```
@Entity
@Table(name="student")
public class Student {

  …

  @ElementCollection
  @CollectionTable(
      name="image",
      joinColumns= @JoinColumn(name="student_id"))
  @Column(name="file_name")
  private Set<String> images = new HashSet<String>();


  …
}
```



Mapping the Collection

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

  …

  @ElementCollection
  @CollectionTable(
      name="image",
      joinColumns= @JoinColumn(name="student_id"))
  @Column(name="file_name")
  private Set<String> images = new HashSet<String>();


  …
}
```

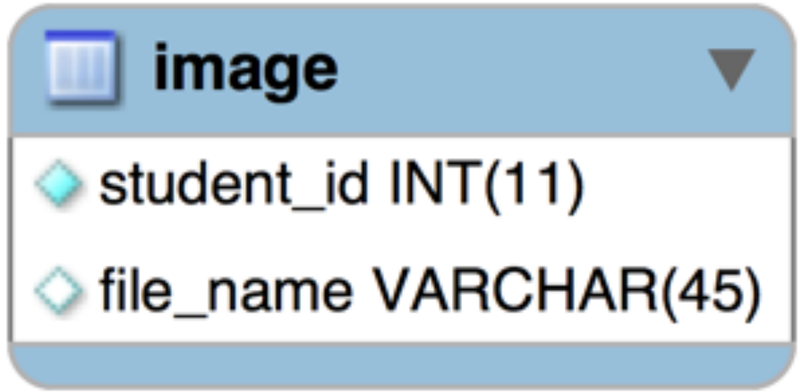
Tells Hibernate we are mapping a collection



Mapping the Collection

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

  …

  @ElementCollection
  @CollectionTable(
      name="image",
      joinColumns= @JoinColumn(name="student_id"))
  @Column(name="file_name")
  private Set<String> images = new HashSet<String>();


  …
}
```

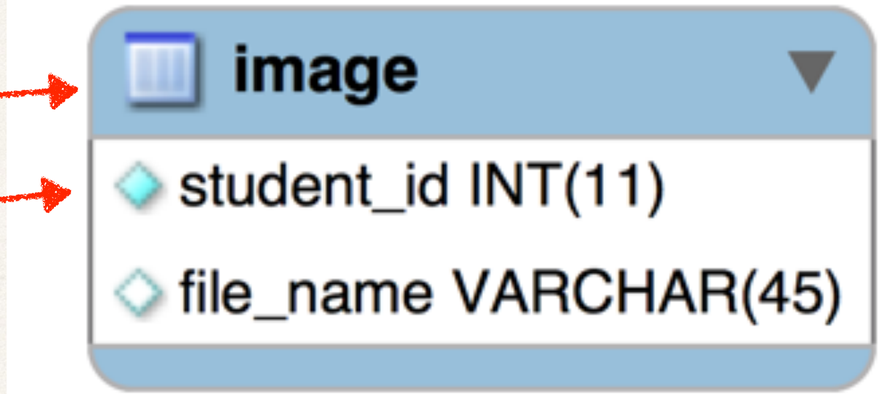
Tells Hibernate we are mapping a collection



Mapping the Collection

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

  …

  @ElementCollection
  @CollectionTable(
      name="image",
      joinColumns= @JoinColumn(name="student_id"))
  @Column(name="file_name")
  private Set<String> images = new HashSet<String>();


  …
}
```

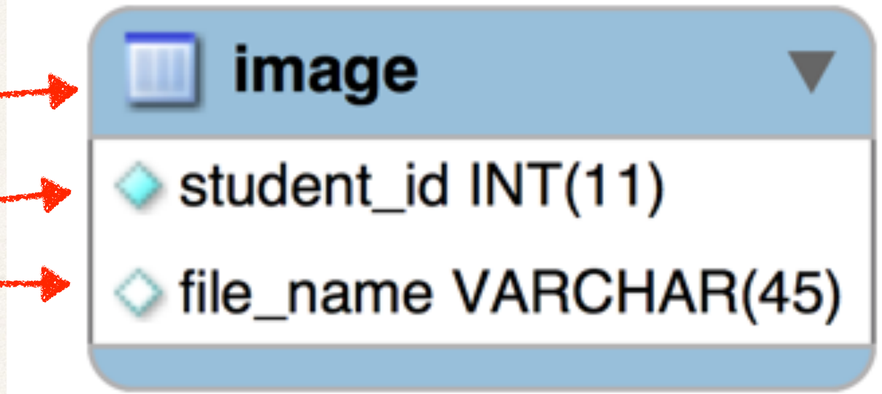
Tells Hibernate we are mapping a collection



Mapping the Collection

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

  …

  @ElementCollection
  @CollectionTable(
      name="image",
      joinColumns= @JoinColumn(name="student_id"))
  @Column(name="file_name")
  private Set<String> images = new HashSet<String>();


  …
}
```

Tells Hibernate we are mapping a collection



Step 4: Develop the main application

Step 4: Develop the main application

```
…
// create the object
Student tempStudent =  new Student("Paul", "Wall", "paul@luv2code.com");
```

Step 4: Develop the main application

```
…
// create the object
Student tempStudent =  new Student("Paul", "Wall", "paul@luv2code.com");
Set<String> theImages = tempStudent.getImages();
```

Get a reference
to the images collection

Step 4: Develop the main application

```
…
// create the object
Student tempStudent =  new Student("Paul", "Wall", "paul@luv2code.com");
Set<String> theImages = tempStudent.getImages();

theImages.add("photo1.jpg");
theImages.add("photo2.jpg");
theImages.add("photo3.jpg");
```

Now, let's add to the images collection

Step 4: Develop the main application

```
…
// create the object
Student tempStudent =  new Student("Paul", "Wall", "paul@luv2code.com");
Set<String> theImages = tempStudent.getImages();

theImages.add("photo1.jpg");
theImages.add("photo2.jpg");
theImages.add("photo3.jpg");

// start a transaction
session.beginTransaction();
```

Step 4: Develop the main application

```
…
// create the object
Student tempStudent =  new Student("Paul", "Wall", "paul@luv2code.com");
Set<String> theImages = tempStudent.getImages();

theImages.add("photo1.jpg");
theImages.add("photo2.jpg");
theImages.add("photo3.jpg");

// start a transaction
session.beginTransaction();

// save the object
System.out.println("Saving the student and images..");
session.persist(tempStudent);
```

Step 4: Develop the main application

```
…
// create the object
Student tempStudent =  new Student("Paul", "Wall", "paul@luv2code.com");
Set<String> theImages = tempStudent.getImages();

theImages.add("photo1.jpg");
theImages.add("photo2.jpg");
theImages.add("photo3.jpg");

// start a transaction
session.beginTransaction();

// save the object
System.out.println("Saving the student and images..");
session.persist(tempStudent);

// commit the transaction
session.getTransaction().commit();
…
```

Run the App

Run the App

Console output

```
Hibernate: insert into student (email, first_name, last_name) values (?, ?, ?)
Hibernate: insert into image (student_id, file_name) values (?, ?)
Hibernate: insert into image (student_id, file_name) values (?, ?)
Hibernate: insert into image (student_id, file_name) values (?, ?)
```

Run the App

Console output

```
Hibernate: insert into student (email, first_name, last_name) values (?, ?, ?)
Hibernate: insert into image (student_id, file_name) values (?, ?)
Hibernate: insert into image (student_id, file_name) values (?, ?)
Hibernate: insert into image (student_id, file_name) values (?, ?)
```

Table: student

id	first_name	last_name	email
101	Paul	Wall	paul@luv2code.com

Run the App

Console output

```
Hibernate: insert into student (email, first_name, last_name) values (?, ?, ?)
Hibernate: insert into image (student_id, file_name) values (?, ?)
Hibernate: insert into image (student_id, file_name) values (?, ?)
Hibernate: insert into image (student_id, file_name) values (?, ?)
```

Table: student

id	first_name	last_name	email
101	Paul	Wall	paul@luv2code.com

Table: image

student_id	file_name
101	photo1.jpg
101	photo2.jpg
101	photo3.jpg

Run the App

Console output

```
Hibernate: insert into student (email, first_name, last_name) values (?, ?, ?)
Hibernate: insert into image (student_id, file_name) values (?, ?)
Hibernate: insert into image (student_id, file_name) values (?, ?)
Hibernate: insert into image (student_id, file_name) values (?, ?)
```

Table: student

id	first_name	last_name	email
101	Paul	Wall	paul@luv2code.com

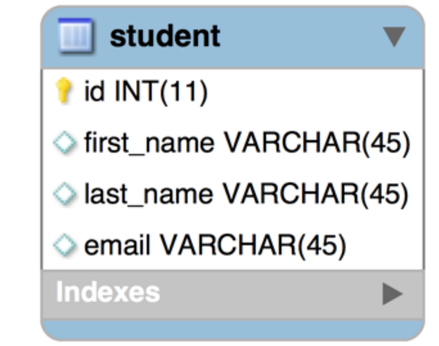
Table: image

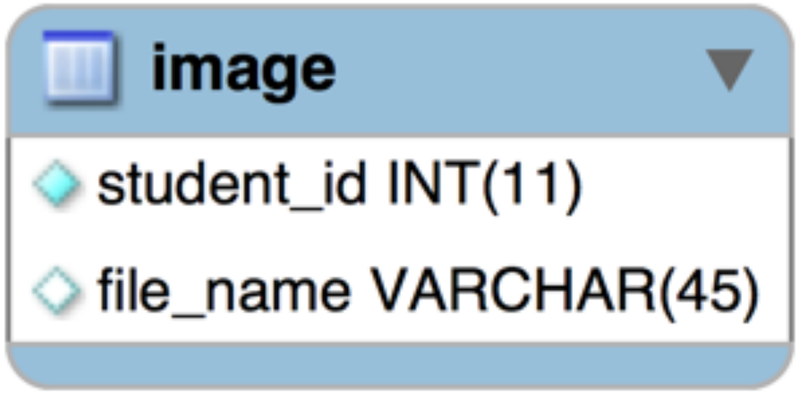
student_id	file_name
101	photo1.jpg
101	photo2.jpg
101	photo3.jpg

Let's Go Deep!

More Info - Mapping the Collection

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




  @ElementCollection
  @CollectionTable(
      name="image",
      joinColumns= @JoinColumn(name="student_id"))
  @Column(name="file_name")
  private Set<String> images = new HashSet<String>();
```

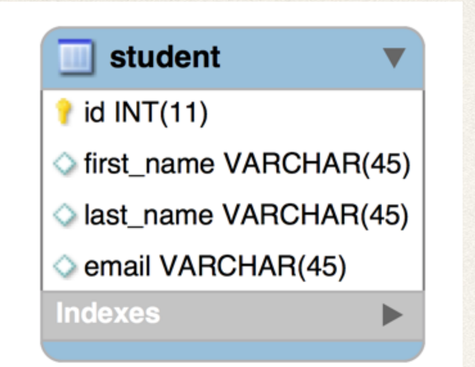


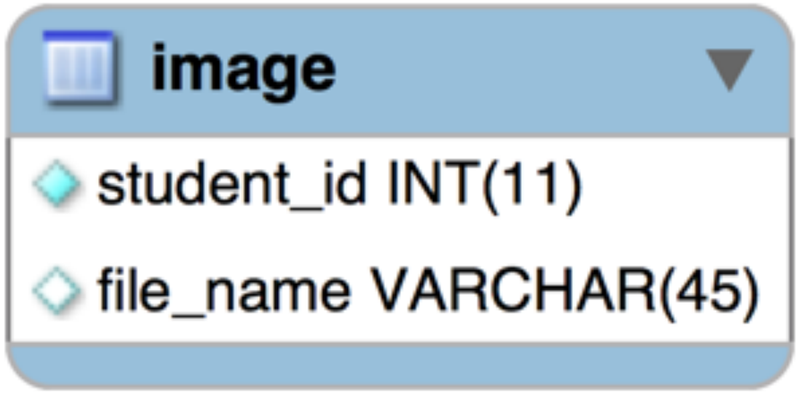


More Info - Mapping the Collection

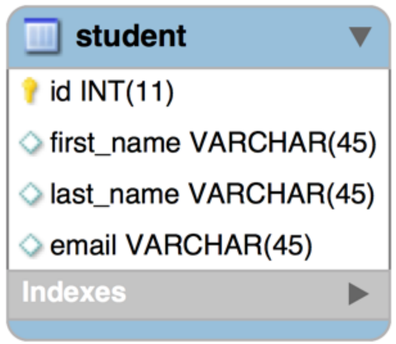
```
@Entity
@Table(name="student")
public class Student {

  @ElementCollection
  @CollectionTable(
      name="image",
      joinColumns= @JoinColumn(name="student_id"))
  @Column(name="file_name")
  private Set<String> images = new HashSet<String>();
```

Tells Hibernate
we are mapping a collection



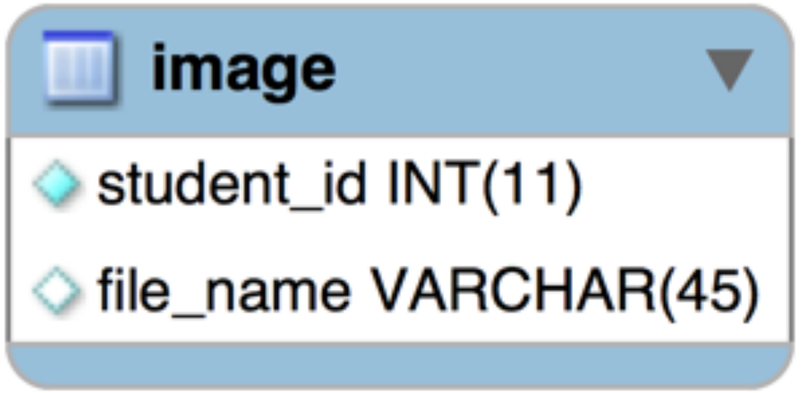


More Info - Mapping the Collection




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



  @ElementCollection
  @CollectionTable(
      name="image",
      joinColumns= @JoinColumn(name="student_id"))
  @Column(name="file_name")
  private Set<String> images = new HashSet<String>();
```

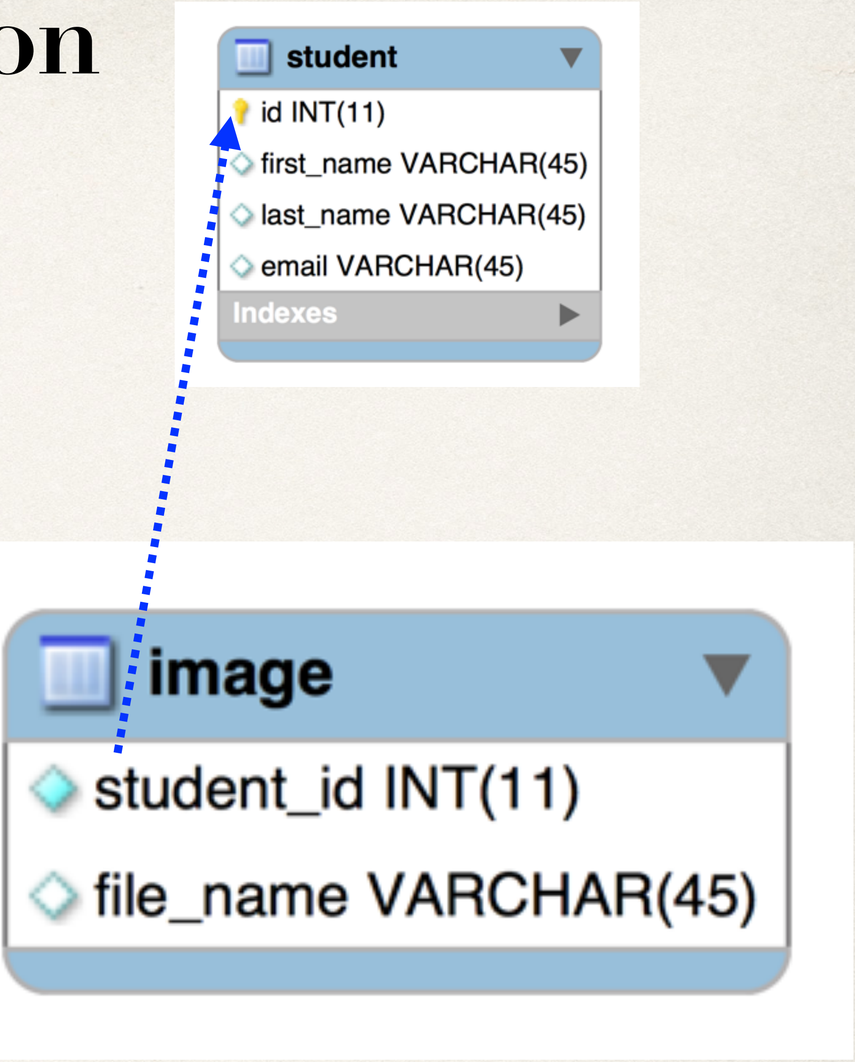
The name of the collection table



More Info - Mapping the Collection

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




  @ElementCollection
  @CollectionTable(
      name="image",
      joinColumns= @JoinColumn(name="student_id"))
  @Column(name="file_name")
  private Set<String> images = new HashSet<String>();
```

The column to join on

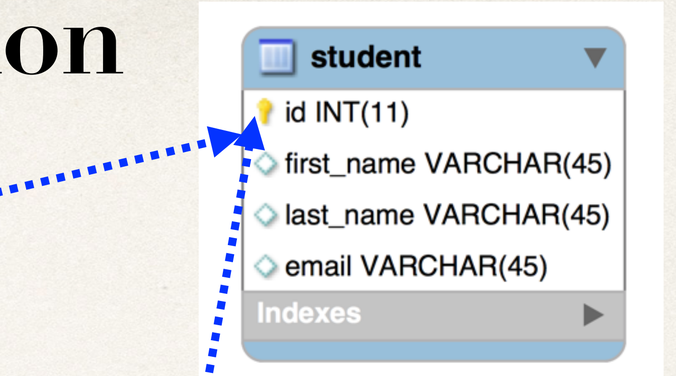


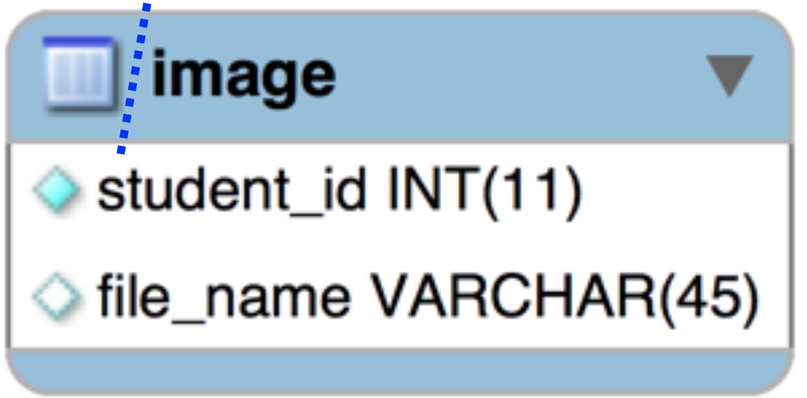
More Info - Mapping the Collection

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

   @Id
   @GeneratedValue(strategy=GenerationType.IDENTITY)
   private int id;

   @ElementCollection
   @CollectionTable(
       name="image",
       joinColumns= @JoinColumn(name="student_id"))
   @Column(name="file_name")
   private Set<String> images = new HashSet<String>();
```

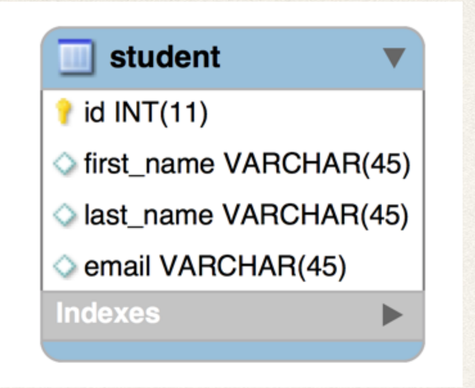
Based on our primary key
@Id: id

The column
to join on

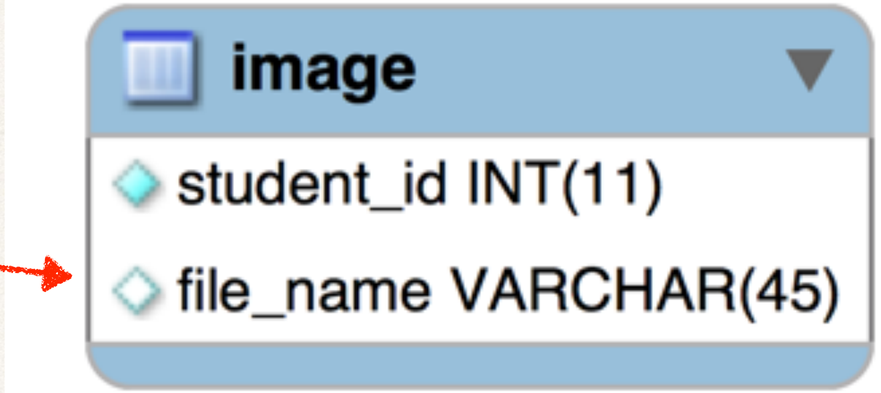




More Info - Mapping the Collection



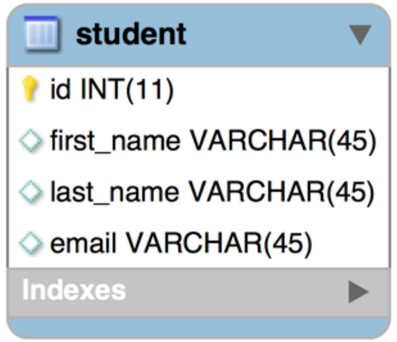

```
@Entity
@Table(name="student")
public class Student {

  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private int id;

  @ElementCollection
  @CollectionTable(
      name="image",
      joinColumns= @JoinColumn(name="student_id"))
  @Column(name="file_name")
  private Set<String> images = new HashSet<String>();
```

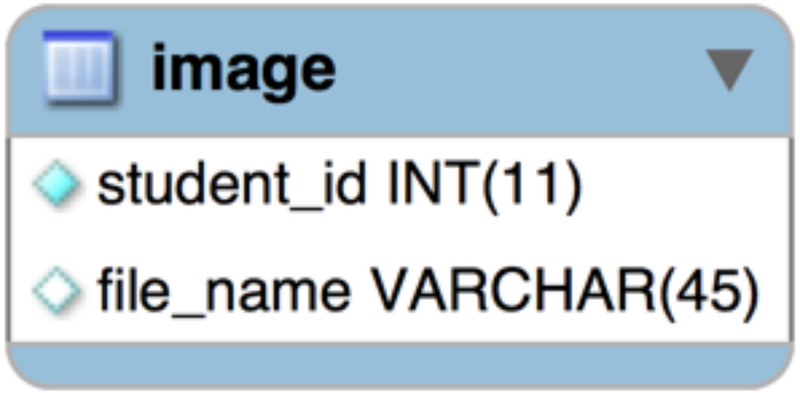


The column for the image file name

More Info - Mapping the Collection

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

  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private int id;

  @ElementCollection
  @CollectionTable(
      name="image",
      joinColumns= @JoinColumn(name="student_id"))
  @Column(name="file_name")
  private Set<String> images = new HashSet<String>();
```





More on @ElementCollection

More on @ElementCollection

- **`@ElementCollection`** can be used to define relationships

More on @ElementCollection

- **`@ElementCollection`** can be used to define relationships
  - One-to-many relationship to an **`@Embeddable`** object

More on @ElementCollection

We'll cover this later in the course

- **`@ElementCollection`** can be used to define relationships
  - One-to-many relationship to an **`@Embeddable`** object

More on @ElementCollection

- **`@ElementCollection`** can be used to define relationships
  - One-to-many relationship to an **`@Embeddable`** object
  - One-to-many relationship to a *Basic* object, such as

More on @ElementCollection

- **`@ElementCollection`** can be used to define relationships
  - One-to-many relationship to an **`@Embeddable`** object
  - One-to-many relationship to a *Basic* object, such as
    - Java primitives (wrappers): `int`, `Integer`, `Double`, etc ...

More on @ElementCollection

- **`@ElementCollection`** can be used to define relationships
  - One-to-many relationship to an **`@Embeddable`** object
  - One-to-many relationship to a *Basic* object, such as
    - Java primitives (wrappers): `int`, `Integer`, `Double`, etc ...
    - `Date`, `String`, etc ...

More on @ElementCollection

- **`@ElementCollection`** can be used to define relationships
  - One-to-many relationship to an **`@Embeddable`** object
  - One-to-many relationship to a *Basic* object, such as
    - Java primitives (wrappers): `int`, `Integer`, `Double`, etc ...
    - `Date`, `String`, etc ...

```
private Set<String> images = new HashSet<String>();
```

Hmmm …

How does this compare to @OneToMany???!!!

Compare @ElementCollection to @OneToMany

Compare @ElementCollection to @OneToMany

- Similar to **`@OneToMany`** except target object is not an **`@Entity`**

Compare @ElementCollection to @OneToMany

- Similar to **`@OneToMany`** except target object is not an **`@Entity`**

Parent object:
Student

Compare @ElementCollection to @OneToMany

- Similar to **`@OneToMany`** except target object is not an **`@Entity`**

Parent object:
Student

Target object:
Collection of Strings

Compare @ElementCollection to @OneToMany

- Similar to **`@OneToMany`** except target object is not an **`@Entity`**

Parent object:
Student

Target object:
Collection of Strings

```
private Set<String> images = new HashSet<String>();
```

Compare @ElementCollection to @OneToMany

- Similar to **`@OneToMany`** except target object is not an **`@Entity`**

Parent object:
Student

Target object:
Collection of Strings

```
private Set<String> images = new HashSet<String>();
```

- Easy way to define a collection with simple/basic objects

Limitations on @ElementCollection

Limitations on @ElementCollection

- With `@ElementCollection`

Limitations on @ElementCollection

- With **`@ElementCollection`**
  - You can't query, persist or merge target objects independently of their parent object

Limitations on @ElementCollection

- With **`@ElementCollection`**
  - You can't query, persist or merge target objects independently of their parent object

- **`@ElementCollection`** does not support a cascade option

Limitations on @ElementCollection

- With **`@ElementCollection`**
  - You can't query, persist or merge target objects independently of their parent object

- **`@ElementCollection`** does not support a cascade option
  - Target objects are ALWAYS persisted, merged, removed with their parent object

Comparison

Comparison

	@ElementCollection	@OneToMany

Comparison

	@ElementCollection	@OneToMany
Query target objects independently of parent object	No	Yes

Comparison

	@ElementCollection	@OneToMany
Query target objects independently of parent object	No	Yes
Supports fine-grained cascading option	No	Yes

Comparison

Easy way to define a collection with simple/basic objects

	@ElementCollection	@OneToMany
Query target objects independently of parent object	No	Yes
Supports fine-grained cascading option	No	Yes