luv2code

Mapping - Sorted Sets

Sorted **Set**

Collection of items that contains no duplicates

When retrieved, sorted by a given field

Use Case for Sorted Sets

Use Case for Sorted Sets

- Useful when
  - Interested in membership … for yes/no decisions

Use Case for Sorted Sets

- Useful when
  - Interested in membership … for yes/no decisions
  - AND order during retrieval is important

Use Case for Sorted Sets

- Useful when
  - Interested in membership … for yes/no decisions
  - AND order during retrieval is important
- Examples

Use Case for Sorted Sets

- Useful when
  - Interested in membership … for yes/no decisions
  - AND order during retrieval is important
- Examples
  - Waiting list: Who is the next customer in line?

Use Case for Sorted Sets

- Useful when
  - Interested in membership … for yes/no decisions
  - AND order during retrieval is important

- Examples
  - Waiting list: Who is the next customer in line?
  - Student ranking: Which student has the highest grade?

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 want to retrieve the images in alphabetical order (descending)

Student and Images

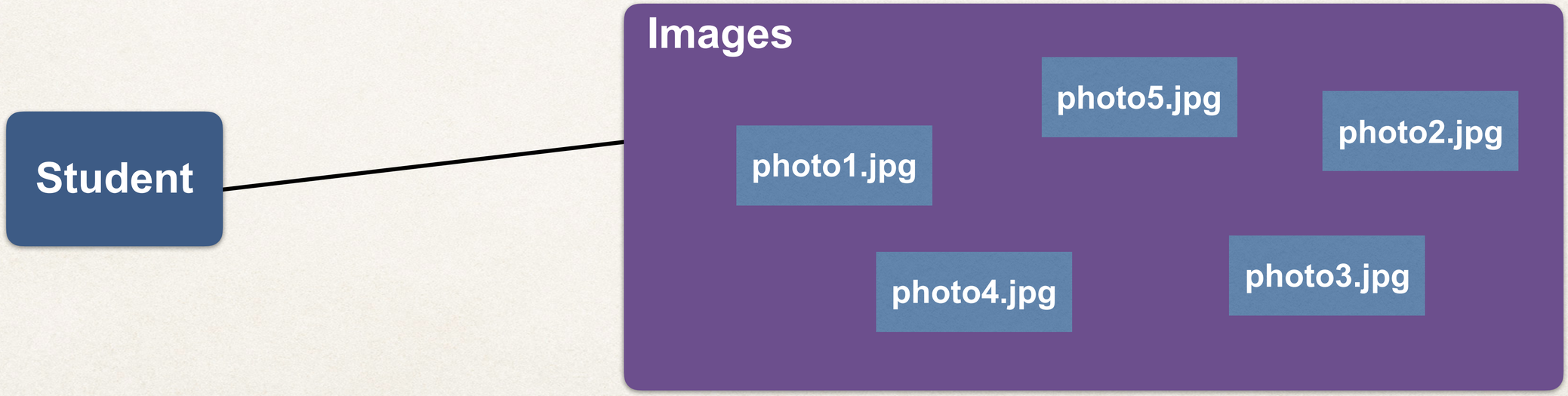
- A student will have a ***set*** of images (image file names)
  - We want to retrieve the images in alphabetical order (descending)
  - There should not be any duplicate images

Student and Images

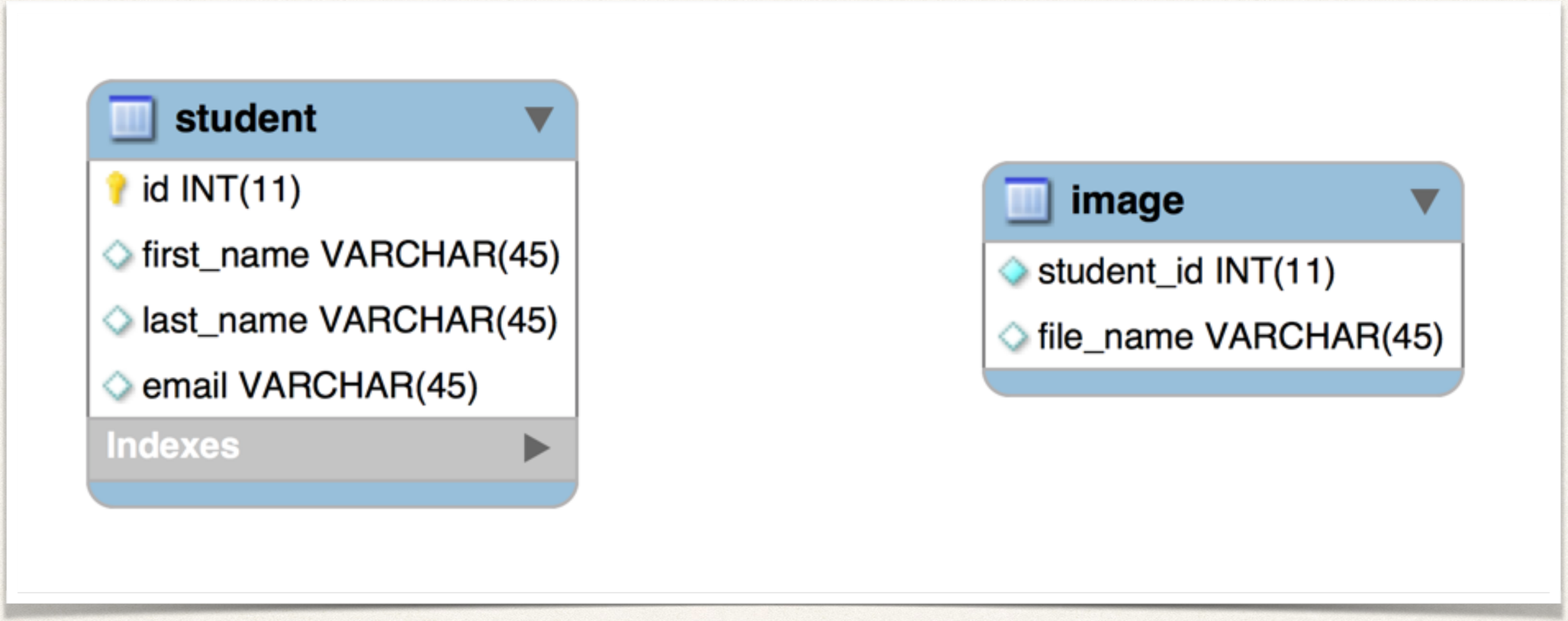
- A student will have a ***set*** of images (image file names)
  - We want to retrieve the images in alphabetical order (descending)
  - There should not be any duplicate images

Student and Images

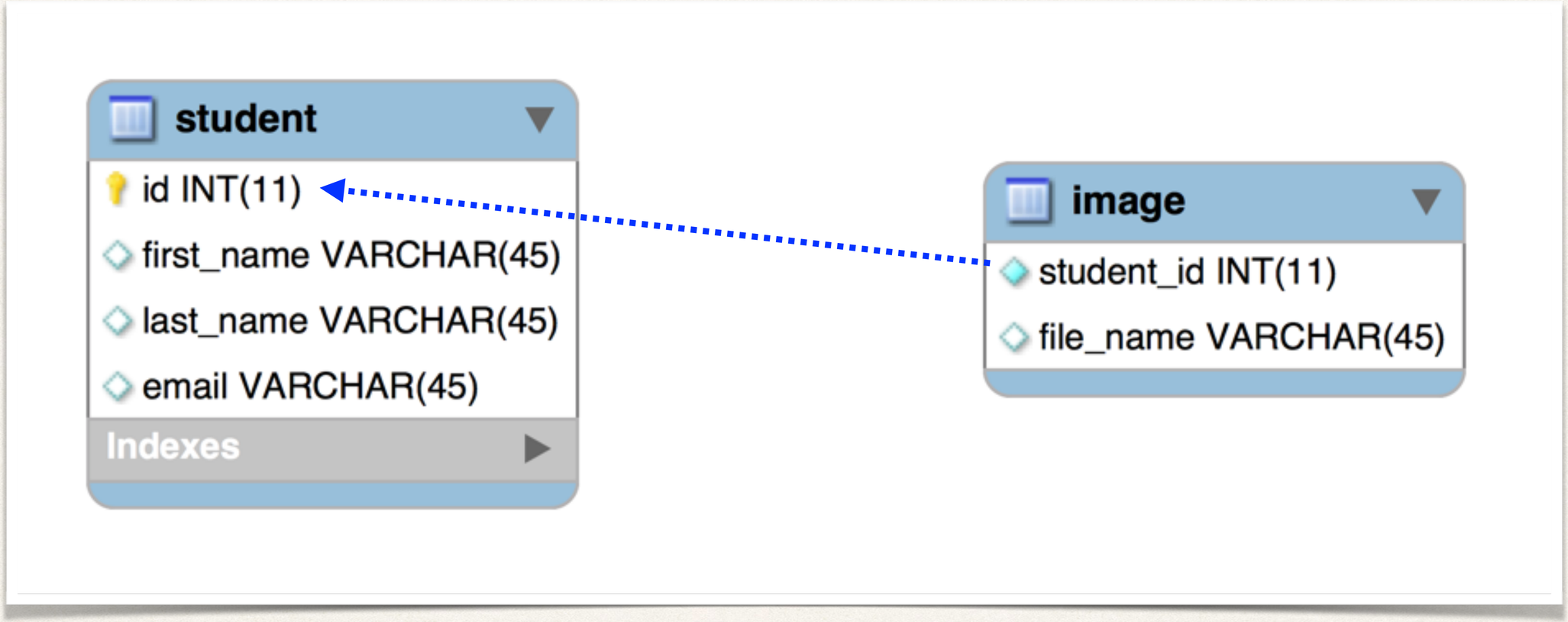
- A student will have a ***set*** of images (image file names)
  - We want to retrieve the images in alphabetical order (descending)
  - There should not be any duplicate images



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. Map the element collection

Development Process

Step-By-Step

1. Create database tables

2. Map the element collection

3. Develop the main application

Step 1: Create database tables

Step 1: Create database tables

- For ease of development and testing, we'll use auto configuration

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

Step 1: Create database tables

- For ease of development and testing, we'll use auto configuration

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

Step 1: Create database tables

- For ease of development and testing, we'll use auto configuration

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

- For **update**

Step 1: Create database tables

- For ease of development and testing, we'll use auto configuration

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

- For **update**
  - If tables do not exist, then **create** the tables

Step 1: Create database tables

- For ease of development and testing, we'll use auto configuration

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

- For **update**
  - If tables do not exist, then **create** the tables
  - If there are changes to tables (from Java annotations)

Step 1: Create database tables

- For ease of development and testing, we'll use auto configuration

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

- For **update**
  - If tables do not exist, then **create** the tables
  - If there are changes to tables (from Java annotations)
    - perform an **update** on table schema(s)

Step 1: Create database tables

- For ease of development and testing, we'll use auto configuration

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

- For **`update`**
  - If tables do not exist, then **create** the tables
  - If there are changes to tables (from Java annotations)
    - perform an **update** on table schema(s)

Note: Tables are NEVER dropped.

So you get to keep your existing data.

Very useful when you want to run your app multiple times and keep existing data.

Step 1: Create database tables

- For ease of development and testing, we'll use auto configuration

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

- For **`update`**
  - If tables do not exist, then **create** the tables
  - If there are changes to tables (from Java annotations)
    - perform an **update** on table schema(s)

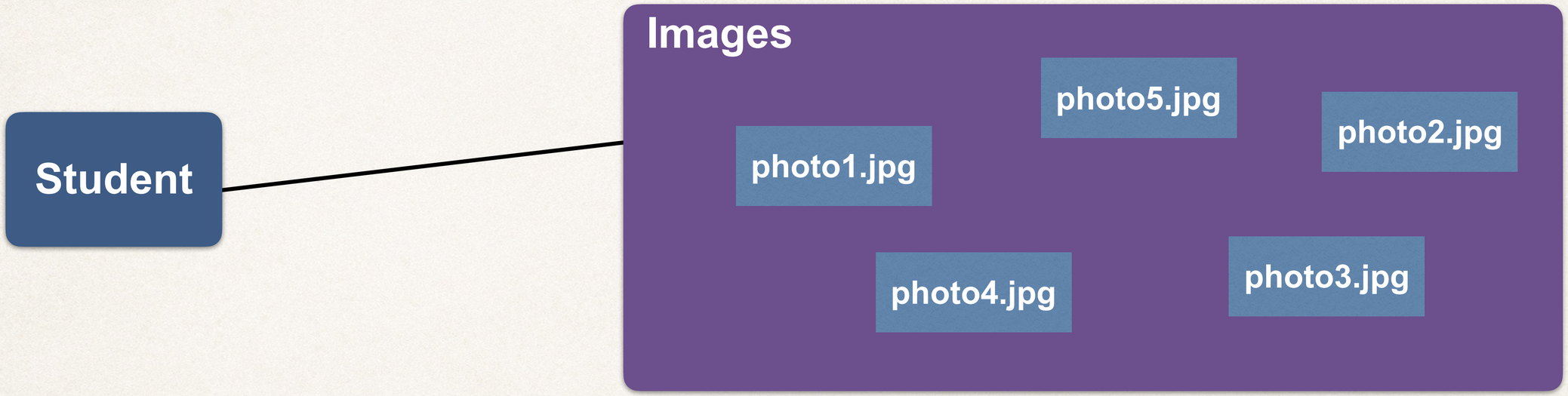
Note: Tables are NEVER dropped.

So you get to keep your existing data.

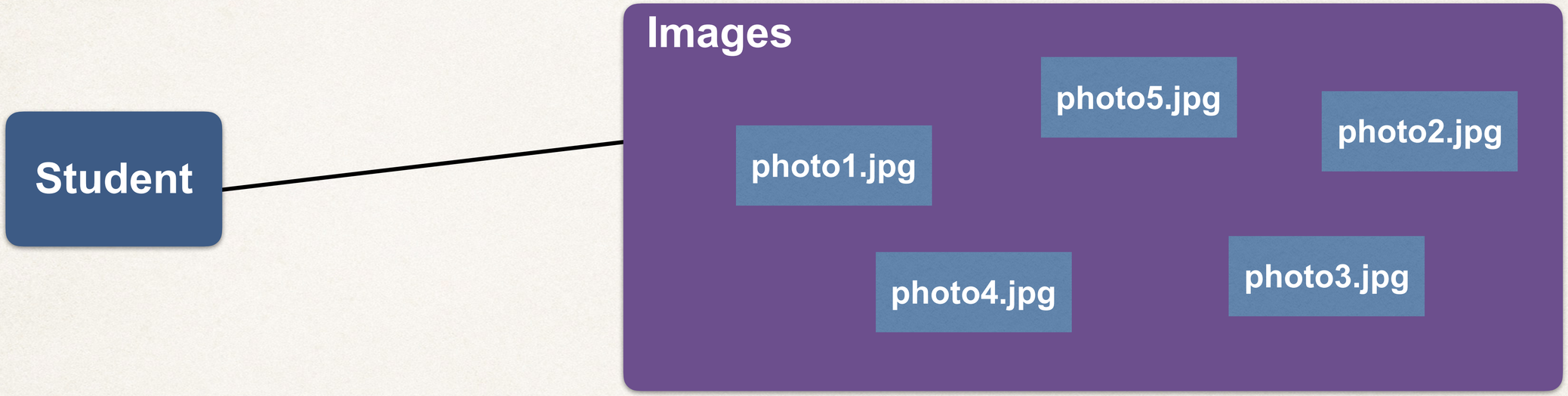
Very useful when you want to run your app multiple times and keep existing data.

Step 2: Map the element collection

Step 2: Map the element collection



Step 2: Map the element collection




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

Annotation for Ordering

Annotation for Ordering

Annotation	Description

Annotation for Ordering

Annotation	Description
`@OrderBy`	**Specifies the ordering of the elements when a collection is retrieved.**

Annotation for Ordering

Annotation	Description
`@OrderBy`	**Specifies the ordering of the elements when a collection is retrieved.**

Syntax: `@OrderBy("[field name or property name] [ASC |DESC]")`

Annotation for Ordering

Annotation	Description
`@OrderBy`	**Specifies the ordering of the elements when a collection is retrieved.**

Syntax: `@OrderBy("[field name or property name] [ASC |DESC]")`

Example: `@OrderBy("file_name DESC")`

Annotation for Ordering

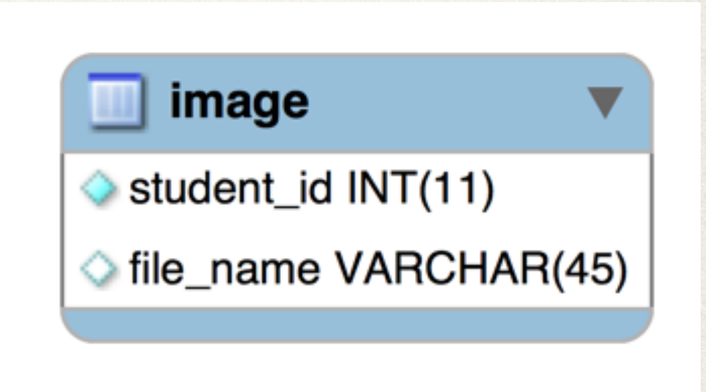
Annotation	Description
`@OrderBy`	**Specifies the ordering of the elements when a collection is retrieved.**

Syntax: `@OrderBy("[field name or property name] [ASC |DESC]")`

Example: `@OrderBy("file_name DESC")`

If ASC or DESC is not specified then ASC is the default

Mapping the Collection

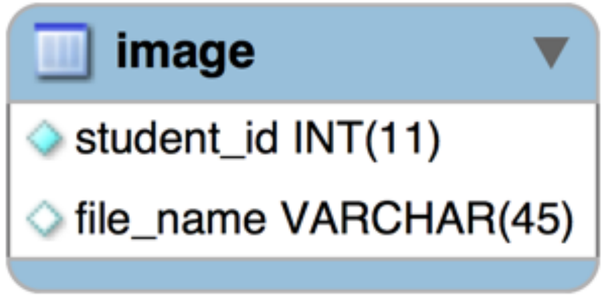


Mapping the Collection

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

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

  …
}
```

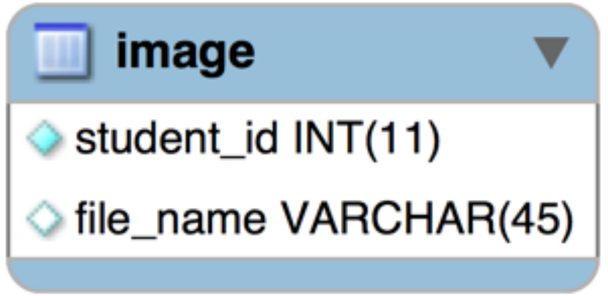


Mapping the Collection

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

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

  …
}
```



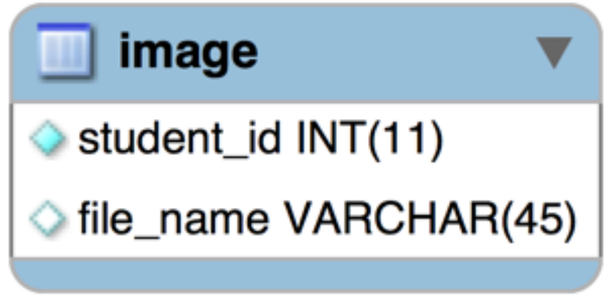
Mapping the Collection

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

  …

  @ElementCollection
  @CollectionTable(name="image")
  @OrderBy("file_name DESC")
  @Column(name="file_name")
  private Set<String> images = new LinkedHashSet<String>();


  …
}
```



Mapping the Collection

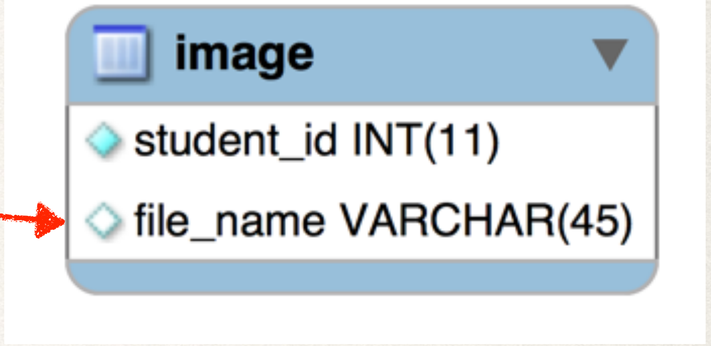
```
@Entity
@Table(name="student")
public class Student {

  …

  @ElementCollection
  @CollectionTable(name="image")
  @OrderBy("file_name DESC")
  @Column(name="file_name")
  private Set<String> images = new LinkedHashSet<String>();


  …
}
```

Sort by file_name when the images are retrieved



Step 3: Develop the main application

Step 3: Develop the main application

CREATE THE IMAGES

Step 3: Develop the main application

CREATE THE IMAGES

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

Step 3: Develop the main application

CREATE THE IMAGES

```
…
// 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 3: Develop the main application

CREATE THE IMAGES

```
…
// 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");
theImages.add("photo4.jpg");
theImages.add("photo5.jpg");
```

Now, let's add to the images collection

Step 3: Develop the main application

CREATE THE IMAGES

```
…
// 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");
theImages.add("photo4.jpg");
theImages.add("photo5.jpg");

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

Step 3: Develop the main application

CREATE THE IMAGES

```
…
// 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");
theImages.add("photo4.jpg");
theImages.add("photo5.jpg");

// start a transaction
session.beginTransaction();

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

Step 3: Develop the main application

Step 3: Develop the main application

RETRIEVE THE IMAGES

Step 3: Develop the main application

RETRIEVE THE IMAGES

```
…
// get the student id
int theId = 1;
Student student = session.get(Student.class, theId);
```

Step 3: Develop the main application

RETRIEVE THE IMAGES

```
…
// get the student id
int theId = 1;
Student student = session.get(Student.class, theId);

// print the student detail
System.out.println("Student details: " + student);
```

Step 3: Develop the main application

RETRIEVE THE IMAGES

```
…
// get the student id
int theId = 1;
Student student = session.get(Student.class, theId);

// print the student detail
System.out.println("Student details: " + student);

// print the associated images
System.out.println("The associated images: " + student.getImages());
…
```

Run the App: Retrieve Student Images

Run the App: Retrieve Student Images

Table: student

id	email	first_name	last_name
1	john@luv2code.com	John	Doe

Run the App: Retrieve Student Images

Table: student

id	email	first_name	last_name
1	john@luv2code.com	John	Doe

Table: image

Student_id	file_name
1	photo1.jpg
1	photo2.jpg
1	photo3.jpg
1	photo4.jpg
1	photo5.jpg

Run the App: Retrieve Student Images

Table: student

id	email	first_name	last_name
1	john@luv2code.com	John	Doe

Table: image

Student_id	file_name
1	photo1.jpg
1	photo2.jpg
1	photo3.jpg
1	photo4.jpg
1	photo5.jpg

Console output

```
Hibernate: select images0_.Student_id as Student_1_0_0_, images0_.file_name as file_nam2_0_0_ from image images0_
where images0_.Student_id=? order by images0_.file_name desc

The associated images: [photo5.jpg, photo4.jpg, photo3.jpg, photo2.jpg, photo1.jpg]
```

Descending order

Run the App: Retrieve Student Images

Table: student

id	email	first_name	last_name
1	john@luv2code.com	John	Doe

Table: image

Student_id	file_name
1	photo1.jpg
1	photo2.jpg
1	photo3.jpg
1	photo4.jpg
1	photo5.jpg

Console output

```
Hibernate: select images0_.Student_id as Student_1_0_0_, images0_.file_name as file_nam2_0_0_ from image images0_
where images0_.Student_id=? order by images0_.file_name desc

The associated images: [photo5.jpg, photo4.jpg, photo3.jpg, photo2.jpg, photo1.jpg]
```

Descending order

```
@ElementCollection
@CollectionTable(name="image")
@OrderBy("file_name DESC")
@Column(name="file_name")
private Set<String> images = new LinkedHashSet<String>();
```