**Spring JPA Hibernate Mapping-Better-Way-2022**

**One To One Unidirectional**

@Entity(name = "Item\_O2OU")

@Table(name = "item\_o2ou")

@Getter @Setter

**public** **class** Item {

@Id @GeneratedValue

**private** **long** id;

@Column(name = "name")

**private** String name;

}

@Entity(name = "ShoppingCart\_O2OU")

@Table(name = "shopping\_cart\_o2ou")

@Getter @Setter

**public** **class** ShoppingCart {

@Id @GeneratedValue

**private** **long** id;

@Column(name = "name")

**private** String name;

**@OneToOne(cascade = CascadeType.*ALL*)**

**private** Item item;

}

**Database Table Design**

Graphical user interface, text

Description automatically generated Graphical user interface, application

Description automatically generated

**One To One Bidirectional**

@Entity(name = "Item\_O2OB")

@Table(name = "item\_o2ob")

@Getter @Setter

**public** **class** Item {

@Id @GeneratedValue

**private** **long** id;

@Column(name = "name")

**private** String name;

**@OneToOne(mappedBy = "item")**

**private** ShoppingCart cart;

}

@Entity(name = "ShoppingCart\_O2OB")

@Table(name = "shopping\_cart\_o2ob")

@Getter @Setter

**public** **class** ShoppingCart {

@Id @GeneratedValue

**private** **long** id;

@Column(name = "name")

**private** String name;

**@OneToOne(cascade = CascadeType.*ALL*)**

**private** Item item;

}

**Database Table Design**

Graphical user interface, text

Description automatically generated Graphical user interface

Description automatically generated

You can also mentioned mappedBy in ShoppingCart class also.

**One To Many Unidirectional**

@Entity(name = "Item\_O2MU")

@Table(name = "item\_o2mu")

@Getter @Setter

**public** **class** Item {

@Id @GeneratedValue

**private** **long** id;

@Column(name = "name")

**private** String name;

}

@Entity(name = "ShoppingCart\_O2MU")

@Table(name = "shopping\_cart\_o2mu")

@Getter @Setter

**public** **class** ShoppingCart {

@Id @GeneratedValue

**private** **long** id;

@Column(name = "name")

**private** String name;

**@OneToMany(cascade = CascadeType.*ALL*)**

**private** Set<Item> items =

**new** HashSet<Item>();

}

**Database Table Design**

Graphical user interface, application

Description automatically generated Text

Description automatically generated with low confidence Table

Description automatically generated

**One To Many Bidirectional**

@Entity(name = "ShoppingCart\_O2MB")

@Table(name = "shopping\_cart\_o2mb")

@Getter

@Setter

**public** **class** ShoppingCart {

@Id

@GeneratedValue

**private** **long** id;

@Column(name = "name")

**private** String name;

**@OneToMany(cascade = CascadeType.*ALL*,**

**mappedBy = "shoppingCart")**

**private** Set<Item> items =

**new** HashSet<Item>();

}

@Entity(name = "Item\_O2MB")

@Table(name = "item\_o2mb")

@Getter @Setter

**public** **class** Item {

@Id @GeneratedValue

**private** **long** id;

@Column(name = "name")

**private** String name;

**@ManyToOne**

**private** ShoppingCart shoppingCart;

}

**Database Table Design**

Graphical user interface, text

Description automatically generated Table

Description automatically generated

**Many 2 Many Unidirectional**

@Entity(name="Author") @Table(name = "author")

@Getter @Setter

@ToString(exclude = "books")

@NoArgsConstructor

**public** **class** Author {

@Id

@GeneratedValue

**private** Long id;

**private** String name; 🡸 Author Class

**public** Author(String name) {

**this**.name = name;

}

@ManyToMany(cascade = CascadeType.***ALL***)

@JoinTable(name = "author\_book",

joinColumns = @JoinColumn(name = "author\_id"),

inverseJoinColumns = @JoinColumn(name = "book\_id"))

**private** Set<Book> books;

}

@Entity(name="Book") @Table(name = "book")

@Getter @Setter

@NoArgsConstructor

@ToString

**public** **class** Book {

@Id

@GeneratedValue

**private** Long id; 🡸 Book class

**private** String title;

**public** Book(String title) {

**this**.title = title;

}

}

@Repository

**public** **interface** AuthorRepository **extends** CrudRepository<Author, Long> {

}

Testing Method

**public** **void** saveAllAuthors() {

Author author1 = **new** Author("Author-1");

Set<Book> bookSet1 = Set.*of*(**new** Book("Book-1"), **new** Book("Book-2"), **new** Book("Book-3"));

author1.setBooks(bookSet1);

Author author2 = **new** Author("Author-2");

Set<Book> bookSet2 = Set.*of*(**new** Book("Book-4"), **new** Book("Book-5"), **new** Book("Book-6"));

author2.setBooks(bookSet2);

List<Author> authorList = List.*of*(author1,author2);

authorRepo.saveAll(authorList);

}

**public** **void** showAllAuthorAndBook() {

Author author1 = authorRepo.findById(52L).get();

System.***out***.println("Author Details: "+author1);

author1.getBooks().forEach(book -> System.***out***.println("Book: "+book));

}

**Database Table Design**

|  |  |
| --- | --- |
| **id** | **title** |
| 1 | Book-2 |
| 2 | Book-3 |
| 3 | Book-1 |
| 4 | Book-6 |
| 5 | Book-4 |
| 6 | Book-5 |

|  |  |
| --- | --- |
| **author\_id** | **book\_id** |
| 52 | 1 |
| 52 | 2 |
| 52 | 3 |
| 53 | 4 |
| 53 | 5 |
| 53 | 6 |

Author Book author\_book

|  |  |
| --- | --- |
| **id** | **name** |
| 52 | Author-1 |
| 53 | Author-2 |

Book

To make a single query to avoid N+1 issue, you can modify like this.

@Repository

**public** **interface** AuthorRepository **extends** CrudRepository<Author, Long> {

@Query("select a from Author a join fetch a.books")

List<Author> getAuthorById(Long id);

}

Testing method

**public** **void** showAuthorAndBook() {

List<Author> authors = authorRepo.getAuthorById(52L);

System.***out***.println("Author Details: "+authors);

authors.forEach( author -> {

author.getBooks().forEach( book -> System.***out***.println("Book: "+book));

});

}

**Many 2 Many Bidirectional**

@Entity(name="Author") @Table(name = "author")

@Getter @Setter @ToString(exclude = "books")

@NoArgsConstructor

**public** **class** Author {

@Id

@GeneratedValue

**private** Long id;

**private** String name;

**public** Author(String name) {

**this**.name = name;

}

@ManyToMany(cascade = CascadeType.***ALL***)

@JoinTable(name = "author\_book",

joinColumns = @JoinColumn(name = "author\_id"),

inverseJoinColumns = @JoinColumn(name = "book\_id"))

**private** Set<Book> books;

}

@Entity(name="Book")@Table(name = "book")

@Getter @Setter

@NoArgsConstructor

@ToString

**public** **class** Book {

@Id

@GeneratedValue

**private** Long id;

**private** String title;

**public** Book(String title) {

**this**.title = title;

}

@ManyToMany(fetch = FetchType.***LAZY***, cascade = CascadeType.***ALL***, mappedBy = "books")

**private** Set<Author> authors = **new** HashSet<>();

}

@Repository

**public** **interface** AuthorRepository **extends** CrudRepository<Author, Long> {

}

**Testing Methods**

**public** **void** saveAllAuthors() {

Author author1 = **new** Author("Author-1");

Author author2 = **new** Author("Author-2");

Set<Author> authorSet = **new** HashSet<>();

authorSet.add(author1); authorSet.add(author2);

Set<Book> bookSet1 = **new** HashSet<>();

Book book1 = **new** Book("Book-1");

book1.setAuthors(authorSet);

author1.setBooks(bookSet1);

bookSet1.add(book1);

authorRepo.saveAll(authorSet);

}

**public** **void** showAllAuthorAndBook() {

Author author1 = authorRepo.findById(2L).get();

System.***out***.println("Author Details: "+author1);

author1.getBooks().forEach(book -> System.***out***.println("Book: "+book));

}

**Database Table Design**

|  |  |
| --- | --- |
| **Table Name: author** | |
| **id** | **name** |
| 1 | Author-2 |
| 2 | Author-1 |

|  |  |
| --- | --- |
| **Table Name: book** | |
| **id** | **title** |
| 1 | Book-1 |

|  |  |
| --- | --- |
| **Table Name: author\_book** | |
| **author\_id** | **book\_id** |
| 2 | 1 |

Similarly, you can also define @JoinTable, columns and inverse join columns in book class. The code is given below.

@Entity(name="Author")

@Table(name="author")

@Getter @Setter @ToString(exclude="books") @NoArgsConstructor

**public** **class** Author {

@Id @GeneratedValue

**private** Long id;

**private** String name;

@ManyToMany(cascade = CascadeType.***ALL***, mappedBy="authors")

**private** Set<Book> books;

**public** Author(String name) {

**this**.name = name;

}

}

@Entity(name = "Book")

@Table(name = "book")

@Getter

@Setter

@ToString(exclude = "authors")

@NoArgsConstructor

**public** **class** Book {

@Id

@GeneratedValue

**private** Long id;

**private** String name;

@ManyToMany(cascade = CascadeType.***ALL***/\* ,mappedBy = "books" \*/)

@JoinTable(name = "book\_author",

joinColumns = @JoinColumn(name = "book\_id"),

inverseJoinColumns = @JoinColumn(name = "author\_id"))

**private** Set<Author> authors;

**public** Book(String name) {

**this**.name = name;

}

}

|  |  |
| --- | --- |
| **Table Name: book\_author** | |
| **book\_id** | **author\_id** |
| 1 | 1 |
| 1 | 2 |

**Database Tables**

|  |  |
| --- | --- |
| **Table Name: author** | |
| **id** | **name** |
| 1 | Author-2 |
| 2 | Author-1 |

|  |  |
| --- | --- |
| **Table Name: book** | |
| **id** | **name** |
| 1 | Book-1 |

**JPA - Bidirectional OneToMany/ManyToOne**

**A quick overview of bidirectional one-to-many/many-to-one relationship in JPA**

* In bidirectional one-to-many/many-to-one relationship, the target side has a reference back to the source entity as well.
* The annotation @OneToMany is used on the side which has the collection reference.
* The annotation @ManyToOne is used on the side which has the single-valued back reference.
* We must use 'mappedBy' element of the @OneToMany annotations to specify that the corresponding table will be the parent table. In other words the other side (which has @ManyToOne) will be the foreign-key table (child table).
* The value of 'mappedBy' element should be the name of the reference variable used in the other class's back reference.
* The side which has 'mappedBy' specified, will be the target entity of the relationship and corresponding table will be the parent of the relationship .
* The side which doesn't have 'mappedBy' element will be the source (owner) and the corresponding table will be the child of the relationship, i.e. it will have the foreign key column.

Example

One ClassRoom has many Students – One To Many

Many Students correspond to a ClassRoom – Many To One

**import** jakarta.persistence.CascadeType;

**import** jakarta.persistence.Column;

**import** jakarta.persistence.Entity;

**import** jakarta.persistence.GeneratedValue;

**import** jakarta.persistence.Id;

**import** jakarta.persistence.OneToMany;

**import** jakarta.persistence.Table;

**import** lombok.Getter;

**import** lombok.Setter;

@Entity

@Table(name = "class\_room")

@Getter

@Setter

**public** **class** ClassRoom {

@Id

@GeneratedValue

**private** **long** id;

@Column(name = "name")

**private** String name;

**@OneToMany(mappedBy = "clsRoom", cascade = CascadeType.*ALL*)**

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

}

**import** jakarta.persistence.CascadeType;

**import** jakarta.persistence.Column;

**import** jakarta.persistence.Entity;

**import** jakarta.persistence.GeneratedValue;

**import** jakarta.persistence.Id;

**import** jakarta.persistence.ManyToOne;

**import** jakarta.persistence.Table;

**import** lombok.Getter;

**import** lombok.Setter;

@Entity

@Table(name = "student")

@Getter

@Setter

**public** **class** Student {

@Id

@GeneratedValue

**private** **long** id;

@Column(name = "name")

**private** String name;

@ManyToOne(cascade = CascadeType.***ALL***)

**private** ClassRoom clsRoom;

@Override

**public** **int** hashCode() {

**return** Objects.*hash*(clsRoom, name);

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Student other = (Student) obj;

**return** Objects.*equals*(clsRoom, other.clsRoom) && Objects.*equals*(name, other.name);

}

}

AutoRun classs

@Component

**public** **class** AutoRun {

@Autowired

**private** SampleService service;

@EventListener(value = ApplicationReadyEvent.**class**)

**public** **void** run() {

System.***out***.println("Running ....");

saveAllClassRooms();

// saveClassRoom();

}

**private** **void** saveClassRoom() {

Set<Student> students = **new** HashSet<>();

ClassRoom clsRoom = **new** ClassRoom();

Student s = **new** Student();

s.setName("Johny Depp");

s.setClsRoom(clsRoom);

students.add(s);

clsRoom.setName("Room-2");

clsRoom.getStudents().add(s);

// service.saveClassRoom(clsRoom);

service.saveStudent(s);

System.***out***.println("Information saved successfully ...");

}

**private** **void** saveAllClassRooms() {

List<ClassRoom> clsRooms = **new** ArrayList<ClassRoom>();

**for** (**int** i = 0; i < 10; i++) {

ClassRoom room = **new** ClassRoom();

room.setName("Room-" + i);

clsRooms.add(room);

**for** (**int** j = 0; j < 10; j++) {

Student student = **new** Student();

student.setName("Stud-" + i + j);

room.getStudents().add(student);

student.setClsRoom(room);

}

}

service.saveAllClassRooms(clsRooms);

System.***out***.println("Information saved successfully ...");

}

}