**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 = "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;

}

@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>();

}

**Database Table Design**

Graphical user interface, text

Description automatically generated Table

Description automatically generated

Simplest Many-To-Many Unidirectional and Bidirectional Mapping

There is no need to use @JoinTable while creating Many-To-Many Unidirectional and Bidirectional mapping.

**Many-To-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;

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

**private Set<Book> books = new HashSet<>();**

**public** Author(String name) {

**this**.name = name;

}

}

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

@Getter @Setter @NoArgsConstructor

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

@Id

@GeneratedValue

**private** Long id;

**private** String name;

**public** Book(String name) {

**this**.name = name;

}

}

@Repository

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

List<Author> findAll();

}

|  |  |
| --- | --- |
| **Database Objects** | |
| **NAME** | **TYPE** |
| author\_seq | SEQUENCE |
| book\_seq | SEQUENCE |
| author | TABLE |
| author\_books | TABLE |
| book | TABLE |

|  |  |
| --- | --- |
| **Table: Author** | |
| **id** | **name** |
| 1 | Author-0 |
| 2 | Author-1 |

|  |  |
| --- | --- |
| **Table: Book** | |
| **id** | **name** |
| 1 | Book-0 |
| 2 | Book-0 |

|  |  |
| --- | --- |
| **Table: author\_books** | |
| **author\_id** | **books\_id** |
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |

**Many-To-Many Unidirectional Mapping Table Design**

**Testing Code**

// For Unidirectional

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

List<Author> authors = **new** ArrayList<>();

**for**( **int** i = 0; i < 5 ; i++ ) {

Author author = **new** Author("Author-"+i);

**for**(**int** j = 0; j < 5; j++) {

Book book = **new** Book("Book-"+i);

author.getBooks().add(book);

}

authors.add(author);

}

authorRepo.saveAll(authors);

}

**public** **void** showAuthorDetail() {

List<Author> authors = authorRepo.findAll();

authors.forEach( author -> {

System.***out***.println("Author Name: "+author.getName());

author.getBooks().forEach(book -> {

System.***out***.println("Book Name: "+book.getName());

});

});

}

**Many-To-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;

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

**private** Set<Book> books = **new** HashSet<>();

**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")

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

**public** Book(String name) {

**this**.name = name;

}

}

|  |  |
| --- | --- |
| **Database Objects** | |
| **NAME** | **TYPE** |
| author\_seq | SEQUENCE |
| book\_seq | SEQUENCE |
| author | TABLE |
| author\_books | TABLE |
| book | TABLE |

**Many-To-Many Bidirectional Mapping Table Design**

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

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

|  |  |
| --- | --- |
| **Table: author\_books** | |
| **authors\_id** | **books\_id** |
| 1 | 1 |

Here in the above book class, we have defined **mappedBy = "books"**. Hence the relational table name is author\_books because **author is the owning side** and book is the inverse side.

We can also define **mappedBy = "authors"** in author class. If we define mappedBy = "authors" in the author class, **book will be the owning side** and author will be inverse side. Let us check now.

@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 = **new** HashSet<>();

**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***)

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

**public** Book(String name) {

**this**.name = name;

}

}

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

|  |  |
| --- | --- |
| **Table: book\_authors** | |
| **books\_id** | **authors\_id** |
| 1 | 1 |
| 1 | 2 |

**Database Table Design**

|  |  |
| --- | --- |
| **Database objects** | |
| **NAME** | **TYPE** |
| author\_seq | SEQUENCE |
| book\_seq | SEQUENCE |
| author | TABLE |
| book | TABLE |
| book\_authors | TABLE |

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

**Testing Code**

//For Bidirectional

**public** **void** saveAllAuthorBooks() {

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);

}

Many-To-Many Unidirectional and Bidirectional Mapping Using @JoinTable

**Many 2 Many Unidirectional (Using @JoinTable)**

@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> {

}

**Database Table Design**

|  |  |
| --- | --- |
| **Table Name: author** | |
| **id** | **name** |
| 52 | Author-1 |
| 53 | Author-2 |

|  |  |
| --- | --- |
| **Table Name: book** | |
| **id** | **title** |
| 1 | Book-2 |
| 2 | Book-3 |

|  |  |
| --- | --- |
| **Table Name: author\_book** | |
| **author\_id** | **book\_id** |
| 52 | 1 |
| 52 | 2 |

**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));

}

**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));

});

}

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

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

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

**private** Set<Book> books;

**public** Author(String name) {

**this**.name = name;

}

}

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

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

@JoinTable(name = "book\_author", 🡸 **Book class we define @JoinTable**

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 |

**Many 2 Many Bidirectional (Using @JoinTable)**

@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", 🡸 **@JoinTable in Author class**

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 in the Book class, 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"), 🡸 **@JoinTable in Book class**

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

**private** Set<Author> authors;

**public** Book(String name) {

**this**.name = name;

}

}

**Database Table Design**

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

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

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

**Use of mappedBy in Bidirectional Mapping**

**In case of OneToOne – Bidirectional**

**public** **class** Author { // Use same type annotations

@Id @GeneratedValue

**private** Long id;

**private** String name;

@OneToOne(cascade = CascadeType.***ALL***, mappedBy="author")

**private** Book book;

**public** Author(String name) {

**this**.name = name;

}

}

**public** **class** Book { // Use same type annotations

@Id @GeneratedValue

**private** Long id;

**private** String name;

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

**private** Author author;

**public** Book(String name) {

**this**.name = name;

}

}

In the above case, it is defined **as mappedBy="author" in Author class**.

**Database Table**

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

|  |  |  |
| --- | --- | --- |
| **Table Name: Book** | | |
| **id** | **name** | **author\_id** |
| 2 | Book-1 | 2 |

If we define **mappedBy="book" in Book class**, the database table structure will be like below.

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

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

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

@Id @GeneratedValue

**private** Long id;

**private** String name;

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

**private** Book book;

**public** Author(String name) {

**this**.name = name;

}

}

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

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

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

@Id @GeneratedValue

**private** Long id;

**private** String name;

@OneToOne(cascade = CascadeType.***ALL***, mappedBy="book")

**private** Author author;

**public** Book(String name) {

**this**.name = name;

}

}

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

**Database Table**

|  |  |  |
| --- | --- | --- |
| **Table Name: Author** | | |
| **id** | **name** | **book\_id** |
| 1 | Author-1 | 1 |

**How to determine mappedBy value in the respective class ?**

In Author class, it has been defined as

**public** **class** Author { // Use same type annotations

@Id @GeneratedValue

**private** Long id;

**private** String name;

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

private Book bookAA; 🡸 Mark it

**public** Author(String name) {

**this**.name = name;

}

}

In Book Class, it is defined as

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

@Id @GeneratedValue

**private** Long id;

**private** String name;

@OneToOne(cascade = CascadeType.***ALL***, **mappedBy="bookAA"**) 🡸 Mark it

**private** Author author;

**public** Book(String name) {

**this**.name = name;

}

}

So in case mappedBy, the variable name used in the opposite class has to be used in the class where you are declaring mappedBy. It is visible from the above classes.