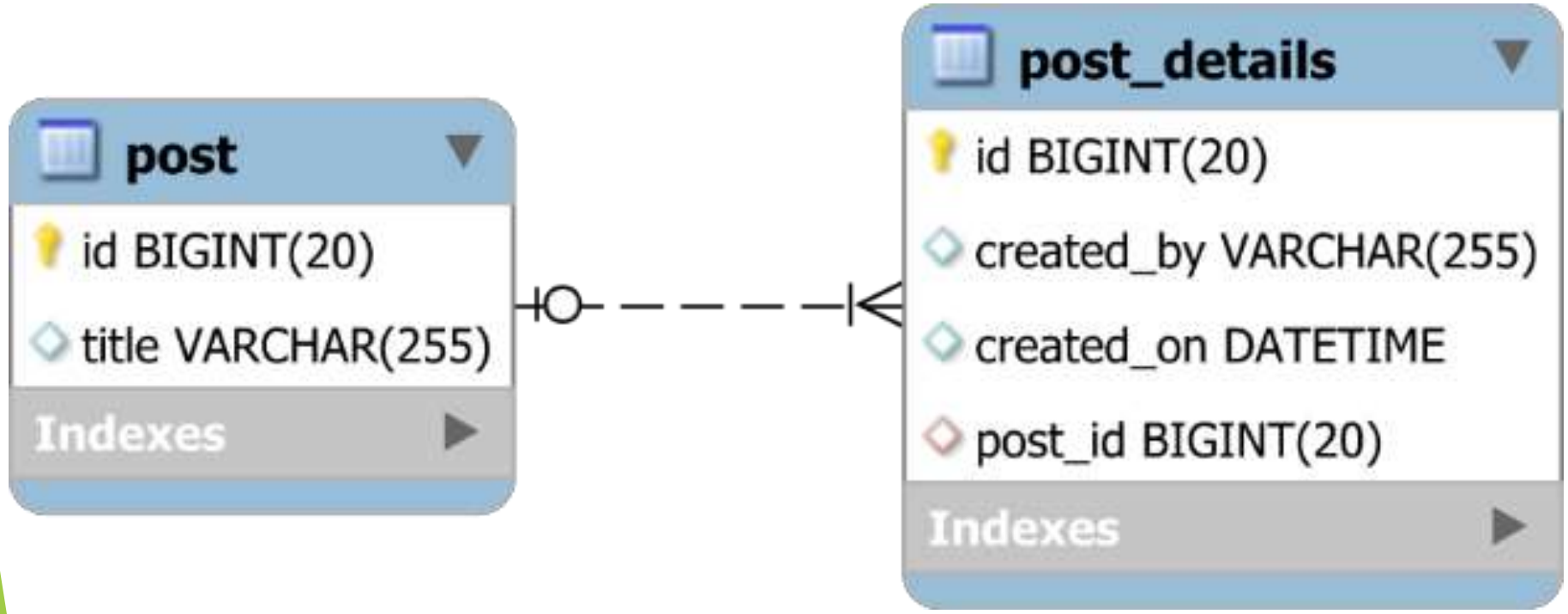
Hibernate Relation

Hibernate - Relationship Mapping

- One-to-One
- One-to-Many
- Many-to-One
- Many-to-Many

Hibernate - @oneToOne



@oneToOne

```
public class Post {
    // id, title
    @OneToOne(mappedBy = "post")
    private PostDetails details;
}

public class PostDetails {
    // id, createdOn, createdBy

    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "post_id")
    private Post post;
}
```

@JoinColumn

- The @JoinColumn annotation allows you to specify the Foreign Key column name

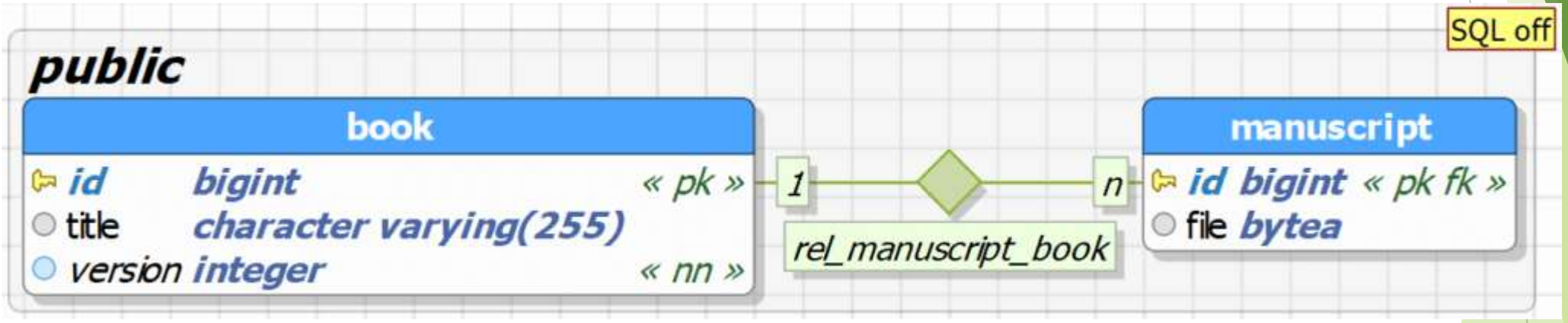
MappdeBy

- mappedBy signals hibernate that the key for the relationship is on the other side.
  This means that although you link 2 tables together, only 1 of those tables has a foreign key constraint to the other one. MappedBy allows you to still link from the table not containing the constraint to the other table.
- the value of *mappedBy* is the name of the association-mapping attribute on the owning side

  With this, we have now established a bidirectional association between our Post and PostComment entities.

@MapsId

- **Most efficient mapping of a one-to-one association**
- Sharing the primary key in OneToOne relation ship



@MapsId

```
@Entity
public class Book {

 @Id
 @GeneratedValue(strategy = GenerationType.SEQUENCE)
 @SequenceGenerator(name = "book_seq")
 private Long id;

private String title;

@OneToOne(mappedBy = "book")
 private Manuscript manuscript;
 ...
}
```

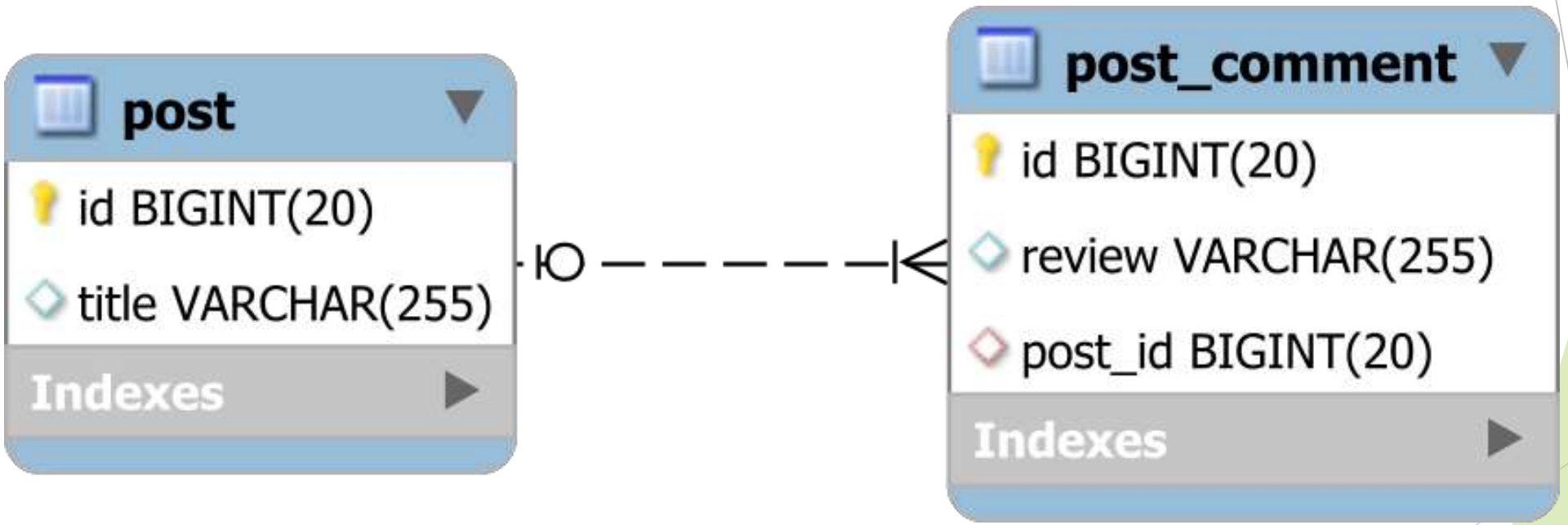
```
@Entity
public class Manuscript {

 @Id
 private Long id;

 @OneToOne
 @MapsId
 @JoinColumn(name = "id")
 private Book book;
  ...
}
```

Test Poject in lesson folder

@oneToMany

- **Hibernate one to many mapping** is made between two entities where first entity can have relation with multiple second entity instances but second can be associated with only one instance of first entity. Its **1 to N** relationship

1 ---------- N



@OneToMany - Example

```
@Entity
@Table(name = "post")

public class Post {

    // id, title
   @OneToMany( fetch =  FetchType.LAZY )
   @JoinColumn( name = "post_id“ )

    private List<PostComment> comments = new ArrayList<>();
}
```

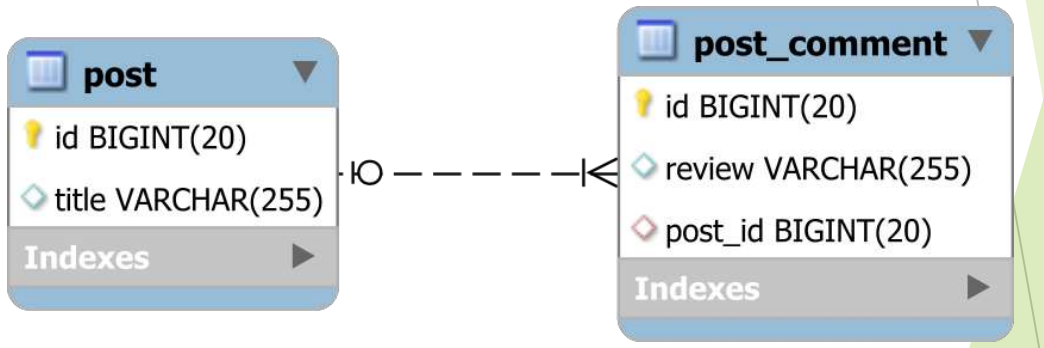
```
@Entity

@Table(name = "post_comment")

public class PostComment {
    // id, content

}
```

- The best way to map a @OneToMany association is to rely on the @ManyToOne side to propagate all entity state changes.

Bidirectional @OneToMany or @ManyToOne

```
@Entity
@Table(name = "post")
public class Post {

    @OneToMany( mappedBy = "post", fetch = FetchType.LAZY)

    private List<PostComment> comments;


}
```

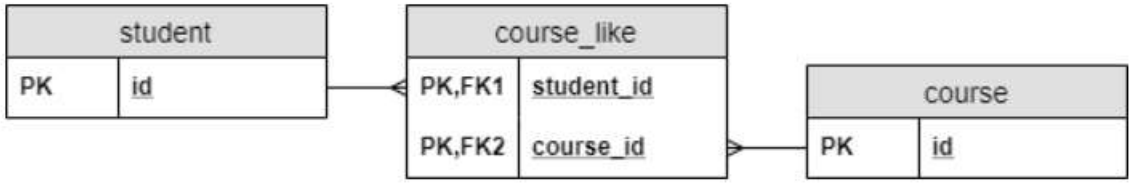
```
@Entity
@Table(name = "post_comment")
public class PostComment {

    // id, content

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "post_id")
    private Post post;
}
```



@ManyToMany

- **A relationship is a connection between two types of entities. In the case of a many-to-many relationship, both sides can relate to multiple instances of the other side.**

- A student can like **many** courses, and **many** students can like the same course.



- Such a table is called a **join table.** In a join table, the combination of the foreign keys will be its composite primary key.

@ManyToMany - Example

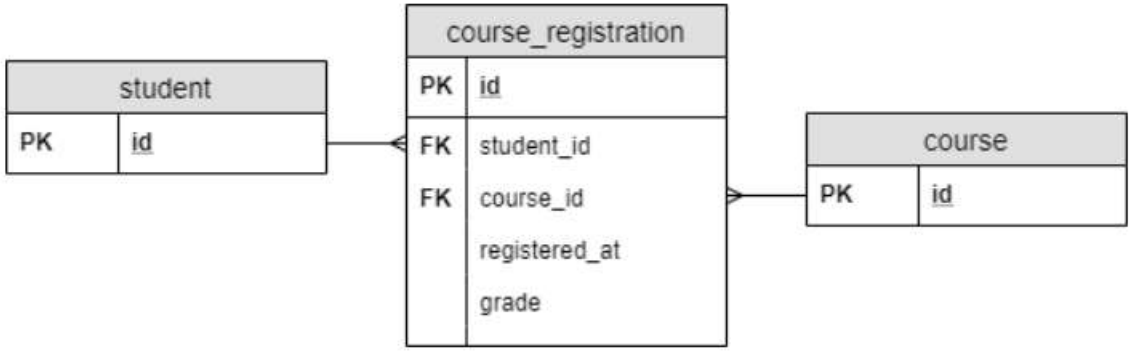
```
@Entity
@Table(name = "student")
public class Student {

    @ManyToMany
    @JoinTable(name = "course_like",

            joinColumns = @JoinColumn(name = "student_id"),

            inverseJoinColumns = @JoinColumn(name = "course_id"))

    private Set<Course> likedCourses;

}

@Entity
@Table(name = "course")
public class Course {

  @ManyToMany(mappedBy = "likedCourses")
  private Set<Student> likes;


}
```

@ManyToMany - Example

```
// ….
Class Student {
@ManyToMany
 @JoinTable(name = "student_book",
       joinColumns = {@JoinColumn(name = "student_id", referencedColumnName = "id")},
       inverseJoinColumns = {@JoinColumn(name = "book_id", referencedColumnName = "id")}

    )
private Set<Book> books;
}

//….
public class Book {
    @ManyToMany(mappedBy = "books")
    private Set<Student> students;

}
```

Many-to-Many With a New Entity

- The only problem is that in JoinTable we cannot add a property to a relationship.
- Therefore, **we had no way to add a property to the relationship itself.**



Many-to-Many With a New Entity

```
public class Student {

    @OneToMany(mappedBy = "student“, fetch = FetchType.LAZY)
    private List<StudentCourse> courseList;

}

public class Course {

    @OneToMany(mappedBy = "course“, fetch = FetchType.LAZY)
    private List<StudentCourse> studentList;

}

public class StudentCourse {
    // id, grade, createdAt …

    @ManyToOne
    @JoinColumn(name = "course_ic")
    private Course course;


    @ManyToOne
    @JoinColumn(name = "student_id")
    private Student student;

}
```

Eager and Lazy Loading

Eager and Lazy Loading

- The fetching strategy is controlled via the fetch attribute of the @OneToMany, @OneToOne, @ManyToOne, or @ManyToMany.
- It is required to define Fetch Type when you use any of these associations
- Fetch Type decides on whether or not to load all the data belongs to associations as soon as you fetch data from parent table
- Fetch type supports two types of loading: Lazy and Eager
- @OneToMany and @ManyToMany use the FetchType.LAZY strategy by default.
- @OneToOne and @ManyToOne use the FetchType.EAGER strategy by default.

Eager and Lazy Loading

- ***FetchType.LAZY***: It fetches the child entities lazily, that is, at the time of fetching parent entity it just fetches proxy (created by cglib or any other utility) of the child entities and when you access any property of child entity then it is actually fetched by hibernate.
- ***FetchType.EAGER***: it fetches the child entities along with parent.
- Lazy initialization improves performance by avoiding unnecessary computation and reduce memory requirements.
- Eager initialization takes more memory consumption and processing speed is slow.
- Having said that, depends on the situation either one of these initialization can be used.

Cascade Types

Cascade Types

- Entity relationships often depend on the existence of another entity — for example, the *Person–Address* relationship.
- Without the *Person*, the *Address* entity doesn't have any meaning of its own. When we delete the *Person* entity, our *Address* entity should also get deleted.
- **When we perform some action on the target entity, the same action will be applied to the associated entity.**
- Cascading only makes sense only for *Parent* – *Child* associations (the *Parent* entity state transition being cascaded to its Child entities). Cascading from *Child* to *Parent* is not very useful and usually, it’s a mapping code smell.
- Odatta bitta Entity ning bor bo’lishi ikkinchi Entity ning mavjut bo’lishiga bog’liq, masalan Person-Address bo’g’lanishlar.
- Person entity siz Address entity ning borligi manoga ega emas. Person dan entity Delete qilinganda Address ham delete bo’lishi kerak.
- Hozirgi Entity da qandaydir hodisa bajarilsa , huddi shunday hodisa bo’g’langan entity da ham bajariladi.

- **JPA Cascade Type**
  - *ALL*
  - *PERSIST*
  - *MERGE*
  - *REMOVE*
  - *REFRESH*
  - *DETACH*
- **Hibernate Cascade Type**
  - *REPLICATE*
  - *SAVE_UPDATE*
  - *LOCK*

CascadeType.PERSIST

- Persistent – doimiy, barqaror…
- The persist operation makes a transient instance persistent.
- **CascadeType *PERSIST* propagates the persist operation from a parent to a child entity.**
- When we save the *person* entity, the *address* entity will also get saved.
- We need to persists parent only and child will be persists also.

- Persist operatsiyasi vaqtinchalik o’bektni doimiy qiladi.
- CascadeType PERSIST da hodisa Parent dan Child ga qarab sodir bo’ladi.
- Biz Person ni save qilsak Address ham save bo’ladi.
- Parentga persistni qo’llasak child ga ham qo’llaniladi.

CascadeType.PERSIST - Example

```
public class Student {
 @OneToOne(fetch = FetchType.LAZY, cascade = CascadeType.PERSIST)
 private Address address;
}

public class Address {
 @OneToOne(fetch = FetchType.EAGER)
 @JoinColumn(name = "student_id")
 private Student student;
}
```

CascadeType.PERSIST - Example

```
Student student = new Student();
student.setFirstName("Ali");;

Address address = new Address();
address.setRegion("Toshkent");
address.setStreet("Katta Ko'cha");

address.setStudent(student);
student.setAddress(address);

session.persist(student);
```

CascadeType.MERGE

- ***CascadeType.MERGE* propagates the merge operation from a parent to a child entity.**
- We only have to merge the *Parent* entity and the associated *child* is merged as well.
- If we change Parent and Child entities and want update them in this case merging parent is enough. As result both Parent and child entities will be updated.

- CascadeType.MERGE da hodisa Parent dan Child ga qarab sodir bo'ladi.
- Parentni merger qilsak, Parent va Child update bo'ladi.
- Biz faqatgina Parentni Merge qilsak uning Child ni ham merge boladi.
- Agar biz Parent va Child entity larni o'zgartirsak va update qilmoqchi bo'lsak , parent ni update qilishni o'zi kifoya. Natijada ikkalasi ham update bo'ladi.

CascadeType.MERGE - Example

```
public class Student {

    @OneToOne(fetch = FetchType.LAZY, cascade = CascadeType.MERGE)
    private Address address;

}

public class Address {

    @OneToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "student_id")
    private Student student;

}
```

CascadeType.MERGE - Example

```
Address savedAddress = session.find(Address.class, address.getId());

savedAddress.setStreet("Kichik ko'cha");


Student savedStudent = savedAddress.getStudent
savedStudent.setFirstName("Vali");


session.merge(savedStudent);
```

CascadeType.REMOVE

- ***CascadeType.REMOVE* propagates the remove operation from parent to child entity**
- If Parent removed Child also removed from DB.
- Remove operation removes the row corresponding to the entity from the database and also from the persistent context.
- **Similar to JPA's *CascadeType.REMOVE*, we have *CascadeType.DELETE*, which is specific to Hibernate.** There is no difference between the two.

- CascadeType.REMOVE da hodisa Parent dan Child ga qarab sodir bo’ladi.
- Agar Parent remove bo’lsa Child ham remove bo’ladi.
- Remove operatsiyasi ob'ektga mos keladigan qatorni DataBase dan o’chiradi, shuningdek, doimiy kontekstdan olib tashlaydi.
- JPA CascadeType.REMOVE dek Hibernate da CascadeType.DELETE bor. Ikkalasi bir xil ish qiladi.

CascadeType.REMOVE - Example

```
public class Student {

    @OneToOne(fetch = FetchType.LAZY, cascade = CascadeType.REMOVE)
    private Address address;

}

public class Address {

    @OneToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "student_id")
    private Student student;

}
```

CascadeType.REMOVE - Example

```
Address savedAddress = session.find(Address.class, address.getId());
Student savedStudent = savedAddress.getStudent();
session.remove(savedStudent);
```

CascadeType.REFRESH

- Refresh operations **reread the value of a given instance from the database.** In some cases, we may change an instance after persisting in the database, but later we need to undo those changes.
- In that kind of scenario, this may be useful. **When we use this operation with Cascade Type *REFRESH*, the child entity also gets reloaded from the database whenever the parent entity is refreshed.**

- Refresh operatsiyasi Entity ni DataBase dan qayta o’qiydi. Ba'zi hollarda biz DataBase dan Entity ni olgandan keyin uni o'zgartirishimiz mumkin, ammo keyinroq bu o'zgarishlarni bekor qilishimiz kerak.
- Bunday stsenariyda bu foydali bo'lishi mumkin. Ushbu operatsiyani Cascade Type REFRESH bilan ishlatganimizda, asosiy ob'ekt yangilanganda, pastki ob'ekt ham ma'lumotlar bazasidan qayta yuklanadi.
- session.refresh(object) - updates object

  session.refresh(object) – updates Parent and Subject objects

CascadeType.REFRESH Example

```
 Person person = buildPerson("devender");
Address address = buildAddress(person);

person.setAddresses(Arrays.asList(address));

session.persist(person);

session.flush();

person.setName("Devender Kumar");
address.setHouseNumber(24);

session.refresh(person);
```

CascadeType.ALL

- *CascadeType.ALL* **propagates all operations from a parent to a child entity.**
- CascadeType.ALL barcha operatsiyalarni Parent dan Child ga qarab sodir qiladi.
- The value cascade=ALL is equivalent to cascade={PERSIST, MERGE, REMOVE, REFRESH, DETACH}

CascadeType.REPLICATE

- ***REPLICATE*** – РЕПЛИКАЦИЯ, копировать, повторять
- **The replicate operation is used when we have more than one data source and we want the data in sync. With CascadeType.REPLICATE, a sync operation also propagates to child entities whenever performed on the parent entity.**

CascadeType.SAVE_UPDATE

- *CascadeType.SAVE_UPDATE* propagates the same operation to the associated child entity. It's useful when we use **Hibernate-specific operations like *save*, *update* and *saveOrUpdate*.**

Links

- https://www.baeldung.com/jpa-cascade-types