Agenda.

→ Representing Cardinalities
  → Mapped By
  → Cascade.

→ Schema Versioning & Migration.

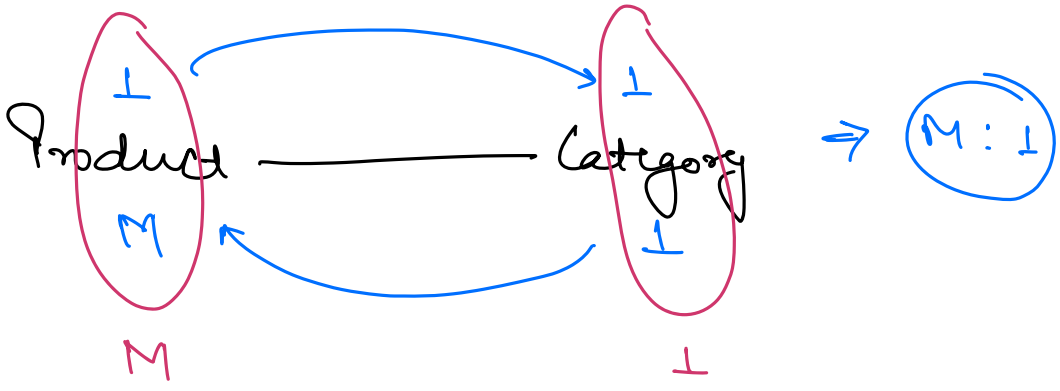
Representing Cardinalities

1 : 1 ⇒ @OneToOne

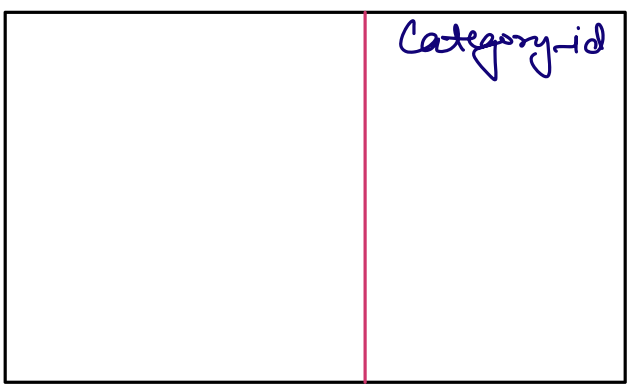
1 : M ⇒ @OneToMany

M : 1 ⇒ @ManyToOne

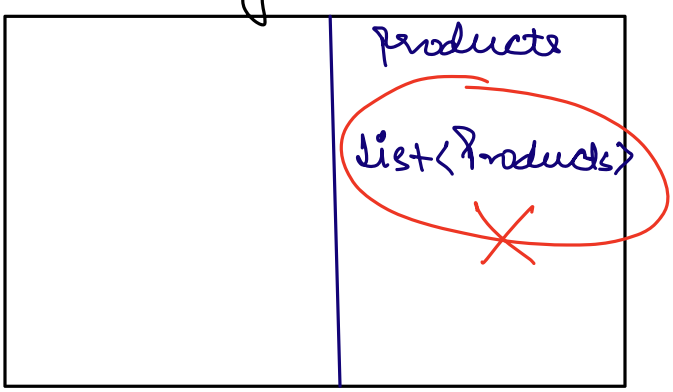
M : M ⇒ @ManyToMany



Products

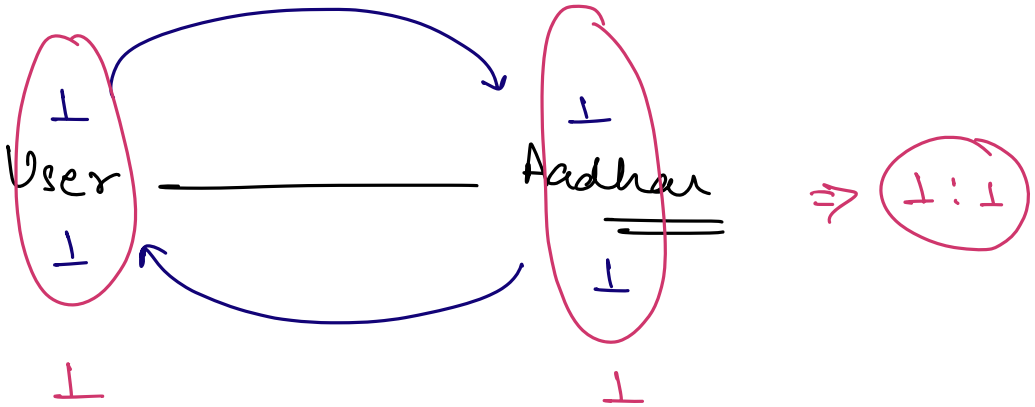


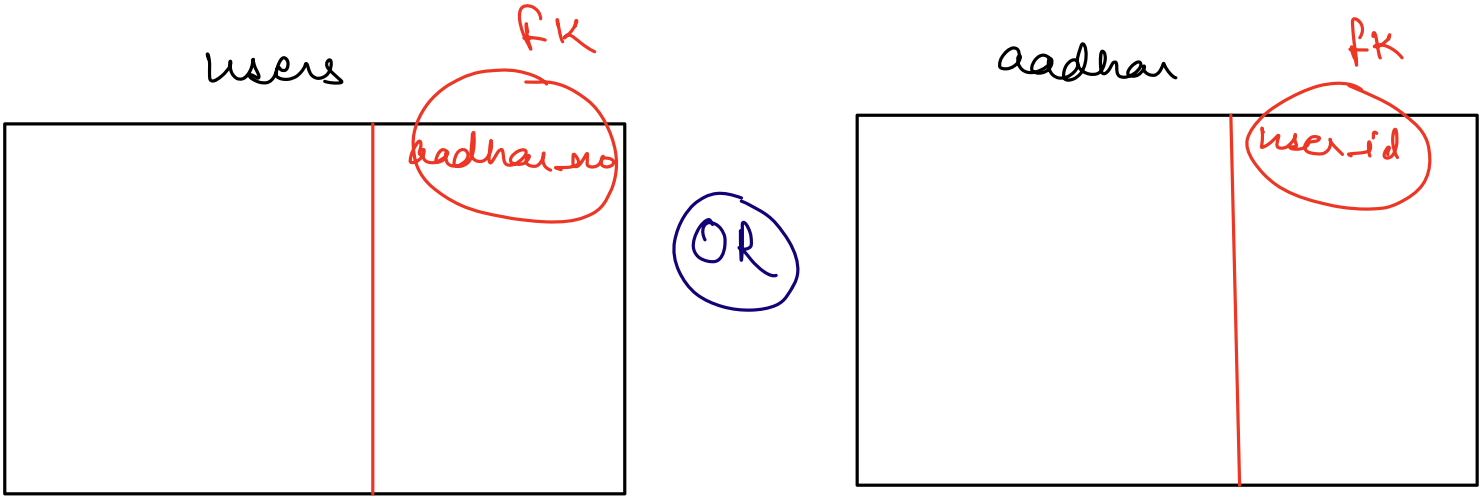
Categories.



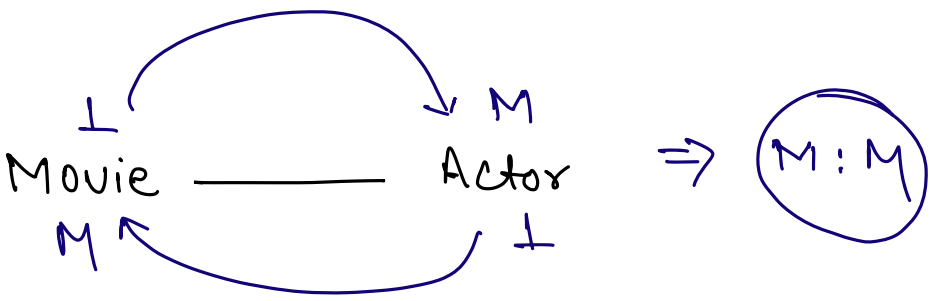
1:M | M:1 ⇒ Id of (1) side on (M) side.

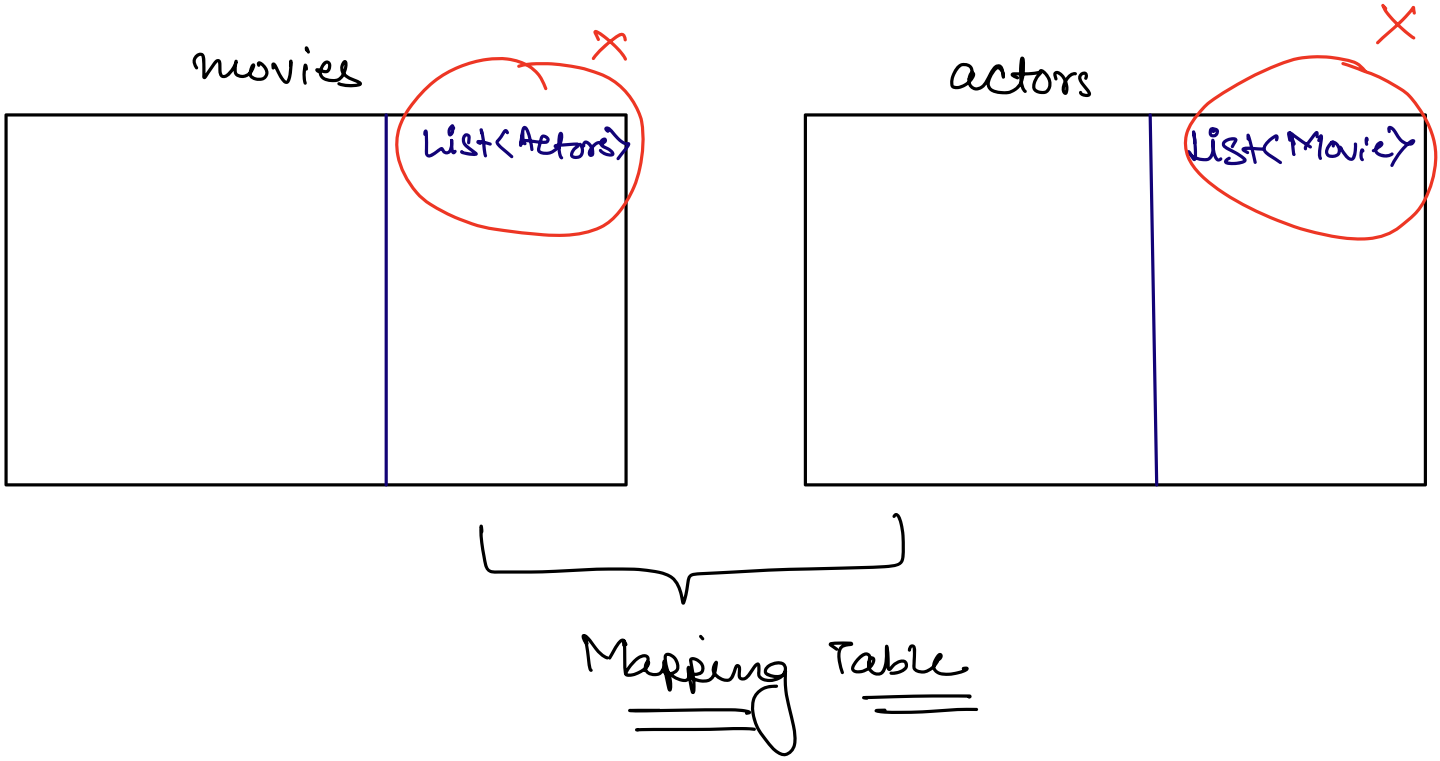
(1:1)





M:M.





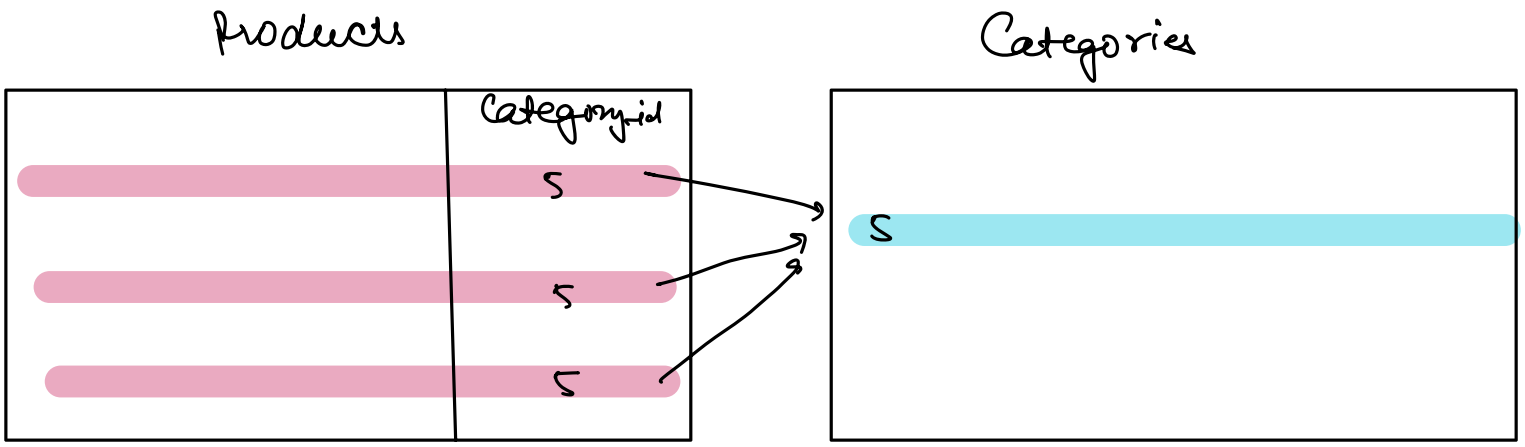
⇒

```
Product {
    ...
    @ManyToOne
    Category
}
```

```
Category {
    ...
    @OneToMany(mappedBy
    List<Products>
}
```

⇒ mappedBy

Cascade.



If we try to delete category, what will happen to Product.

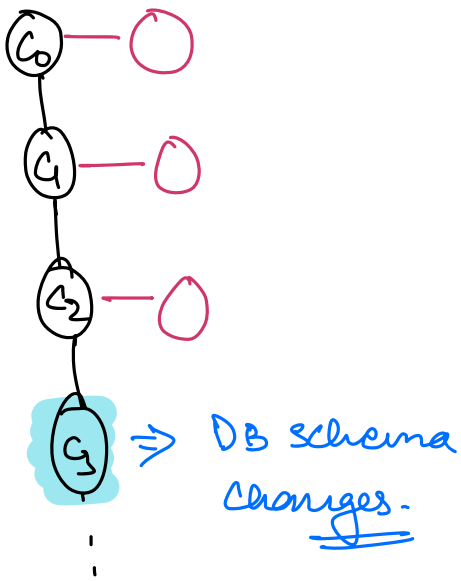
1. Give Error, Don't allow.
2. Update product with NULL categoryid.
3. Delete the products as well.

Schema Migration & Versioning.

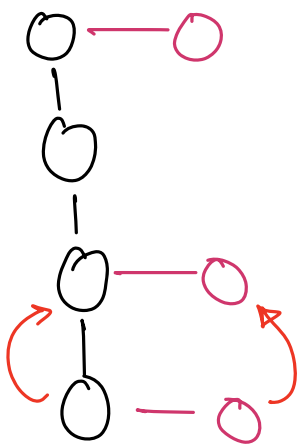
⇒ All the tables were created by ORM, we didn't have any visibility over that.

(1) We might want to have more control over how the tables are being created internally.

(2)

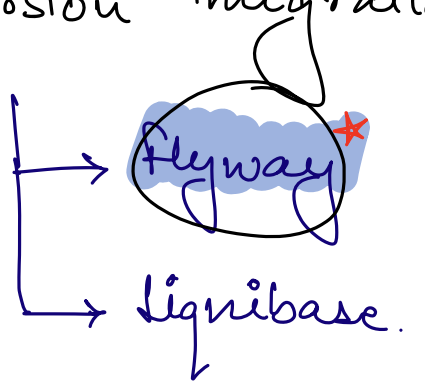


Along with maintaining the code history, we should also maintain the DB history.



⇒ It is important to maintain DB changes history.

⇒ DB version migration libraries.



⇒ In our project, a db.migration folder will be created.

```
db.migration/
        V1__init.sql
        V2__adding_qty_col.sql
        V3__***
```

⇒ Whenever we do any code change that also requires DB changes

→ Create a new DB version sql file with the queries required to do this change.

V1__init.sql ⇒ Create Queries.

V2__*.sql

V3__*.sql

incremental

Every version file will store the queries for delta from the previous version.

⇒ Migration tools like flyway maintains a schema history table to maintain which all the versions have been applied to the DB.

flyway_schema_history

version	timestamp	—	—	—
V1	15th 12:00:0			
V2	—			
V3	—			
V4	—			

⇒ Fetch Mode.

⇒ N+1 problem.

```
Class Category {
    id
    title
    desc
    List<Product> products
}
```

LAZY.

```
List<Category> categories =
                                    ⇒ (1)
    CategoryRepository.findByTitleContains(—);

for ( Category c : categories ) {          // M categories
    for (Product p : c.getProducts()) {
        Sout (p.getTitle())                // (M) times.
    }
}
```

Assumption : M Categories

Each category has N products.

1. (1) + (M) → for each category fetch list of products.
   ↓ fetch all Category

2. 1 + N×M ⇒ N×M + 1 ⇒ Worst.

3. Select * from Categories where title like '%...%' → [.....]

   Select * from products where category_id IN [.....]

⇒ If your ORM isn't optimising the N/w calls, write your own native queries.

@Query(—>)