How table creation mappings happen?--

```
@Entity
class Student
    @Id
    long id

    String name

    string psp
```

```
@Entity
class laptop
    @Id
    long id

    string name

    string brand
```

student.

| id | name | psp |
|---|---|---|
| | | |

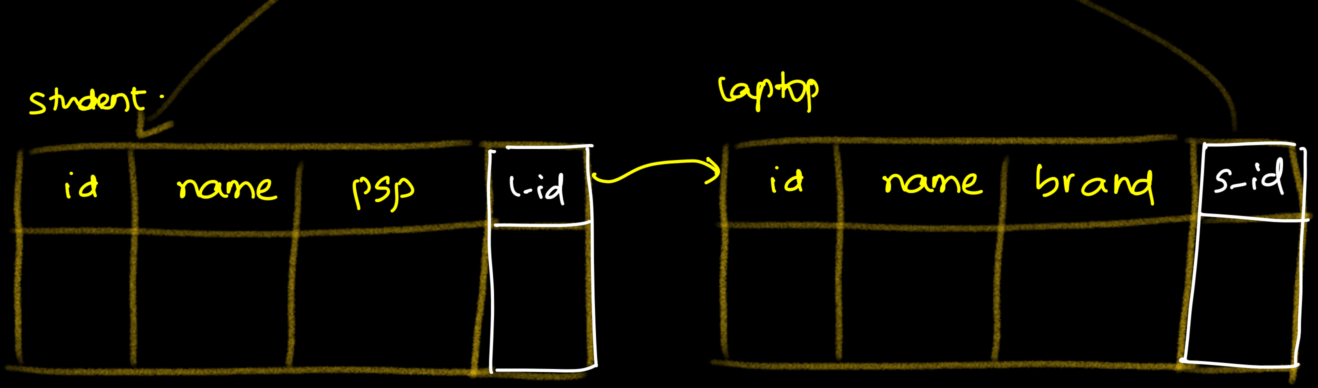
laptop

| id | name | brand |
|---|---|---|
| | | |

Each student can have a laptop. (1:1 relation).

```
@Entity
class Student
    @Id
    long id

    String name

    string psp

    @OneToOne
    Laptop laptop
```

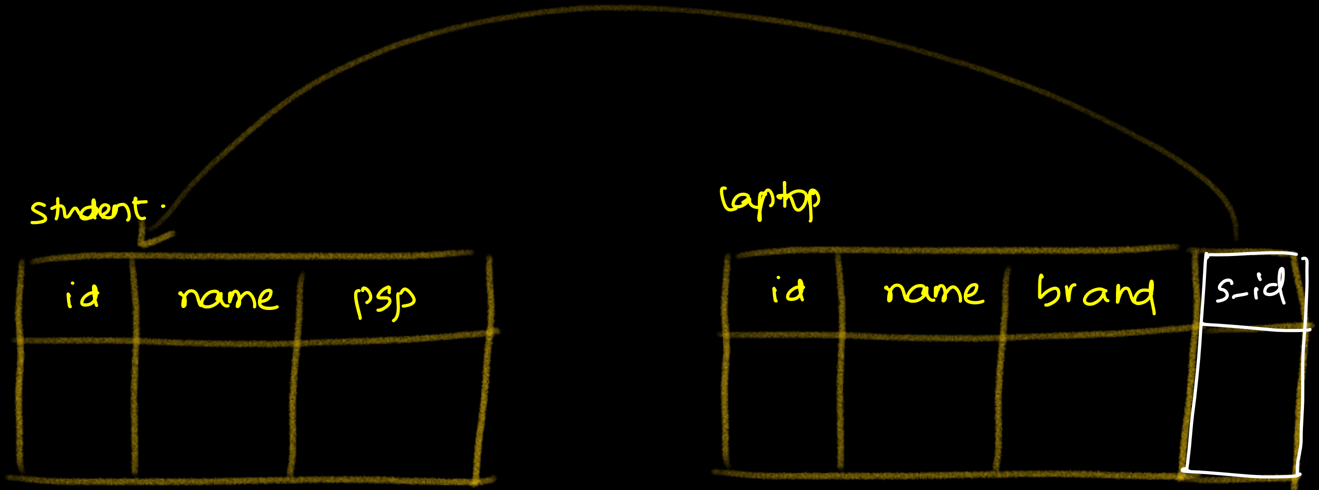
```
@Entity
class laptop
    @Id
    long id

    string name

    string brand

    @OneToOne
    Student student
```



Explicitly mention that its a duplicate mapping.

```
@Entity
class Student
    @Id
    long id

    String name

    string psp

    @OneToOne (mappedBy = Student)
    Laptop laptop
```

```
@Entity
class laptop
    @Id
    long id

    string name

    string brand

    @OneToOne
    Student student
```



Let's say,

Each student can have multiple laptops. ( 1: M)

```
@Entity
class Student
    @Id
    long id

    String name

    string psp

    @OneToMany
    List<Laptop> laptops
```

```
@Entity
class laptop
    @Id
    long id

    string name

    string brand

    @ManyToOne
    Student student
```

What gets created:

laptop

| id | name | brand | s-id |
|---|---|---|---|
| | | | |

student

| id | name | psp |
|---|---|---|
| | | |

Student-laptop

| s-id | l-id |
|---|---|
| 1 | 2 |
| 1 | 3 |
| 1 | 5 |

Soln:

```
@Entity
class Student
    @Id
    long id

    String name

    string psp

    @OneToMany(mappedBy = "student")
    List<Laptop> laptops
```

```
@Entity
class laptop
    @Id
    long id

    string name

    string brand

    @ManyToOne
    Student student
```

This will make sure the mapping table

is not created.

Let's say---

Each student can have multiple laptops, but some laptops can be shared by many students.

$$\left.\begin{array}{l} 1:M \\ M:1 \end{array}\right\} \; M:M.$$

```
@Entity
class Student
    @Id
    long id

    String name

    String psp

    @OneToMany
    List<Laptop> laptops
```

```
@Entity
class Laptop
    @Id
    long id

    String name

    String brand

    @OneToMany
    List<Student> students
```

laptop

| id | name | brand |
|---|---|---|
| | | |

student

| id | name | psp |
|---|---|---|
| | | |

Student-laptop

| s-id | l-id |
|---|---|
| 1 | 2 |
| 1 | 3 |
| 1 | 5 |

laptop-Student

| s-id | l-id |
|---|---|
| 1 | 2 |
| 1 | 3 |
| 1 | 5 |

Sol$^n$:

```
@Entity
class Student
    @Id
    long id

    String name

    String psp

    @OneToMany(mappedBy = students)
    List<Laptop> laptops
```

```
@Entity
class Laptop
    @Id
    long id

    String name

    String brand

    @OneToMany
    List<Student> students
```

laptop

| id | name | brand |
|---|---|---|
| | | |

student

| id | name | psp |
|---|---|---|
| | | |

laptop-student

| s-id | l-id |
|---|---|
| 1 | 2 |
| 1 | 3 |
| 1 | 5 |

Types of queries (Inside repository).

1. Declared queries.

→ 2. HQL

3. Native queries.

HQL : Hibernate query language.

Very similar to SQL.

(Combination of SQL and OOps)

SQL → select * from product. → tableName

HQL → select * from Product → modelName.

→ Column name.

→ field name.

```
@Query(" select s.psp, s.brand from student s")
List<CustomClass> getPspAndBrand();
```

```
~~class CustomClass~~
    ~~String psp;~~
    ~~String brand;~~
```

```
interface custom
    String getPsp();
    String getBrand();
```

Advantages:

1. Easy to write complex queries
2. HQL queries are DB independent
3. More control over the query.

## Disadvantages

1. Queries may not be optimal for all DB types.

## Native Queries.

Write the exact query that you want to execute in your DB language.

```
@Query("[redacted]", nativeQuery = true)
List<CustomClass>   getPspAndBrand();
```