```
@Entity
@Table(name = "employee")
public class Employee {

@ManyToOne
    @JoinColumn(name = "department_id", referencedColumnName = "id")
    private Department department;

}
```

- The above code written under "Employee" class.
- **Below are the queries for Employee and Department table**

```
CREATE TABLE department (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL
);

CREATE TABLE employee (
    id SERIAL PRIMARY KEY,
    firstname VARCHAR(255) NOT NULL,
    lastname VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL,
    department_id BIGINT UNSIGNED DEFAULT NULL,
    CONSTRAINT fk_department FOREIGN KEY (department_id) REFERENCES department(id)
);
```

@JoinColumn is normally used we want to map a foreign key (owing the entity) to primary key (another entity, not the owing)

- In the above example, the "department_id" is a foreign key from employee table which is referencing to primary key of department table.
- where as in department entity we will not have same mapping like we have in employee entity, below is the example of one.

```
@OneToMany(mappedBy = "department", cascade = CascadeType.ALL, orphanRemoval = true)
@JsonIgnore
private List<Employee> employeeList = new ArrayList<>();
```

- Important to note that, while defining mapping between foreign key to primary key in employee table, we had defined this annotation on top Department table instance, the reason behind is that we are referencing our foreign key to primary key (which resides in Department table)

- Same goes for "mappedBy", we have defined the annotations on top of a List object of Employee entity (table), which connects the linking of department table with employee table.

```
@Entity
@Table(name = "employee")
public class Employee {

@ManyToMany(fetch = FetchType.LAZY)
   @JoinTable(
         name = "employee_skill",
         joinColumns = {@JoinColumn(name = "employee_id", referencedColumnName = "id") },
         inverseJoinColumns = { @JoinColumn(name = "skill_id", referencedColumnName = "id") }
   )
private List<Skill> skill = new ArrayList<>();
}
```

- **Below are the queries on which the above column joining are happening.**

```
CREATE TABLE employee (
   id SERIAL PRIMARY KEY,
   firstname VARCHAR(255) NOT NULL,
   lastname VARCHAR(255) NOT NULL,
   email VARCHAR(255) NOT NULL,
   department_id BIGINT UNSIGNED DEFAULT NULL,
   CONSTRAINT fk_department FOREIGN KEY (department_id) REFERENCES department(id)
);

CREATE TABLE employee_skill (
   employee_id BIGINT UNSIGNED,
   skill_id BIGINT UNSIGNED,
   PRIMARY KEY (employee_id, skill_id),
   CONSTRAINT fk_employee FOREIGN KEY (employee_id) REFERENCES employee(id),
   CONSTRAINT fk_skill FOREIGN KEY (skill_id) REFERENCES skill(id)
);

CREATE TABLE skill (
   id SERIAL PRIMARY KEY,
   name VARCHAR(255) NOT NULL
);
```

- Inside employee table.query we can notice from the above query, one foreign key from employee_skill table is referencing to primary key of employee table (owing entity)
- There is another foreign key inside employe table/entity which is referencing to a primary key of skill table (another - not owning entity)

- Now let's discuss below annotations.

   -
     ```
     name = "employee_skill",
           joinColumns = {@JoinColumn(name = "employee_id", referencedColumnName = "id") },
           inverseJoinColumns = { @JoinColumn(name = "skill_id", referencedColumnName = "id") }
     private List<Skill> skill = new ArrayList<>();
     ```

   - JoinColumn consist of a foreign key "employee_id" from "employee_skill" table referencing to primary key "id" from "employee" table.
   - So the concept remains same as per previous discussion (page1).
   - Except there's one change, in the page1, we discussed that we normally put @JoinColumn annotations on top of the instance on which the primary key is referencing but in this case we have instance of Skill entity, the reason behind is that employee table is owning entity, where in page1 scenario we were referencing to primary key of not owning entity, which was department.

   - inverseJoinColumns is used when we are referencing a foreign key (another - not owning entity) to a primary key (another - not owning entity). Other than this everything remains same JoinColumn.
   - we have used a instance of Skill entity to annotate/define inverseJoinColumns.
   - Inside Skill entity we will have mappedBy annotation, on top of Employee's instance (instance of employee entity)

     ```
     @ManyToMany(mappedBy = "skill")
     @JsonIgnore
     private List<Employee> employeeList;
     ```