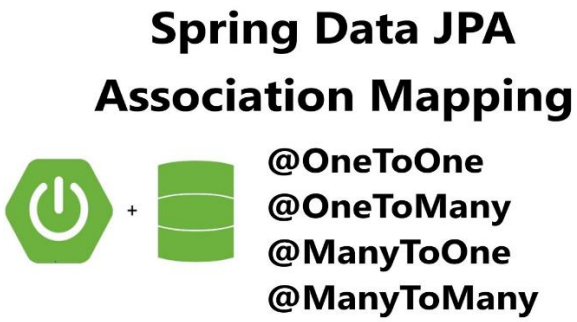
Materi 11 : Relasi Antar Tabel (Association Mapping)

DAFTAR ISI

1.1 Association Mapping

Pada pertemuan sebelumnya kita sudah melakukan koneksi ke database dan membuat sebuah entitas. Anggap lah kalian sudah memiliki lebih dari 1 entitas, maka tugas selanjutnya adalah bagaimana cara menghubungkan beberapa entitas tersebut?

Jawabannya adalah dengan menggunakan konsep Association Mapping (Asosiasi Mapping). Asosiasi Mapping ini adalah proses menghubungkan 2 entitas di JPA dengan tujuan untuk menyimpan dan mengambil data dari database secara efisien.



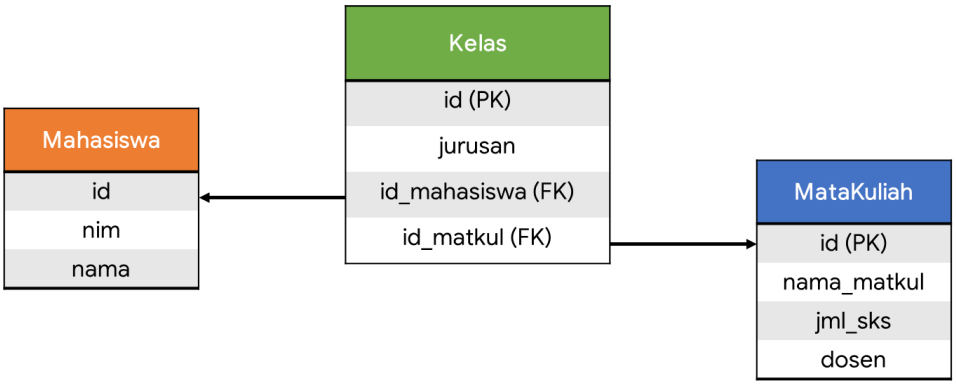
Terdapat 4 macam anotasi yang bisa dilakukan di JPA, yaitu:

- `@OneToOne`, atau satu ke satu
- `@OneToMany`, atau satu ke banyak
- `@ManyToOne`, atau banyak ke satu
- `@ManyToMany`, atau banyak ke banyak

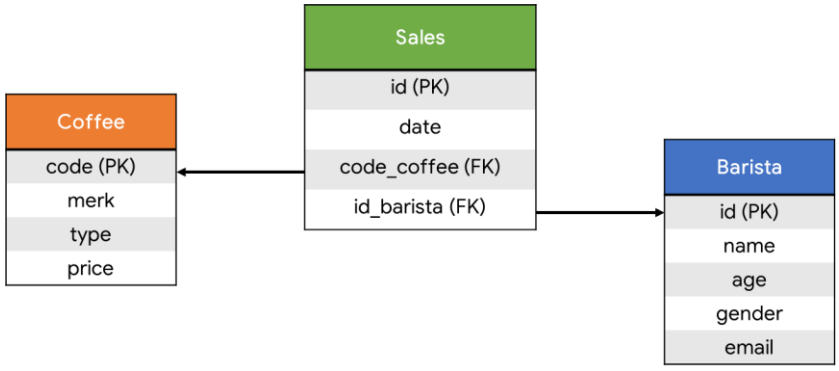
Di pertemuan ini, kita hanya akan membahas anotasi `@ManyToOne` saja, karena fokus kita sekarang adalah menghubungkan beberapa entitas dengan foreign_key. Untuk jenis-jenis relasinya akan dibahas pertemuan selanjutnya atau kalian bisa belajar mandiri.

1.2 Implementasi Kode

Misalnya kita memiliki 2 entitas bernama `Mahasiswa` dan `MataKuliah`. Kemudian kita ingin membuat entitas baru yaitu `Kelas` dengan skema database seperti gambar berikut.



Atau misalkan kita memiliki 2 entitas bernama **`Coffee`** dan **`Barista`**. Kemudian kita ingin membuat entitas baru yaitu **`Sales`** dengan skema database seperti gambar berikut.



Seperti biasa, hal yang harus kita lakukan adalah membuat entitasnya terlebih dahulu, kemudian buat repository, lalu buat end-pointnya untuk ditampilkan di view.

1.2.1 Membuat Entity Sales

Cara membuat Sales cukup sederhana, hanya ditambah beberapa kode berikut ini.

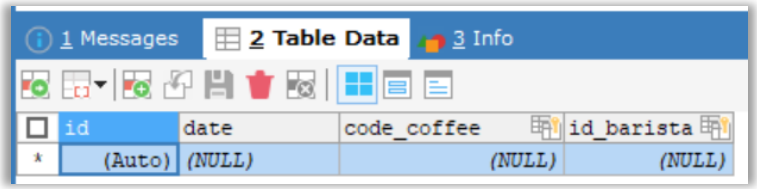
```
@Data
@Entity
public class Sales {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private String date;

    @ManyToOne
    @JoinColumn(name = "code_coffee", referencedColumnName = "code")
    private Coffee codeCoffee;

    @ManyToOne
    @JoinColumn(name = "id_barista", referencedColumnName = "id")
    private Barista idBarista;
}
```

Kode di atas tidak beda jauh dengan kode yang biasa kita buat pada pelatihan sebelumnya, hanya ada beberapa baris kode baru dengan keterangan sebagai berikut:

- **`@ManyToOne`** artinya 1 pesanan bisa memuat banyak kopi, namun 1 kopi hanya bisa dipesan dengan satu pesanan/struk. Begitu juga dengan barista, 1 pesanan bisa dibuat oleh banyak barista, namun 1 barista hanya bisa membuat 1 pesanan.

- `@JoinColumn` artinya kita membuat sebuah foreign_key dengan nama alias/inisial `code_coffee`, kemudian referensi kolomnya merujuk ke kolom `code` yang ada di tabel `coffee`. Begitu juga dengan objek barista.
- `private Coffee codeCoffee` dan `private Barista idBarista` artinya kita membuat sebuah objek dengan tipe data `Coffee` (injeksi dependensi) dengan nama `codeCoffee`. Begitu juga dengan barista. Objek ini bertujuan untuk mengolah dan menyimpan data yang diambil atau disimpan di database.

Jika kode di atas di run, maka akan membuat sebuah tabel di database seperti gambar di bawah ini:



1.2.2 Akses Database dengan SalesRepository

Kemudian kita akan membuat sebuah class interface untuk mengakses database menggunakan `JpaRepository`

```
public interface SalesRepository extends JpaRepository<Sales, Integer>{
}
```

1.2.3 Membuat End-Point dengan SalesController

Setelah kita membuat entitas dan repositori, sekarang kita harus membuat end-point agar data sales bisa kita tampilkan.

```
@Controller
public class SalesContraller {
    @Autowired
    private SalesRepository salesRepository;

    @GetMapping("sales")
    public String showSales(Model model){
        model.addAttribute("sales", salesRepository.findAll());
        return "show-sales";
    }
}
```

1.2.4 Menampilkan Data dengan Thymelaf

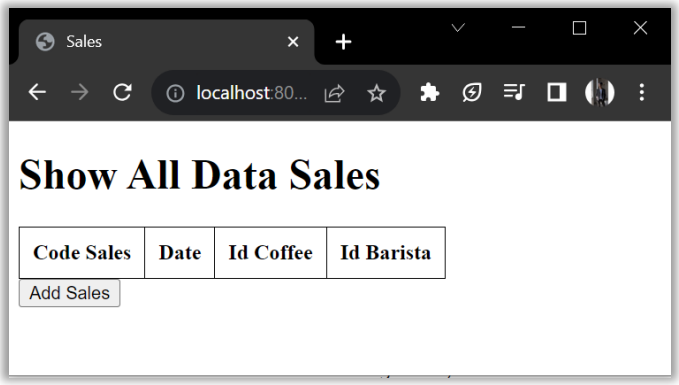
Dari kode sebelumnya, method **showSales** mengembalikan nilai string **"show-sales"**. Untuk itu sekarang kita buat sebuah file HTML dengan nama tersebut. Untuk singkatnya buatlah kode berikut:

```
<body>
    <h1>Show All Data Sales</h1>
    <table>
        <tr>
            <th>Code Sales</th>
            <th>Date</th>
            <th>Id Coffee</th>
            <th>Id Barista</th>
        </tr>
        <tr th:each="s : ${sales}">
            <td th:text="${s.id}"></td>
            <td th:text="${s.date}"></td>
            <td th:text="${s.codeCoffee.code}"></td>
            <td th:text="${s.idBarista.id}"></td>
        </tr>
    </table>
    <a th:href="@{add-sales}">
        <button>Add Sales</button>
    </a>
</body>
```

Dari kode di atas, terdapat 2 buah kode yang baru kita lihat dengan keterangan sebagai berikut:

- **<td th:text="${s.codeCoffee.code}" ></td>** memiliki arti bahwa kita mengakses kolom **code_coffee** yang ada di database melalui objek **codeCoffee.code**
- Begitu juga dengan **<td th:text="${s.idBarista.id}" ></td>** yang memiliki arti bahwa kita mengakses kolom **id_barista** yang ada di database melalui objek **idBarista.id**

Jika sudah membuat kode dari beberapa tahapan di atas maka akan menghasilkan output sebagai berikut:



Kenapa tidak ada data yang muncul? Tentu saja karena kita belum mengisi data apapun di tabel tersebut 😅.

1.3 Menambahkan Data

Cara menambahkan datanya pun hampir sama seperti pertemuan kemarin. Untuk lebih lengkapnya perhatikan beberapa langkah berikut ini:

1.3.1 Membuat End-Point Untuk Menerima Inputan

Sebelumnya kita sudah membuat sebuah button di file **`show-sales.html`** yang mana jika kita klik tombol **`"Add Sales"`** maka akan menuju pada end-point **`"add-sales"`**. Sekarang tugas kita adalah membuat sebuah controller untuk menerima inputan dengan kode sebagai berikut

```
@Autowired
private CoffeeRepository coffeeRepository;

@Autowired
private BaristaRepository baristaRepository;

@GetMapping("add-sales")
public String addSales(Model model){
    Sales sales = new Sales();
    model.addAttribute("sales", sales);
    model.addAttribute("kopi", coffeeRepository.findAll());
    model.addAttribute("baris", baristaRepository.findAll());
    return "save-sales";
}
```

Sama seperti kemarin kita menyiapkan sebuah objek **`Sales`** yang digunakan sebagai tampungan dari inputan yang kita kirim.

Lalu kenapa kita memanggil tabel `coffee` dan `barista` dari database? Ini opsional saja, tidak memakai pun tidak apa-apa. Namun pada program ini kita akan menggunakannya di inputan `<select>`.

1.3.2 Membuat Tampilan Inputan di Halaman Web

Dari kode sebelumnya, method `addSales` mengembalikan nilai string berupa `"save-sales"`. Jadi sekarang kita buat sebuah file HTML dengan nama `save-sales.html` kemudian isi dengan kode berikut ini.

```
<body>
 <h1>Add Sales</h1>
 <form action="#" th:action="@{save-sale}" method="post">
    <label>Masukan tanggal</label>
    <input type="date" placeholder="Tanggal" th:field="${sales.date}" />
    <br />
    <label>Masukan code_coffee</label>
    <select th:field="${sales.codeCoffee.code}">
    <option
        th:each="s : ${kopi}"
        th:value="${s.code}"
        th:text="${s.code}"
    ></option>
    </select>
    <br />
    <label>Masukan id_barista</label>
    <select th:field="${sales.idBarista.id}">
    <option
        th:each="s : ${baris}"
        th:value="${s.id}"
        th:text="${s.id}"
    ></option>
    </select>
    <br />
    <input type="submit" value="Kirim" />
 </form>
</body>
```

Seperti yang disampaikan sebelumnya, pengambilan data dari tabel `coffee` dan `barista` digunakan untuk inputan drop down di tag `<select>` dimana data data tersebut akan ditampilkan di tag `<option>`.

Tujuannya adalah agar kita tidak salah menginputkan `code_coffee` ataupun `id_barista`. Hal ini tidak diwajibkan karena di pertemuan selanjutnya kita akan membahas validasi di back-end (menggunakan package service).

Kode di atas akan menghasilkan output berikut:

Jika di dalam drop down code_coffee dan id_barista tidak memunculkan apa-apa, maka coba isi dulu data yang ada di kedua tabel tersebut.

Misalnya kita mengirimkan inputan seperti gambar berikut.

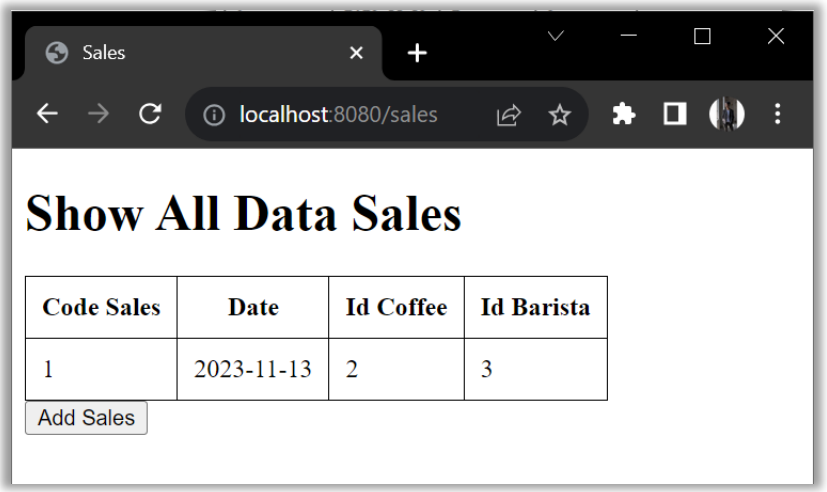
Dari form di atas, ketika kita menginputkan data maka data akan di kirimkan oleh atribut `th:action="@{save-sale}"`. Maka dari itu, sekarang kita akan membuat end-point untuk menyimpan data tersebut.

1.3.3 Menyimpan Data ke Database

Sekarang kita harus membuat end-point POST untuk menambahkan data dari client ke database kita dengan menambahkan kode berikut ini.

```
@PostMapping("save-sale")
public String saveSales(Sales sales){
   salesRepository.save(sales);
   return "redirect:/sales";
}
```

Method tersebut akan mengembalikan nilai string `"redirect:/sales"` sehingga jika data yang kita inputkan tadi di kirim, maka akan menghasilkan output berikut:


Sales
localhost:8080/sales
Show All Data Sales
Code Sales
Date
Id Coffee
Id Barista
1
2023-11-13
2
3
Add Sales