SORU2

Aşağıdaki SQL sorgusu, "Car" tablosundaki "brand name" ve "model year" parametrelerine göre eşleşen ve "Auto Gallery" tablosundaki ilişkilendirilmiş galeri kayıtlarını içeren verilerin sayısını bulur. Ardından, bu sayısı 3'ten fazla olan kayıtları döndürür:

```
SELECT count(*) as counts , c.brand_name AS brand_name, c.model_year AS
model_year

FROM car c

WHERE c.brand_name = 'Honda Civic' AND c.model_year = '2003'

GROUP BY c.brand_name, c.model_year

HAVING COUNT(*) > 3;
```

Yukarıdaki sorguyu kullanırken, "brand_name_değeri" ve "model_year_değeri" parametrelerini kendi gereksinimlerinize göre değiştirmeniz gerekmektedir. Bu parametreler, aradığınız marka adını ve model yılını temsil etmelidir.

SORU 3

@GeneratedValue : Birincil anahtar (Primary Key) değerleri için oluşturma stratejilerinin belirtimini sağlar.
@İD
@GeneratedValue (strateji = GenerationType.TABLE , generator = "student_generator")

```
@Entity
public class Student {

    @Id
    @GeneratedValue (strategy = GenerationType.IDENTITY)
    private long studentId;

    // ...
}
```

@Lob : LOB veya Büyük Object, büyük nesneleri depolamak için değişken uzunluklu bir veri tipini ifade eder.
Veri tipinin iki çeşidi vardır:

- CLOB – *Karakter Büyük Nesnesi,* büyük metin verilerini depolar
- BLOB – *İkili Büyük Nesne,* görüntü, ses veya video gibi ikili verileri depolamak içindir

3. LOB Veri Modeli

Modelimiz *"Kullanıcı"* özellikler olarak id, isim ve fotoğrafa sahiptir. *Kullanıcının* fotoğraf özelliğinde bir resim saklayacağız ve onu bir BLOB'a eşleyeceğiz:

```java
@Entity
@Table(name="user")
public class User {

    @Id
    private String id;

    @Column(name = "name", columnDefinition="VARCHAR(128)")
    private String name;

    @Lob
    @Column(name = "photo", columnDefinition="BLOB")
    private byte[] photo;

    // ...
}
```

@Bean : Bir metodun Spring tarafından yönetilen bir Bean ürettiğini belirtir .
@Configuration : Bean tanımlamaları gibi tanımlamalar için bir Bean sınıfı olduğunu belirtir.

IOC containerında @service , @repository vs. referansı oluşturulur . Bean ise dışarıdan bir şey kullancagım zaman örnekleyip applicationContextin içerisinde referansını oluşturur bu sayede @service , @repository gibi sınıfları kullandığımız gibi bean ile oluşturduğumuz nesnemizide kullanabiliriz . Bu IOC'lerin kullanım mantığı da Dependency Injection .

Spring bize @autowired anatasyonunu önermez . Constructur dependency injection yönetemi ile esnek bağlama yapabilriz . @AllConst..

```java
@Configuration
public class ResourceSecurityConfigurer {

    @Bean
    public FilterRegistrationBean<CorsFilter> corsFilter() {
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        CorsConfiguration config = new CorsConfiguration();
        config.setAllowCredentials(true);
        config.addAllowedOriginPattern("*");
        config.addAllowedHeader("*");
        config.addAllowedMethod("*");
        source.registerCorsConfiguration("/**", config);
        FilterRegistrationBean<CorsFilter> bean = new FilterRegistrationBean<CorsFilter>(new CorsFilter(source));
        bean.setOrder(0);
        return bean;
    }

}
```

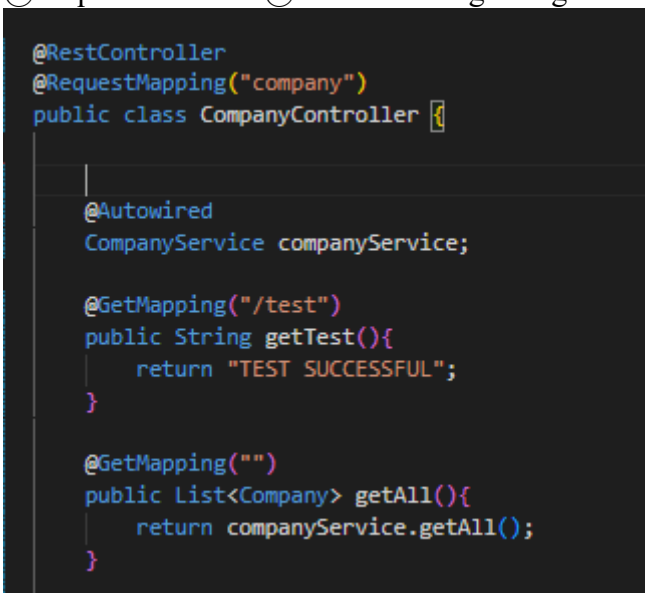
@RestController : Controller sınıfının handle ettiği (işlediği) HTTP Requestlerin path eşleştirmesini yapar .

```java
@RestController
@RequestMapping("books-rest")
public class SimpleBookRestController {

    @GetMapping("/{id}", produces = "application/json")
    public Book getBook(@PathVariable int id) {
        return findBookById(id);
    }

    private Book findBookById(int id) {
        // ...
    }
}
```

@RequestMapping : Bu anotasyon, bir metod veya sınıf için istek yönlendirmesini tanımlar. Örneğin, bir metodun /customer/create adresiyle çağrılmasını sağlamak için @RequestMapping("/customer/create") anotasyonunu kullanabilirsiniz. Bu anotasyon, @RequestParam ve @PathVariable gibi diğer anotasyonlarla birlikte kullanılabilir.



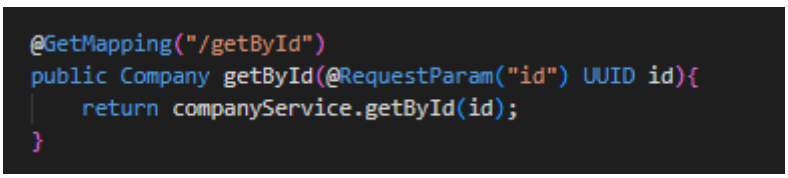
```java
@RestController
@RequestMapping("company")
public class CompanyController {

    @Autowired
    CompanyService companyService;

    @GetMapping("/test")
    public String getTest(){
        return "TEST SUCCESSFUL";
    }

    @GetMapping("")
    public List<Company> getAll(){
        return companyService.getAll();
    }
}
```

@RequestParam: Bu anotasyon, bir metodun çağrılması sırasında gönderilen bir parametreyi almasını sağlar. Örneğin, bir metodun /customer/create adresiyle çağrılması sırasında gönderilen "firstName" parametresini almasını sağlamak için @RequestParam("firstName") String firstName anotasyonunu kullanabilirsiniz.



```java
@GetMapping("/getById")
public Company getById(@RequestParam("id") UUID id){
    return companyService.getById(id);
}
```

@RequestBody : *HttpRequest* **gövdesini bir transfer veya etki alanı(domain) nesnesine eşler ve gelen HttpRequest gövdesinin bir Java nesnesine otomatik olarak seri** *hale* getirilmesini *sağlar*

```java
@PostMapping("")
public String add(@RequestBody Company company){
    companyService.add(company);
    return "SUCCESSFUL ADDED";
}
```

@PathVariable: Bu anotasyon, bir metodun çağrılması sırasında gönderilen bir yol değişkenini almasını sağlar. Örneğin, bir metodun /customer/{id} adresiyle çağrılması sırasında gönderilen "id" değişkenini almasını sağlamak için @PathVariable("id") int id anotasyonunu kullanabilirsiniz.

```java
@PutMapping("/{id}")
public String update(@PathVariable UUID id, @RequestBody Company company){
    companyService.update(id,company);
    return "SUCCESSFUL UPDATED";
}
```

@Autowired : Constructor, Değişken yada setter metodlar için dependency injection işlemi gerçekleştirir .

Spring Contex, container enjeksiyon noktalarını bularak, orada bir nesne oluştrur. New anahtar kelimesi ile bir nesne oluşturulmaz. Kullanılacak nesne Spring tarafından oluşturulur. Bu sayede, **dependency injection** tasarım kalıbı (design pattern) da uygulanmış olur. Özetle **@Autowired** yalnızca injection için kullanılır. @Autowired ifadesini kullanabilmek için, her iki sınıf da **Bean** sınıfı olmalı. Örneğin **@Service** anotasyonu ile tanımlanmış bir servis sınıfı, **@Repository** ile tanımlanmış bir sınıftan **@Autowired** ile bir nesne üreterek, veritabanı işlemlerini gerçekleştirebiliyor. Burada @Service ve @Repository tanımlarının yapıldığı sınıflar için **Spring IoC Container** (application context) içerisinde gerekli **Bean**'ler oluşturulduğu için, her iki sınıf da aslında **Bean sınıfı** olmuş oluyor. Dolayısıyla service sınıfı içerisinde repository sınıfı için **@Autowired** işlemi yapılabiliyor.

Kısaca **@Autowired**, bean nesnesinin istenilen alana, başka bir bean nesnesinin alınıp, yerleştirilmesiyle olur.

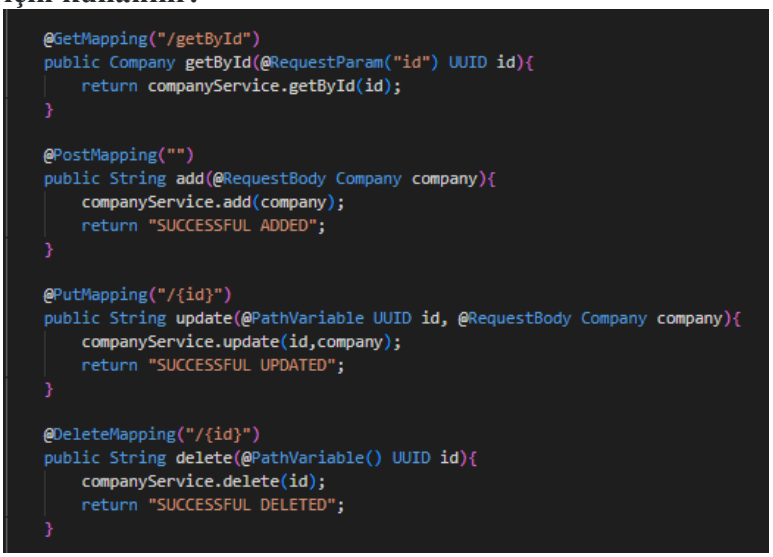
```java
@RestController
@RequestMapping("company")
public class CompanyController {

    @Autowired
    CompanyService companyService;

    @GetMapping("/test")
    public String getTest(){
        return "TEST SUCCESSFUL";
    }
```

@GetMapping : GET **HTTP** isteği, tek veya birden çok kaynak almak için ve HTTP GET isteklerini belirli işleyici yöntemlerine eşlemek için **@GetMapping** notu almak için kullanılır
@PostMapping : POST **HTTP yöntemi,** HTTP POST isteklerini belirli işleyici yöntemlerine eşlemek için bir kaynak ve **@PostMapping** ek açıklaması oluşturmak için kullanılır .

@PutMapping : PUT **HTTP** yöntemi, kaynağı güncellemek için ve HTTP PUT isteklerini belirli işleyici yöntemlerine eşlemek için **@PutMapping notunu güncellemek için kullanılır.**
@DeleteMapping : DELETE **HTTP yöntemi,** kaynağı silmek için ve HTTP DELETE isteklerini belirli işleyici yöntemlerine eşlemek için **@DeleteMapping ek açıklaması silmek için kullanılır.**



```java
@GetMapping("/getById")
public Company getById(@RequestParam("id") UUID id){
    return companyService.getById(id);
}

@PostMapping("")
public String add(@RequestBody Company company){
    companyService.add(company);
    return "SUCCESSFUL ADDED";
}

@PutMapping("/{id}")
public String update(@PathVariable UUID id, @RequestBody Company company){
    companyService.update(id,company);
    return "SUCCESSFUL UPDATED";
}

@DeleteMapping("/{id}")
public String delete(@PathVariable() UUID id){
    companyService.delete(id);
    return "SUCCESSFUL DELETED";
}
```

SORU 4

Eğer isGiftCart metodunda exception oluşturmasını istemiyorsanız, genellikle cart.getType() çağrısının null değer döndürmemesini ve CartType sınıfındaki GIFT değeriyle karşılaştırmadan önce null kontrolü yapmanızı öneririm. Aşağıda örnek bir refactor(*) yapısını gösterebilirim:

```
public boolean isGiftCart(CartModel cart) {
    if (cart != null && cart.getType() != null) {
        return cart.getType().equals(CartType.GIFT);
    }
    return false;
    /*else{
         throw new RunTimeException("Cart ve Cart'ın tipi null olamaz");
    }*/
}
```

Bu şekilde, cart parametresi veya cart.getType() ifadesi null olduğunda bir exception fırlatılmadan false değeri döndürülür. Böylece potansiyel bir NullPointerException hatası önlenir.

SORU 5- Local database kurulumu (mysql, postgresql veya herhangi bir database)

- Java spring uygulaması ayağa kaldırılması,

- İki adet tablo yer alamalı ve bu tabloların birbirleriyle bağı olmalıdır.

Order (create date, id , total price, customer,)

Customer (id , name , age , orders)

- Java spring uygulamasında ekleme,silme,güncelleme,listeleme gibi servisler yer almalıdır ve responseda yapılan işlem detayı return edilmelidir.

- Ekleme,silme,güncelleme,listeleme işlemlerini postman vb ile işlem yapılabilmelidir.

Bu adımlar sırasıyla izlenip java uygulaması üzerinden database' e kayıt atılmalı (Herhangi bir kayıt olabilir fark etmez. Database'de bir tablo açılıp o tabloya değer girilmesi java isteği üzerinden). Daha sonra aynı istek atılan uygulama ile (örnek postman ...) get ve list java spring endpointleri çağırılarak, database e atılan kayıt response olarak dönülmeli.

- Yukarıdaki ekleme silme güncelleme listeleme işlemlerine ek olarak

1- Bir servis olmalı ve parametre olarak verilen tarihten sonra oluşturulmuş siparişleri listelesin.

2- Bir servis olmalı ve bir kelime yada harf değerini parametre olarak alsın ve isminin içerisinde bu değer geçen müşteri ve müşteriye ait sipariş id sini getirsin.

3- Bir servis olmalı ve siparişi olmayan müşterileri listesin

---

Projeyi githuba ekledim . Github linki => https://github.com/Aliimranatabey/GeneralWorkspace/tree/main/javaChallange4 .

Github üzerinden bilgisayarınıza çekerek visual studio code üzerinden açıp application.properties içerisindeki kodları kendinize göre ayarlayarak bilgisayarınızda çalıştırabilirsiniz .