Spring Veri JPA Repozitörilerinde Türetilmiş Sorgu Yöntemleri

Son değişiklik: 29 Temmuz, 2020 tarihinde

[baeldung](https://www.baeldung.com/author/baeldung/) tarafından

* [**Spring Data**](https://www.baeldung.com/category/persistence/spring-persistence/spring-data/)
* [**JPA**](https://www.baeldung.com/tag/jpa/)

**Referans Spring Data JPA kursuyla Spring Data JPA ile başlayın:**

[**>> KURSU İNCELEYİN**](https://www.baeldung.com/learn-spring-data-jpa-course)

Java ekosisteminde birkaç yıllık deneyime sahipseniz ve bu deneyimi toplulukla paylaşmak (ve tabii ki işiniz için para kazanmak) ilginizi çekiyorsa, "[Bize Yazın](https://www.baeldung.com/contribution-guidelines)" sayfasına bir göz atın. Eugen

**1. Giriş**

Basit sorgular için, **yalnızca kodumuzdaki ilgili yöntem adına bakarak** sorgunun ne olması gerektiğini türetmek kolaydır.

Bu eğiticide, [Spring Data JPA](https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa#customquery)'nın bir yöntem adlandırma kuralı biçiminde bu fikirden nasıl yararlandığını keşfedeceğiz.

**2. Spring’de Türetilmiş Sorgu Yöntemlerinin Yapısı**

**Türetilmiş yöntem adlarının, ilk *By* anahtar sözcüğüyle ayrılmış iki ana bölümü vardır:**

List<User> **findByName**(String name)

İlk kısım - *find* gibi - tanıtıcıdır ve geri kalanı - *ByName* gibi - kriterlerdir.

**Spring Data JPA bulmayı(find), okumayı(read), sorgulamayı(query), saymayı(count) ve almayı(get) destekler.** Yani, örneğin, *queryByName'i* yapabilirdik ve Spring Data aynı şekilde davranırdı.

Yinelenenleri kaldırmak veya sonuç [kümemizi sınırlandırmak](https://www.baeldung.com/jpa-limit-query-results#spring-data-jpa) için *Distinct, First* veya *Top*'u da kullanabiliriz:

List<User> **findTop3ByAge**()

**Ölçüt bölümü, sorgunun varlığa özgü koşul ifadelerini içerir.** Varlığın özellik isimleriyle birlikte koşul anahtar kelimelerini kullanabiliriz. İfadeleri *And* ve *Or* ile de birleştirebilir ve birazdan görebiliriz.

**3. Örnek Uygulama**

First, we'll, of course, need [an application using Spring Data JPA](https://www.baeldung.com/the-persistence-layer-with-spring-and-jpa).

İlk olarak, elbette [Spring Data JPA kullanan bir uygulama](https://www.baeldung.com/the-persistence-layer-with-spring-and-jpa)ya ihtiyacımız olacak.

In that application, let's define an entity class:

Bu uygulamada, bir varlık sınıfı (entity class) tanımlayalım:

@Table(name = "users")

@Entity

**class** **User** {

@Id

@GeneratedValue

**private** Integer id;

**private** String name;

**private** Integer age;

**private** ZonedDateTime birthDate;

**private** Boolean active;

// standard getters and setters

}

And, let's also define a repository. It'll extend *JpaRepository,* one of [the Spring Data Repository types](https://www.baeldung.com/spring-data-repositories):

Va ayrıca bir de depo(repozitori) tanımlayalım[. Spring Data Repository türleri](https://www.baeldung.com/spring-data-repositories)nden biri olan *JpaRepository*'yi genişletir(extend eder):

**interface** **UserRepository** **extends** **JpaRepository**<**User**, **Integer**> {}

Tüm türetilmiş sorgu yöntemlerimizi yerleştireceğimiz yer burasıdır.

**4. Eşitlik Koşulu Anahtar Kelimeleri**

Tam eşitlik, sorgularda en çok kullanılan koşullardan biridir. Sorguda = veya IS operatörlerini ifade etmek için birkaç seçeneğimiz var.

Tam eşleme koşulu için özellik(property) adını herhangi bir anahtar kelime olmadan ekleyebiliriz:

List<User> **findByName**(String name);

Ve okunabilirlik için *Is* veya *Equals* ekleyebiliriz:

List<User> **findByNameIs**(String name);

List<User> **findByNameEquals**(String name);

Bu ekstra okunabilirlik, bunun yerine eşitsizliği ifade etmemiz gerektiğinde işe yarar:

List<User> **findByNameIsNot**(String name);

Bu, *findByNameNot (String)* 'den biraz daha okunabilir!

*null* eşitlik özel bir durum olduğundan, = operatörünü kullanmamalıyız. Spring Data JPA varsayılan olarak [*null* parametreleri](https://www.baeldung.com/spring-data-jpa-null-parameters) işler. Dolayısıyla, bir eşitlik koşulu için boş bir değer ilettiğimizde, Spring sorguyu oluşturulan SQL'de IS NULL olarak yorumlar.

Sorguya IS NULL ölçütü eklemek için *IsNull* anahtar sözcüğünü de kullanabiliriz:

List<User> **findByNameIsNull**();

List<User> **findByNameIsNotNull**();

Ne *IsNull* ne de *IsNotNull*'un bir metod argümanı gerektirmediğini unutmayın.

Ayrıca herhangi bir bağımsız değişken gerektirmeyen iki anahtar kelime daha var. *boolean* türleri için eşitlik koşulları eklemek için *True* ve *False* anahtar kelimelerini kullanabiliriz:

List<User> **findByActiveTrue**();

List<User> **findByActiveFalse**();

Elbette bazen tam eşitlikten daha hafif(lenient) bir şey istiyoruz, bakalım başka ne yapabiliriz.

**5. Benzerlik Koşulu Anahtar Kelimeler**

Sonuçları bir özellik pattern’i ile sorgulamamız gerektiğinde, birkaç seçeneğimiz var.

*StartWith* kullanarak bir değerle başlayan isimleri bulabiliriz:

List<User> **findByNameStartingWith**(String prefix);

Kabaca bu, " WHERE *name* LIKE *‘value%'* ’’ olarak çevrilir.

Bir değerle biten isimler istiyorsak, istediğimiz *EndingWith*:

List<User> **findByNameEndingWith**(String suffix);

Veya, *Containing* ile hangi adların bir değer içerdiğini bulabiliriz:

List<User> **findByNameContaining**(String infix);

Yukarıdaki tüm koşulların önceden tanımlanmış kalıp ifadeleri olarak adlandırıldığını unutmayın. Dolayısıyla, bu yöntemler çağrıldığında **argümanın içine% operatörü eklememize gerek yoktur.**

But, let's suppose we are doing something more complex. Say we need to fetch the users whose names start with an *a*, contain *b,* and end with *c.*

Ama daha karmaşık bir şey yaptığımızı varsayalım. Adları *a* ile başlayan, *b* içeren ve *c* ile biten kullanıcıları getirmemiz gerektiğini varsayalım.

Bunun için *Like* anahtar kelimesi ile kendi LIKE'ımızı ekleyebiliriz:

List<User> **findByNameLike**(String likePattern);

Ve yöntemi çağırdığımızda LIKE modelimizi oluşturabiliriz:

String likePattern = "a%b%c";

userRepository.findByNameLike(likePattern);

İsimler konusunda şimdilik bu kadar yeter. *User*'daki diğer bazı değerleri deneyelim.

**6. Karşılaştırma Koşulu Anahtar Kelimeleri**

Ayrıca, < ve <= operatörlerini kullanarak kayıtları verilen değerle karşılaştırmak için *LessThan* ve *LessThanEqual* anahtar kelimelerini kullanabiliriz:

List<User> **findByAgeLessThan**(Integer age);

List<User> **findByAgeLessThanEqual**(Integer age);

Öte yandan, tam tersi durumda, *GreaterThan* ve *GreaterThanEqual* anahtar kelimeleri kullanabiliriz:

List<User> **findByAgeGreaterThan**(Integer age);

List<User> **findByAgeGreaterThanEqual**(Integer age);

Veya *Between* ile iki yaş arasındaki kullanıcıları bulabiliriz:

List<User> **findByAgeBetween**(Integer startAge, Integer endAge);

Ayrıca *In* kullanımıyla eşleşecek bir yaş koleksiyonu da sağlayabiliriz:

List<User> **findByAgeIn**(Collection<Integer> ages);

Kullanıcıların doğum tarihlerini bildiğimizden, belirli bir tarihten önce veya sonra doğmuş kullanıcıları sorgulamak isteyebiliriz. Bunun için *Before* ve *After* kullanırız:

List<User> **findByBirthDateAfter**(ZonedDateTime birthDate);

List<User> **findByBirthDateBefore**(ZonedDateTime birthDate);

**7. Çoklu Koşul İfadeleri**

*And* ve *Or* anahtar sözcüklerini kullanarak ihtiyaç duyduğumuz kadar çok ifadeyi birleştirebiliriz:

List<User> **findByNameOrBirthDate**(String name, ZonedDateTime birthDate);

List<User> **findByNameOrBirthDateAndActive**(String name, ZonedDateTime birthDate, Boolean active);

Öncelik sırası *And* ardından *Or*, tıpkı Java gibi.

**Spring Data JPA, ekleyebileceğimiz ifade sayısını sınırlamasa da, burada çıldırmamalıyız.** Uzun isimler okunamaz ve bakımı zordur. Karmaşık sorgular için **bunun yerine** [***@Query***](https://www.baeldung.com/spring-data-jpa-query) anotasyonuna bakın.

**8. Sonuçların Sıralanması**

Sırada sıralama var. Kullanıcıların *OrderBy* kullanarak adlarına göre alfabetik olarak sıralanmasını isteyebiliriz:

List<User> **findByNameOrderByName**(String name);

List<User> **findByNameOrderByNameAsc**(String name);

Ascending order is the default sorting option, but we can use *Desc* instead to sort them in reverse:

Artan sıralama (Ascending), varsayılan sıralama seçeneğidir, ancak bunları tersine sıralamak için *Desc* komutunu kullanabiliriz:

List<User> **findByNameOrderByNameDesc**(String name);

**9. *CrudRepository’de findOne ve findById***

Spring ekibi, Spring Boot 2.x ile [*CrudRepository*](https://www.baeldung.com/spring-data-repositories#crudrepository)'de bazı büyük değişiklikler yaptı. Bunlardan biri *findOne*’ı *findById* olarak yeniden adlandırmıştır.

Spring Boot 1.x ile daha önce, bir varlığı birincil anahtarıyla almak istediğimizde *findOne*'ı çağırırdık:

User user = userRepository.findOne(1);

Spring Boot 2.x'ten buyana aynı şeyi *findById* ile yapabiliriz:

User user = userRepository.findById(1);

*findById ()* metodunun bizim için *CrudRepository*'de zaten tanımlandığını unutmayın. Bu nedenle, *CrudRepository*'yi genişleten (extend eden) özel depolarda(repozitörilerde) bunu açıkça tanımlamamız gerekmez.

**10. Sonuç**

Bu yazıda, Spring Data JPA'daki sorgu türetme mekanizmasını açıkladık. Spring Data JPA depolarında(repozitörilerinde) türetilmiş sorgu yöntemleri yazmak için özellik koşulu anahtar kelimelerini kullandık.

Bu öğreticinin kaynak kodu [Github projesi](https://github.com/eugenp/tutorials/tree/master/persistence-modules/spring-data-jpa-repo)nde mevcuttur.