Spring Data JPA Mastery: 100 Temel ve İleri Düzey Konu Başlığı ve Detaylı Açıklamaları

1. **Spring Data JPA Nedir?** Spring Data JPA, Java Persistence API (JPA) tabanlı veri erişim katmanlarının uygulanmasını kolaylaştıran bir Spring Framework modülüdür. Geliştiricilerin veritabanı işlemlerini daha az kod yazarak gerçekleştirmelerini sağlar.
2. **Repository Kavramı** Repository, veri erişim katmanını soyutlayan bir arayüzdür. Spring Data JPA, CRUD (Create, Read, Update, Delete) operasyonları için hazır metodlar sunan `JpaRepository` gibi önceden tanımlanmış repository arayüzleri sağlar.
3. **@Entity Anotasyonu** JPA entity'lerini işaretlemek için kullanılan bir anotasyondur. Bu anotasyon ile işaretlenen sınıflar, veritabanı tablolarına karşılık gelir.
4. **@Id Anotasyonu** Entity sınıfındaki birincil anahtar (primary key) alanını belirtmek için kullanılır. Her entity'nin benzersiz bir kimliği olmalıdır.
5. **@GeneratedValue Anotasyonu** Birincil anahtarın otomatik olarak nasıl oluşturulacağını belirtir. Örneğin, `strategy = GenerationType.IDENTITY` ile otomatik artan bir değer kullanılabilir.
6. **CrudRepository Arayüzü** Temel CRUD işlemleri için metodlar sağlayan bir repository arayüzüdür. `save()`, `findById()`, `delete()` gibi metodları içerir.
7. **JpaRepository Arayüzü** `CrudRepository`'den türetilmiş, JPA'ya özgü ek metodlar sağlayan bir arayüzdür. Sayfalama ve sıralama gibi işlevler sunar.
8. **Query Metodları** Repository arayüzlerinde metod isimlendirme kurallarına göre otomatik olarak sorgu oluşturulan metodlardır. Örneğin, `findByLastName(String lastName)`.
9. **@Query Anotasyonu** Özel sorgular yazmak için kullanılır. JPQL (Java Persistence Query Language) veya native SQL sorguları tanımlanabilir.

10. **Paging ve Sorting** Büyük veri setlerini sayfalara bölmek ve sıralamak için kullanılan özelliklerdir. `Pageable` ve `Sort` parametreleri ile kullanılır.
11. **@ManyToOne İlişkisi** İki entity arasında çoka-bir ilişkiyi temsil eder. Örneğin, bir öğrencinin bir sınıfa ait olması.
12. **@OneToMany İlişkisi** Bire-çok ilişkiyi temsil eder. Örneğin, bir sınıfın birden fazla öğrencisi olması.
13. **@ManyToMany İlişkisi** Çoka-çok ilişkiyi temsil eder. Örneğin, öğrencilerin birden fazla kursa kaydolabilmesi ve kursların birden fazla öğrencisi olabilmesi.
14. **Cascade İşlemleri** İlişkili entity'ler üzerinde otomatik işlemler gerçekleştirmek için kullanılır. Örneğin, bir entity silindiğinde ilişkili entity'lerin de silinmesi.
15. **FetchType (LAZY ve EAGER)** İlişkili entity'lerin ne zaman yükleneceğini belirler. LAZY, ihtiyaç duyulduğunda; EAGER, ana entity ile birlikte yüklenmesini sağlar.
16. **@Transactional Anotasyonu** Veritabanı işlemlerinin atomik olarak gerçekleştirilmesini sağlar. Bir metod içindeki tüm işlemlerin ya tamamen başarılı olmasını ya da hiçbirinin gerçekleşmemesini garanti eder.
17. **Auditing** Entity'lerin oluşturulma, güncellenme tarihlerini ve kim tarafından yapıldığını otomatik olarak takip etmek için kullanılır. `@CreatedDate`, `@LastModifiedDate` gibi anotasyonlar kullanılır.
18. **Specification API** Karmaşık sorgular oluşturmak için kullanılan bir API'dir. Dinamik olarak sorgu kriterleri oluşturmayı sağlar.
19. **@Modifying Anotasyonu** `@Query` anotasyonu ile birlikte kullanılarak UPDATE veya DELETE işlemlerini gerçekleştiren sorguları işaretlemek için kullanılır.
20. **Projections** Entity'lerin belirli alanlarını seçerek sonuçları özelleştirmek için kullanılır. Performansı artırmak ve veri transferini optimize etmek için faydalıdır.
21. **@Embeddable ve @Embedded** Bir entity içinde başka bir nesneyi gömülü olarak kullanmak için kullanılır. Veritabanı normalizasyonunu korurken nesne-yönelimli tasarımı destekler.
22. **Native Queries** Veritabanına özgü SQL sorgularını kullanmak için `@Query` anotasyonuyla birlikte `nativeQuery = true` parametresi kullanılır.
23. **Named Queries** Sık kullanılan sorguları entity sınıfında veya XML dosyasında tanımlamak için kullanılır. `@NamedQuery` anotasyonu ile işaretlenir.
24. **Query By Example (QBE)** Örnek bir entity nesnesi kullanarak dinamik sorgular oluşturmayı sağlayan bir tekniktir. Karmaşık filtreleme senaryoları için kullanışlıdır.
25. **Derived Query Methods** Method isimlerinden otomatik olarak oluşturulan sorgulardır. Örneğin, `findByEmailAndName(String email, String name)`.
26. **@Version Anotasyonu** Optimistik kilitleme (optimistic locking) için kullanılır. Eşzamanlı güncellemeleri yönetmeye yardımcı olur.
27. **Inheritance Strategies** JPA'da kalıtım ilişkilerini modellemek için kullanılan stratejilerdir. Single Table, Joined Table ve Table Per Class stratejileri mevcuttur.
28. **@NamedEntityGraph** İlişkili entity'lerin yüklenmesini özelleştirmek için kullanılır. Performans optimizasyonu için faydalıdır.
29. **Composite Primary Keys** Birden fazla alanın birlikte birincil anahtar oluşturduğu durumlar için kullanılır. `@IdClass` veya `@EmbeddedId` ile uygulanabilir.
30. **Stored Procedure Calls** Veritabanında saklı prosedürleri çağırmak için kullanılır. `@Procedure` anotasyonu veya repository metodları ile yapılabilir.
31. **Auditing with Spring Data JPA** Entity'lerin oluşturulma ve güncellenme zamanlarını otomatik olarak kaydetmek için kullanılır. `@EnableJpaAuditing` ile etkinleştirilir.
32. **Custom Repository Implementations** Özel repository metodları eklemek için kullanılır. Standart CRUD işlemlerinin ötesinde özelleştirilmiş işlevsellik sağlar.
33. **@DynamicUpdate** Sadece değişen alanların güncellenmesini sağlar. Büyük entity'lerde performans artışı sağlayabilir.

34. **Batch Operations** Toplu veri işlemleri için kullanılır. `saveAll()`, `deleteAll()` gibi metodlar veya özel batch işlemleri uygulanabilir.
35. **Criteria API** Karmaşık sorguları programatik olarak oluşturmak için kullanılır. Dinamik sorgular için JPA Criteria API'sini kullanır.
36. **Locking Mechanisms** Eşzamanlı erişimi yönetmek için kullanılır. Optimistik ve pesimistik kilitleme stratejileri mevcuttur.
37. **@SQLDelete ve @SQLUpdate** Özel SQL ifadeleriyle silme ve güncelleme işlemlerini tanımlamak için kullanılır. Soft delete gibi senaryolarda faydalıdır.
38. **@Where Anotasyonu** Entity'lere global where koşulları eklemek için kullanılır. Örneğin, silinmemiş kayıtları otomatik olarak filtrelemek için.
39. **Attribute Converters** Entity alanlarını veritabanı sütunlarına dönüştürmek ve tersini yapmak için özel mantık uygulamak üzere kullanılır.
40. **Hibernate Envers ile Revision Control** Entity'lerdeki değişikliklerin geçmişini tutmak için kullanılır. Her değişiklik için ayrı bir revizyon kaydı oluşturur.
41. **@Modifying ve @Query ile Bulk Updates** Toplu güncelleme işlemleri için kullanılır. Tek bir sorgu ile birden fazla kaydı güncellemeye olanak tanır.
42. **First ve Top Keywords** Sorgu sonuçlarını sınırlamak için kullanılır. Örneğin, `findFirstByOrderByLastnameAsc()` veya `findTop3ByAge(int age)`.
43. **Projections ile DTO Dönüşleri** Sorgu sonuçlarını özel DTO (Data Transfer Object) sınıflarına doğrudan eşlemek için kullanılır.
44. **Specifications ve Criteria API Entegrasyonu** Karmaşık ve dinamik sorgular oluşturmak için Specifications ve Criteria API'nin birlikte kullanımı.
45. **@Nullable ve @NonNullApi Anotasyonları** Null değer kontrolü ve güvenliği için kullanılır. Metod parametrelerinin ve dönüş değerlerinin null olup olamayacağını belirtir.
46. **Custom ID Generators** Özel ID üretme stratejileri uygulamak için kullanılır. Örneğin, UUID veya özel bir algoritma ile ID üretimi.
47. **Entity Listeners** Entity yaşam döngüsü olaylarını dinlemek ve bu olaylara tepki vermek için kullanılır. Örneğin, `@PrePersist`, `@PostLoad` gibi.
48. **Hibernate Specific Features** Hibernate'e özgü özelliklerin Spring Data JPA ile kullanımı. Örneğin, `@NaturalId`, `@Cache` gibi.
49. **Query Hints** JPA provider'ına sorgu optimizasyonu için ipuçları vermek üzere kullanılır. Örneğin, `@QueryHints` anotasyonu.
50. **Async Query Methods** Asenkron sorgu metodları tanımlamak için kullanılır. `@Async` anotasyonu ile birlikte `CompletableFuture` veya `ListenableFuture` dönüş tipleri.
51. **Customizing Repository Base Class** Tüm repository'ler için ortak işlevsellik sağlamak üzere özel bir temel sınıf oluşturmak için kullanılır.
52. **Index Creation** Veritabanı indekslerini JPA entity'leri üzerinden tanımlamak için `@Index` anotasyonu kullanılır.
53. **Composite Repositories** Birden fazla repository arayüzünü birleştirerek daha kapsamlı repository'ler oluşturmak için kullanılır.
54. **Query by Example Executor** Örnek nesneler kullanarak dinamik sorgular oluşturmak için `QueryByExampleExecutor` arayüzü kullanılır.
55. **Lazy Attribute Loading** Entity'lerin belirli alanlarının lazy (geç) yüklenmesini sağlamak için `@Basic(fetch = FetchType.LAZY)` kullanılır.
56. **Derived Delete Queries** Method isimlendirmesi yoluyla otomatik silme sorguları oluşturmak için kullanılır. Örneğin, `deleteByLastName(String lastName)`.
57. **Entity Graphs** İlişkili entity'lerin yüklenmesini özelleştirmek için `@EntityGraph` anotasyonu kullanılır.
58. **Persistable Interface** Özel entity durum yönetimi için `Persistable` arayüzünün uygulanması.
59. **Attribute Override** Kalıtım alınan sınıflardaki alan özelliklerini geçersiz kılmak için

`@AttributeOverride` kullanılır.

60. **Lifecycle Callbacks** Entity yaşam döngüsü olaylarına tepki vermek için kullanılan metodları işaretlemek için `@PrePersist`, `@PostLoad` gibi anotasyonlar kullanılır.
61. **@Formula Anotasyonu** Veritabanı tarafında hesaplanan alanları entity'lere eklemek için kullanılır. SQL ifadeleri ile özel hesaplamalar yapılabilir.
62. **Repository Populators** Test veya geliştirme amaçlı olarak veritabanını otomatik olarak doldurmak için kullanılır.
63. **Querydsl Integration** Tip güvenli sorgular oluşturmak için Querydsl kütüphanesi ile entegrasyon sağlar.
64. **Derived Count Queries** Belirli kriterlere uyan kayıt sayısını döndüren metodlar oluşturmak için kullanılır. Örneğin, `countByLastName(String lastName)`.
65. **@Transient Anotasyonu** Entity sınıflarında veritabanına kaydedilmeyecek geçici alanları işaretlemek için kullanılır.
66. **Batch Insert/Update** Toplu ekleme ve güncelleme işlemlerini optimize etmek için kullanılır. JDBC batch işlemleriyle entegre edilebilir.
67. **@ColumnTransformer** Veritabanı sütunlarına yazarken veya okurken değerleri dönüştürmek için kullanılır.
68. **Custom Repository Base Classes** Tüm repository'ler için ortak işlevsellik sağlamak amacıyla özel temel sınıflar oluşturmak için kullanılır.
69. **Projections with Nested Properties** İç içe geçmiş property'leri içeren projeksiyonlar oluşturmak için kullanılır.
70. **@NamedSubgraph** Karmaşık entity graph'ları tanımlamak için kullanılır. İç içe ilişkileri yönetmede yardımcı olur.
71. **Auditing with Spring Security Integration** Spring Security ile entegre edilerek, mevcut kullanıcı bilgilerini otomatik olarak audit alanlarına eklemek için kullanılır.
72. **Specification Executer** Dinamik sorgu oluşturmak için `Specification` arayüzünü kullanan özel repository metodları tanımlamak için kullanılır.
73. **@Lob Anotasyonu** Büyük nesneleri (Large Objects) işaretlemek için kullanılır. Örneğin, büyük metin alanları veya dosya içerikleri.
74. **Composite Primary Key with @EmbeddedId** Birleşik anahtarları tek bir gömülü sınıf olarak tanımlamak için kullanılır.
75. **Query Derivation Mechanism** Spring Data JPA'nın method isimlerinden nasıl sorgu türettiğini anlamak önemlidir. Bu mekanizma, karmaşık sorguları method isimleriyle ifade etmeyi sağlar.
76. **@DynamicInsert** Sadece null olmayan alanların insert sorgusuna dahil edilmesini sağlar. Performans optimizasyonu için kullanılabilir.
77. **Custom Collection Mappings** Özel koleksiyon tipleriyle çalışmak için kullanılır. Örneğin, `List` yerine özel bir koleksiyon sınıfı kullanmak.
78. **Dirty Checking** JPA'nın entity'lerdeki değişiklikleri nasıl tespit ettiğini ve yönettiğini anlamak önemlidir.
79. **@SqlResultSetMapping** Native SQL sorgularının sonuçlarını entity'lere veya DTO'lara eşlemek için kullanılır.
80. **Event Publishing** Repository olaylarını yayınlamak ve dinlemek için Spring'in event mekanizmasıyla entegrasyon.
81. **@Convert ve AttributeConverter** Özel veri tipleri ile veritabanı sütunları arasında dönüşüm yapmak için kullanılır.
82. **Lazy Loading ve N+1 Problemi** Lazy loading'in avantajları ve dezavantajları, özellikle N+1 sorgu problemi ve çözümleri.
83. **Persistence Context ve First Level Cache** JPA'nın persistence context'i ve bunun first-level cache olarak nasıl çalıştığını anlamak.
84. **Second Level Cache** Hibernate veya diğer JPA sağlayıcıları ile ikinci seviye önbellek

kullanımı.

85. **Soft Delete Implementation** Kayıtları fiziksel olarak silmek yerine işaretleyerek silme işlemi gerçekleştirmek.
86. **Streaming Query Results** Büyük veri setleri için sorgu sonuçlarını stream olarak işlemek.
87. **@SqlResultSetMapping ve @ConstructorResult** Karmaşık SQL sorgularının sonuçlarını özel DTO'lara eşlemek için kullanılır.
88. **Repository Fragments** Repository arayüzlerini daha modüler hale getirmek için kullanılır. Farklı fragment'lar farklı işlevsellikler sağlayabilir.
89. **@JoinFormula** İlişkileri SQL formülleri kullanarak tanımlamak için kullanılır. Karmaşık join senaryolarında faydalıdır.
90. **Multi-Tenancy Support** Çoklu kiracı (multi-tenant) uygulamalar için veritabanı şeması veya veritabanı bazında ayrım yapmak.
91. **@FilterDef ve @Filter** Hibernate'in filter mekanizmasını kullanarak dinamik filtreler uygulamak.
92. **Optimistic Locking with @Version** Eşzamanlı güncellemeleri yönetmek için iyimser kilitleme mekanizmasının kullanımı.
93. **Pessimistic Locking** Kötümser kilitleme stratejileri ve bunların ne zaman kullanılacağı.
94. **Derived Exists Queries** Belirli bir kritere uyan kayıtların varlığını kontrol eden sorgular türetmek.
95. **@Immutable** Değişmez entity'leri işaretlemek ve bunların davranışlarını anlamak.
96. **Lifecycle Callbacks vs. Entity Listeners** Entity yaşam döngüsü olaylarını yönetmenin farklı yolları ve bunların karşılaştırılması.
97. **Bulk Delete and Update Operations** Toplu silme ve güncelleme işlemlerinin performans etkileri ve dikkat edilmesi gereken noktalar.
98. **Query Plan Cache** JPA provider'ın sorgu planı önbelleğini anlamak ve optimize etmek.
99. **Integration Testing with @DataJpaTest** Spring Boot ile JPA katmanını test etmek için `@DataJpaTest` anotasyonunun kullanımı.
100. **Performance Tuning and Best Practices** Spring Data JPA uygulamalarında performans iyileştirme teknikleri ve en iyi uygulamalar.