


# Web Programlama II

## Ders 03: JPA



# Slaytlar Hakkında

- Bu slaytlar oldukça yüksek seviyede genel bir bakış açısı sunmaktadır
  - Detaylar Git repository'sindeki yorum satırlarında bulunmaktadır
-

# Locks

- Örnek: Veritabanından bir sayaç değeri okuyup sonra 1 artırdıktan sonra tekrar veritabanına kaydedeceğinizi varsayalım
- Peki biri (örnek, thread/process) sayaç değerini okuduktan sonra alıp değiştirirse ve siz üzerine yazmak isterseniz ne olacak?
- Bu yüzden atomik operasyonlara ihtiyaç duyulur
- JPA *Optimistic* ve *Pessimistic* Locks adı verilen iki mekanizma sunar



# Optimistic Locks

- Entity @Version ile işaretlenmiş sayısal bir değere sahiptir
  - Entity içerisinde gerçekleşen her bir operasyon sonunda version artırılır
  - Tekrar değeri yazdırırken version değeri artırılmış mı artırılmamış mı kontrol edilir
    - Eğer arttıysa bir başkası tarafından paralel bir işlemle değerinin değiştirildiği belirlenmiş olur
  - Eğer version uyumsuz olursa bir exception fırlatılabilir veya bir başka operasyon gerçekleştirilmesi sağlanır
  - Optimistic: Maliyeti düşüktür ve nadir gerçekleşebilecek durumlar için uygundur
-

# Pessimistic Locks

- Doğrudan veritabanı tarafından ele alınır
- Daha maliyetlidir, atomik bir işlem tamamlanana kadar diğer threadler askıya alınacağından yüksek maliyet ortaya çıkmaktadır
- Pessimistic: Çok fazla eş zamanlı erişim bulunan noktalarda en iyi yaklaşımdır



# Threads

- İlerleyen zaman diliminde daha fazla ayrıntıya gireceğiz ancak bu ders itibari ile de kullanmaya başlıyoruz
  - Bir kod çalıştırıldığında, bir *thread* içerisinde koşar
  - *Thread'ler* işletim sistemi tarafından tahsis edilir
  - Bir process'in 1 veya daha fazla thread'i bulunabilir
  - Her bir thread'in kendine ait metot-çağrı-yığını vardır ancak aynı heap'i paylaşırlar
  - Farklı thread'ler farklı CPU'larda paralel olarak veya sıralı olarak (paralelmiş gibi görünen) çalışabilirler
-



# Thread Kullanımı

- Java'da, thread'leri **java.lang.Thread** ile durdurabilir/çalıştırabilirsiniz.
  - Genellikle ve özellikle JEE/Spring'de, **Thread**'ler doğrudan ele alınmazlar
    - Burada test ve eğitim amaçlı yapacağız
  - Bir web sunucusu bir thread havuzundan oluşur
    - Her bir gelen HTTP mesajını bir thread ele alır
    - Bir thread oluşturma/kaldırma oldukça maliyetlidir (OS kaynakları), bu yüzden havuzda var olan thread'in tekrar kullanımı en iyi yaklaşımdır
-

# Doğrulama (Validation)

- Veritabanındaki bir String ifadenin çok uzun olamayacağı nasıl söylenebilir?
- Bir String ifadenin geçerli bir mail adresi olup olmadığı nasıl söylenebilir?
- Bir Integer değerin belirtilen aralıkta olup olmadığı nasıl söylenebilir?
- @Entity alanlarına özel doğrulamalar yapılabilir
- Ayrıca özel kısıtlamalara sahip olabilir



# Validation Annotation'ları

- *@NotNull*
- *@Size(min=?, max=?)*
- *@Pattern*
- *@NotBlank*
- *@Email*
- vs.

# Kısıtlama Kullanma Sebepleri

1. Eğer bir şeyler yanlış giderse, fail fast ilkesine dayanarak olabildiği kadar hızlı hata vermesini istersiniz
2. Dokümantasyon için oldukça faydalıdır
3. Güvenlik, ör, kullanıcı adı alanına 10GB uzunluğunda bir kullanıcı adı ile gerçekleştirilebilir DOS saldırısını engellersiniz



# JPA Uygulaması

- *Hibernate* en sık kullanılan JPA uygulamasıdır
  - *EclipseLink* bir diğeridir
  - ORM'lerin rollerinden biri EntityManager ve JPQL'inizi verimli SQL kodlarına dönüştürmektir
  - Pek çok durumda bu oldukça fayda sağlar, ancak bazen verimsiz SQL kodlarına veya çeşitli tuhaf durumlara sebep olabilir
  - Hatırlatma: Kütüphanelerin bugları veya oldukça tuhaf beklenmedik davranışları bulunabilir
-

# Git Repository Modülü

- *Not: Pek çok açıklama kod içerisinde yorum satırları halinde verilmiştir*
- **intro/jee/jpa/lock**
- **intro/jee/jpa/validation**
- **intro/jee/jpa/outerjoin**
- Ders 03 alıştırması (dokümantasyona bakınız)