Lombok, Java sınıflarında sıkça yazılan getter, setter, constructor, equals, hashCode, toString gibi kodları otomatik olarak oluşturur. Böylece kodunuz daha temiz, okunabilir ve kısa olur.

column üzerindeki ayarları (örneğin nullable, unique, length gibi) hem Liquibase changelog (örneğin YAML/ XML) ile hem de JPA entity annotation ile yapabilirsin. Ama hangisi ne zaman ve nasıl tercih edilir, önemli.

Migration dosyanda DB kısıtlamalarını açıkça belirt.

Entity annotation’da da aynı kuralları yazarak kodun okunurluğunu ve uyumluluğunu koru.

Böylece hem DB hem kod birbirini destekler.

Liquibase gibi migration araçlarında **değişikliklerin farklı dosyalara bölünmesi** genellikle şu sebeplerle yapılır:

**1. Değişikliklerin Modüler ve Yönetilebilir Olması**

* Proje büyüdükçe migration dosyaları çok büyük ve karmaşık olur.
* Her değişiklik ayrı dosyada olursa:
  + Kolayca bulunur ve takip edilir,
  + Sadece ilgili dosya değiştirilir, risk azalır,
  + Merge (birleştirme) çatışmaları (conflicts) azalır.

**2. Versiyon Kontrolü ve İşbirliği**

* Takım içinde herkes farklı migration’ları ayrı ayrı geliştirebilir.
* Aynı dosyada çok kişi çalışınca çatışma olur.
* Ayrı dosyada olursa, paralel çalışmak kolaylaşır.

**3. Fonksiyonel Ayrım ve Anlamlı Gruplama**

* Örneğin:
  + user\_table.yml → kullanıcı tablosunun oluşturulması,
  + audit\_user\_table.yml → kullanıcı tablosuna audit (log) kolonlarının eklenmesi,
  + increment\_user\_table.yml → kullanıcı tablosuna yeni kolon eklenmesi veya ID arttırma stratejisi değişikliği.
* Böylece hangi dosyada ne olduğunu anlamak kolaylaşır.

**4. Migration’ların Sıra ve Kontrolü**

* Liquibase, dosyaları sırasıyla uygular.
* Farklı dosyalarda sıralamayı, değişiklik setlerini daha net kontrol edebilirsin.
* Özellikle rollback ve hata yönetimi kolaylaşır.

**5. Tekrar Kullanılabilirlik ve Modülerlik**

* Belirli migration dosyalarını sadece belirli ortamlarda ya da projelerde kullanabilirsin.
* Modüler yapıyla migration’ları kolayca paylaşabilir veya yeniden kullanabilirsin.

**Özet:**

| **Sebep** | **Açıklama** |
| --- | --- |
| Modülerlik | Büyük migration dosyalarından kaçınmak |
| Takım çalışması | Merge çatışmalarını azaltmak |
| Anlamlı gruplayabilmek | Fonksiyonel veya mantıksal parçalara ayırmak |
| Kontrol ve yönetim | Migration sırasını ve rollback’i kolaylaştırmak |

**Kısacası:**  
Migration dosyalarını mantıklı, küçük parçalara bölmek, proje sürdürülebilirliği, hata yönetimi ve takım işbirliği açısından çok faydalıdır.