Bahsedilmesi Gerekenler

* Custom Lookup oluşturulabilir.
* Custom Fields oluşturulabilir.

Documentation

1. Models and Database
   1. Models
      1. Fields
         1. ~~Field Types~~
         2. Field Options
         3. Relationships
         4. Model across files
         5. Field name restrictions
         6. Custom Field Types
      2. ~~Meta Options~~
      3. Model Methods
      4. Model Inheritance
   2. Making queries
      1. Retrieving objects
         1. Lookups that span relationships
         2. Filters can refernce fields on the model
         3. The pk lookup shortcut
         4. Caching and QuerySets
      2. Complex lookup with Q
      3. Comparing objects
      4. Related objects
   3. Aggregation
   4. Managers
   5. Database transactions
   6. ~~Multiple databases~~
   7. Database Access optimization
   8. Database instruments
   9. Model relationship API usage

------------------ BİRDEN ÇOK VERİTABANI KULLANMA ------------------

settings.py’de birden fazla veritabanı tanımlayabileceğimizi söylemiştik. veritabanları settings.py’den alias (takma ad) verilerek tanımlanmaktadır. Varsayılan veritabanı default aliası ile tanımlanır. Default aliası olması zorunludur. Eğer default olarak veritabanı kullanmak istemiyorsanız bunu boş bırakabilirsiniz. Bu durumda kullanılan her model için DATABASE\_ROUTERS tanımlanması gerekmektedir. Default veritabanı kullanım istediğiniz model için DATABASE\_ROUTERS tanımlaması da yapabilirsiniz. Mesela kullanıcı yönetimi için yapılan tüm işlemleri başka bir veritabanından yönetilmesini ayarlayabilirsiniz. Bu konu hakkında ayrıntılı bilgi için Django dokümantasyonundan yardım alabilirsiniz.

Veritabanına özel migrate işlemi yapabilirsiniz.

**./manage.py migrate --database=backup**

Farklı bir veri tabanından çekeceksek using() kullanırız. Mesela backup ismi ile yeni veri tabanı tanımladığımızı düşünelim. Kullanıcıları çekmek için using methodunda backup parametresini göndeririz. Veritabanına kayıt için ise save() methunda using parametresi olarak veritabanı adını göndeririz.

**User.objects.using(‘backup’).all()**

**object.save(using=’backup’)**

Bir veritabanındaki nesnesyi diğer veritabanına kopayalayabiliriz. Bir örnek ile inceleyelim. Veritabanımıza staff diye bir kullanıcı kayıt edelim. Kullanıcı default veritabanına eklendiğinde primary key alır. Daha sonra bu primary key ile backup veritabanına kayıt etmeye çalışır. Eğer primary key kullanılmıyorsa aynı kaydı yapar. Eğer primary key kullanılıyorsa eski kayıt üzerine yazar.

**user = User(username=’staff’)**

**user.save()**

**user.save(using=’backup’)**

Eğer üzerine yazmasını istemiyorsak primary keyi sıfırlamamız gerekir.

**user = User(username=’staff’)**

**user.save()**

**user.pk = None**

**user.save(using=’backup’)**

İstenilen veritabanından silme işlemlerinde seçilen kullanıcı hangi veritabanına ait ise silme işlemi o veritabanından gerçekleştirilir. İki veritabanı eşzamanlı olarak güncelleniyor ve bir kaydı diğerinden de silmek istiyorsanız user nesnesi defaul veritabanında olsa bile using parametre olarak silmek istediğiniz veritabanını gönderebilirsiniz.

**user.delete(using=’backup’)**

Aynı anda birden fazla veritabanı ile çalışmak için daha detaylı bilgiye Django dokümantasyonundan ulaşabilirsiniz.

------------------ DATABASE TRANSACTIONS ------------------

------------------ DATABASE SEEDING ------------------

------------------ RELATIONSHIP ------------------