der Model Manager

Was ist ein Manager?

Der Manager ist ein Interface des Models, welches einen Zugriff auf die Datenbank bietet. Jedes Model hat mindestens einen Manager.

Der Default Manager heisst objects.

Company.objects

<django.db.models.manager.Manager object at 0x7f8f09cb71c0>

Die Methoden des Managers

Ein Manager verfügt über diverse Methoden, um Ergebnisse zu liefern. Zum Beispiel:

get(): liefert ein Objekt der Klasse (z.b. ein Objekt der Klasse Company)

all(): liefert alle Objekte einer Klasse (alle Objekte von Company)

order_by(): liefert alle Objekte einer Klasse, aber geordnet nach

values(): liefert alle Objekte in Form von Dictionaries

Alle Manager-Methoden ausser get() liefern ein Queryset-Objekt!

get()

```
o = Company.objects.get(pk=2)
type(o)
<Company: Suberduber AG>
```

get liefert immer genau ein Objekt! Falls ein Objekt nicht aufgefunden wird, wird eine Modelname.DoesNotExist-Exception geworfen.

```
try:
    company = Company.objects.get(pk=42)
except Company.DoesNotExist:
     raise Http404("Diese Firma existiert nicht")
```

all()

Die Methode all() des Managers liefert alle Ergebnisse eines Models.

```
qs = Company.objects.all()
type(qs)
<Queryset>
```

Das Ergebnis der Methode all() ist ein QuerySet. Auf ein QuerySet können Filter angewandt werden (siehe Powerpoint QuerySet). Alle mit all() erstellten QuerySets beziehen sich immer auf den Ursprungsdatensatz.

all() mit Limit

```
qs = Company.objects.all()[0:2]
```

mit dem Slice-Operator können wir die Anfrage auf Einträge limitieren. Wichtig ist, hier zu verstehen, dass nicht das Ergebnis gesliced wird, sondern die SQL-Query schon die LIMIT Anweisung enthält (Performance).

Die erstellte SQL-Query würde so aussehen:

```
>>> print(qs.query)
```

```
SELECT "company_company"."id", "company_company"."name",
"company_company"."description", "company_company"."number_of_employees",
"company_company"."company_type", "company_company"."sub_title" FROM
"company_company" LIMIT 2
```

Intermezzo: Queryset

Ein QuerySet repräsentiert eine Sammlung von Objekten aus der Datenbank. Um das Ergebnis eines Querysets einzuschränken, lassen sich filter auf das Set anwenden. Der Rückgabwert eines Filters ist wieder ein Queryset.

Ein Queryset arbeitet lazy, d.h. es wird auf der Datenbank erst ausgeführt, wenn es tatsächlich evaluiert wird.

https://docs.djangoproject.com/en/3.1/ref/models/querysets/#when-querysets-are-evaluated

Queryset: Beispiel Filter()

```
qs = Company.objects.all()
qs = qs.filter(name__startswith="A“)
qs = qs.filter(description__icontains="AG")

qs
<QuerySet [<Company: Superduper AG>]>
```

https://docs.djangoproject.com/en/3.1/ref/models/querysets/

order_by()

order_by() erzeugt ein geordnetes Queryset und entspricht der ORDER BY Klausel in SQL. Wird order_by() direkt auf dem Manager ausgeführt, bekommen wir den ganzen Datensatz geliefert.

```
qs = Company.objects.order_by('name')
```

Absteigend sortieren:

```
qs = Company.objects.order_by('-name')
```

Order_by auf Queryset anwenden:

```
qs = Company.objects.all().filter(name__startswith='A').order_by('name', 'date')
```