# 06 - Django ORM (CRUD) & Admin Interface

## Einleitung

- **Themen**: Dieses Skript baut direkt auf den erstellten Modellen auf und behandelt die praktische Interaktion mit der Datenbank.
- **Fokus**: Durchführung von CRUD-Operationen (Create, Read, Update, Delete) über das Django Object-Relational Mapping (ORM). Zwei zentrale Werkzeuge werden vorgestellt: die interaktive Django Shell und das mächtige, von Django bereitgestellte Admin Interface.
- **Lernziele**:
  - Das Konzept von CRUD-Operationen verstehen.
  - Die Django Shell nutzen, um interaktiv mit der Datenbank zu arbeiten.
  - Verschiedene Methoden zur Erstellung, zum Auslesen, Aktualisieren und Löschen von Datenbankeinträgen anwenden können.
  - Das Django Admin Interface einrichten, einen Superuser erstellen und eigene Modelle dort registrieren.

## 1. Das Django ORM: Die Brücke zur Datenbank

Das Django ORM (Object-Relational Mapping) ist die Schicht, die es uns ermöglicht, mit der Datenbank über Python-Objekte zu kommunizieren, anstatt SQL-Code schreiben zu müssen. Jede Interaktion mit den Model-Klassen (z.B. `Recipe.objects.all()`) wird vom ORM in eine entsprechende Datenbankabfrage übersetzt.

## 2. Die Django Shell: Das interaktive Testlabor

Bevor wir Logik in Views schreiben, ist es extrem nützlich, ORM-Abfragen interaktiv zu testen. Die Django Shell ist dafür das perfekte Werkzeug. Sie lädt die gesamte Django-Projektumgebung, sodass man direkten Zugriff auf alle Modelle hat.

- **Starten der Shell**:

  ```
  python manage.py shell
  ```

Innerhalb der Shell kann man nun die Modelle importieren und direkt mit ihnen arbeiten.

```
# In der Django Shell
from polls.models import Question, Choice
```

## 3. CRUD-Operationen mit dem ORM

CRUD steht für die vier grundlegenden Funktionen der Datenpersistenz: **C**reate, **R**ead, **U**pdate, **D**elete.

### 3.1 Create (Erstellen von Einträgen)

Beim Erstellen von neuen Datenbankeinträgen müssen nicht zwingend alle Felder des Modells mit Werten versorgt werden.

- Felder, die in `models.py` einen `default`-Wert definiert haben, erhalten diesen automatisch, wenn kein anderer Wert angegeben wird.
- Felder, die `null=True` gesetzt haben, können weggelassen werden; Django speichert dann `NULL` in der Datenbank.
- Felder ohne `default` und ohne `null=True` (und `blank=False`) sind Pflichtfelder und müssen beim Erstellen einen Wert erhalten, sonst gibt es einen Fehler.
- Automatisch generierte Felder (wie `id` oder Felder mit `auto_now_add=True`) werden von Django verwaltet.

Es gibt zwei primäre Methoden, um neue Objekte in der Datenbank zu erstellen:

- **Methode 1: `objects.create()`** Diese Methode erstellt ein neues Objekt und speichert es in einem einzigen Schritt in der Datenbank. Sie ist kurz und effizient, wenn keine weiteren Operationen vor dem Speichern nötig sind.

  **Beispiel:**

  ```
  ```

```
# In der Django Shell
from django.utils import timezone

q1 = Question.objects.create(question_text="Was ist deine Lieblingsfarbe?",
pub_date=timezone.now())
# q1.id ist jetzt verfügbar
```

Wenn `pub_date` einen `default`-Wert hätte oder `null=True` wäre, könnte man es hier weglassen:

```
# q2 = Question.objects.create(question_text="Noch eine Frage?") # Würde Default für
pub_date nehmen
```

- **Methode 2: Objekt-Instanzierung + `.save()`** Bei dieser Methode wird zuerst eine Python-Objekt-Instanz des Modells erstellt. Diese existiert zunächst nur im Speicher. Erst durch den Aufruf der `.save()`-Methode wird der Eintrag in die Datenbank geschrieben. Dieser Ansatz bietet mehr Kontrolle, da man das Objekt vor dem Speichern noch verändern oder validieren kann.

  **Beispiel:**

```
# In der Django Shell
from django.utils import timezone

new_q = Question(question_text="Wie ist das Wetter heute?")
# Wenn pub_date ein Pflichtfeld ist, muss es vor .save() gesetzt werden:
new_q.pub_date = timezone.now()
# Man könnte auch hier Felder mit Default oder null=True weglassen.

new_q.save()
# new_q.id ist jetzt verfügbar
```

### 3.2 Read (Auslesen von Einträgen)

- **Alle Objekte holen: `.all()`** Gibt ein `QuerySet` (eine Art Liste) aller Objekte eines Modells zurück.

```
all_questions = Question.objects.all()
print(all_questions)
```

- **Ein einzelnes Objekt holen: `.get()`** Holt genau ein Objekt, das den Kriterien entspricht. Löst einen Fehler aus, wenn kein oder mehr als ein Objekt gefunden wird.

```
# Holt das Objekt mit der ID 1
q1 = Question.objects.get(id=1)

# Holt das Objekt, dessen Text genau "FRAGE" ist
q_frage = Question.objects.get(question_text="FRAGE")
```

- **Objekte filtern: `.filter()` und Field Lookups (`__`)** Gibt ein `QuerySet` aller Objekte zurück, die den Filterkriterien entsprechen. Die Kriterien werden als Keyword-Argumente übergeben. Die Magie von `.filter()` liegt in den **Field Lookups**. Diese werden mit einem doppelten Unterstrich (`__`) an den Feldnamen angehängt und erlauben spezifischere Abfragen.

  **Syntax**: `ModelName.objects.filter(feldname__lookuptyp='wert')`

  **Wichtige Field Lookups**:

  - `exact`: Exakte Übereinstimmung (Standard, wenn kein Lookup angegeben wird).

```
```

```
q_exact = Question.objects.filter(question_text__exact="Was ist das?")
# Kurzform:
q_exact_short = Question.objects.filter(question_text="Was ist das?")
```

- `iexact`: Exakte Übereinstimmung, ohne Berücksichtigung der Groß-/Kleinschreibung.

```
q_iexact = Question.objects.filter(question_text__iexact="was ist das?")
```

- `contains`: Enthält den angegebenen String.

```
q_contains = Question.objects.filter(question_text__contains="Lieblings")
```

- `icontains`: Enthält den angegebenen String, ohne Berücksichtigung der Groß-/Kleinschreibung.

```
q_icontains = Question.objects.filter(question_text__icontains="lieblings")
```

- `startswith` / `istartswith`: Beginnt mit dem String (case-sensitive / case-insensitive).

```
q_startswith = Question.objects.filter(question_text__startswith="Was")
```

- `endswith` / `iendswith`: Endet mit dem String (case-sensitive / case-insensitive).

```
q_endswith = Question.objects.filter(question_text__iendswith="?")
```

- `gt` (greater than), `gte` (greater than or equal to): Größer als / größer gleich.

```
# Fragen mit ID größer als 5
q_gt = Question.objects.filter(id__gt=5)
```

- `lt` (less than), `lte` (less than or equal to): Kleiner als / kleiner gleich.

```
# Choices mit weniger als 10 Stimmen
c_lte = Choice.objects.filter(votes__lte=10)
```

- `in`: Der Wert des Feldes ist in der übergebenen Liste enthalten.

```
q_in = Question.objects.filter(id__in=[1, 3, 5])
```

- `isnull`: Prüft, ob ein Feld `NULL` ist (`True`) oder nicht (`False`).

```
# Angenommen, ein Feld 'optional_notes' könnte NULL sein
# q_isnull = Question.objects.filter(optional_notes__isnull=True)
```

- Für Datums-/Zeitfelder gibt es spezifische Lookups wie `year`, `month`, `day`, `week_day`, etc.

```
```

```
from django.utils import timezone
current_year = timezone.now().year
q_year = Question.objects.filter(pub_date__year=current_year)
```

Man kann mehrere Filterkriterien verketten (entspricht einem logischen UND):

```
q_combined = Question.objects.filter(question_text__icontains="farbe",
pub_date__year=current_year)
```

### 3.3 Update (Aktualisieren von Einträgen)

- **Methode 1: Objekt holen, ändern, `.save()`** Dies ist die gängigste Methode. Ein Objekt wird aus der DB geholt, seine Attribute werden geändert und dann wird `.save()` aufgerufen, um die Änderungen zu speichern.

  ```
  q = Question.objects.get(id=1)
  q.question_text = "Neue Frage?"
  q.save()
  ```

- **Methode 2: `.update()` auf einem QuerySet** Diese Methode ist effizienter für die Aktualisierung mehrerer Einträge auf einmal, da sie direkt auf Datenbankebene arbeitet.

  ```
  Question.objects.filter(id__in=[1,2,3]).update(question_text="Einheitlicher Text")

  # Aktualisiert alle Fragen, die mit "Was" beginnen
  Question.objects.filter(question_text__startswith="Was").update(question_text="Eine neue
  Frage")
  ```

### 3.4 Delete (Löschen von Einträgen)

Um ein Objekt zu löschen, holt man es zuerst und ruft dann die `.delete()`-Methode auf.

**Beispiel:**

```
q = Question.objects.get(id=2)
q.delete()
```

## 4. Das Django Admin Interface: Datenverwaltung per Klick

Django kommt mit einer voll funktionsfähigen, produktionsreifen Admin-Oberfläche. Sie liest die Metadaten aus den Modellen, um eine schnelle, modell-zentrierte Schnittstelle bereitzustellen, mit der man Inhalte verwalten kann.

- **Einen Superuser anlegen** Um sich im Admin-Bereich anmelden zu können, benötigt man einen Benutzer mit Superuser-Status.

  ```
  python manage.py createsuperuser
  ```

  Anschließend wird man aufgefordert, einen Benutzernamen, eine E-Mail und ein Passwort einzugeben. [cite: 26]

- **Modelle im Admin registrieren** Standardmäßig werden keine Modelle im Admin angezeigt. Man muss sie explizit in der `admin.py`-Datei der jeweiligen App registrieren. [cite: 26, 517]

  **Beispiel für `polls/admin.py`:**

  ```
  ```

```
from django.contrib import admin
from .models import Question, Choice

admin.site.register(Question)
admin.site.register(Choice)
```

Nach dem Speichern der Datei und dem Neustarten des Entwicklungsservers sind die Modelle `Question` und `Choice` im Admin-Bereich unter `http://127.0.0.1:8000/admin/` sichtbar und können dort verwaltet werden.

### Fazit

- **ORM für CRUD**: Das Django ORM bietet intuitive Methoden (`.create()`, `.get()`, `.filter()`, `.save()`, `.update()`, `.delete()`), um Daten zu verwalten, ohne SQL schreiben zu müssen.
- **Django Shell**: Ein unverzichtbares Werkzeug zum interaktiven Testen von Datenbankabfragen und zur schnellen Datenmanipulation.
- **Admin Interface**: Eine extrem zeitsparende Funktion von Django, die nach einfacher Registrierung der Modelle eine komplette Weboberfläche zur Datenverwaltung bereitstellt.
- **Zwei Wege zur Erstellung**: `.create()` ist schnell und direkt. Die Instanzierung mit anschließendem `.save()` bietet mehr Kontrolle.

---

### Projekt-Anwendung (Leitfaden-Projekt)

Für das "Online-Umfragesystem" werden die Modelle im Admin-Interface registriert, um die ersten Daten einfach per Weboberfläche eingeben zu können.

**Aufgabe für die Demonstration:**

1. **Modelle registrieren**: Die Datei `polls/admin.py` öffnen und die Modelle `Question` und `Choice` registrieren.

   ```
   # polls/admin.py
   from django.contrib import admin
   from .models import Question, Choice

   admin.site.register(Question)
   admin.site.register(Choice)
   ```

2. **Superuser erstellen**: Den Befehl im Terminal ausführen:

   ```
   python manage.py createsuperuser
   ```

3. **Daten anlegen**: Den Entwicklungsserver starten (`python manage.py runserver`), zum Admin-Bereich navigieren (`/admin/`) und sich mit dem Superuser einloggen. Dort mehrere Fragen (`Question`) und zugehörige Antworten (`Choice`) erstellen.

---

### Cheat Sheet

**CRUD-Befehle mit dem ORM**

- **Create**:
  - `MyModel.objects.create(pflichtfeld='wert', optionalfeld='wert')`
  - `obj = MyModel(pflichtfeld='wert'); obj.optionalfeld='wert'; obj.save()`
- **Read**:
  - `MyModel.objects.all()`
  - `MyModel.objects.get(id=1)`
  - `MyModel.objects.filter(feldname__lookuptyp='wert')` (z.B. `textfeld__icontains='suche'`)
- **Update**:
  - `obj = MyModel.objects.get(id=1); obj.feld1='neu'; obj.save()`

    - `MyModel.objects.filter(...).update(feld1='neu')`
- **Delete**:
    - `obj = MyModel.objects.get(id=1); obj.delete()`

**Wichtige Field Lookups für `.filter()`**

- `__exact` (oder ohne Lookup): Exakte Übereinstimmung.
- `__iexact`: Exakt, Case-Insensitive.
- `__contains`: Enthält Teilstring.
- `__icontains`: Enthält Teilstring, Case-Insensitive.
- `__startswith` / `__istartswith`: Beginnt mit.
- `__gt` / `__gte`: Größer als / Größer gleich.
- `__lt` / `__lte`: Kleiner als / Kleiner gleich.
- `__in`: Wert ist in einer Liste.
- `__isnull`: Ist `True` oder `False`.

**Admin Interface**

- **Superuser erstellen**:

```
python manage.py createsuperuser
```

- **Modell registrieren** (in `app/admin.py`):

```
from django.contrib import admin
from .models import MyModel

admin.site.register(MyModel)
```

---

## Übungsaufgaben

1. **Django Shell für CRUD nutzen**:
    - Das `Product`-Modell aus der vorherigen Übung verwenden.
    - Die Django Shell starten.
    - Das `Product`-Modell importieren.
    - Mindestens drei neue Produkte mit der `.create()`-Methode erstellen. Achten Sie darauf, alle notwendigen Felder zu befüllen.
    - Alle Produkte ausgeben, deren Name "Laptop" enthält (case-insensitive).
    - Alle Produkte filtern, deren Preis größer als 50.00 ist.
    - Den Preis eines Produkts aktualisieren und speichern.
2. **Admin-Bereich füllen**:
    - Das `Product`-Modell in der `admin.py`-Datei der App registrieren.
    - Einen Superuser erstellen, falls noch nicht geschehen.
    - Sich im Admin-Bereich anmelden und dort manuell weitere Produkte anlegen und bestehende bearbeiten/löschen.

---

## Schüler-Projekt (Eigenständig): Community Recipe Sharing Platform

Die erstellten Modelle werden nun mit ersten Daten gefüllt. Dies geschieht sowohl über das Admin Interface als auch über die Django Shell, um beide Methoden zu üben.

**Aufgabe:**

1. **Modelle im Admin registrieren**:
    - Die Datei `recipes/admin.py` öffnen.
    - Die Modelle `Recipe`, `Ingredient` und `Step` aus `recipes.models` importieren.

- Alle drei Modelle mit `admin.site.register()` registrieren.

2. **Superuser erstellen und Daten im Admin anlegen**:
   - Falls noch nicht geschehen, einen Superuser mit dem Befehl `python manage.py createsuperuser` erstellen.
   - Den Entwicklungsserver starten und den Admin-Bereich unter `/admin/` aufrufen.
   - Sich mit den Superuser-Daten anmelden.
   - Mindestens **zwei vollständige Rezepte** über die Admin-Oberfläche anlegen. Das bedeutet:
     - Ein `Recipe`-Objekt erstellen (z.B. "Spaghetti Carbonara").
     - Mehrere `Ingredient`-Objekte erstellen (z.B. "Spaghetti", "Ei", "Pecorino", "Guanciale").
     - Mehrere `Step`-Objekte erstellen, die die Zubereitung beschreiben. *(Hinweis: Die direkte Verknüpfung zwischen diesen Modellen über das Admin-Interface wird im nächsten Skript durch die Definition von Beziehungen (`ForeignKey`, `ManyToManyField`) und Anpassungen im Admin erleichtert. Fürs Erste werden die Objekte einzeln erstellt.)*

3. **CRUD in der Django Shell üben**:
   - Die Django Shell starten.
   - Die Modelle `Recipe` und `Ingredient` importieren.
   - Ein neues Rezept-Objekt nur mit Python-Code erstellen (z.B. "Pancakes"). Achten Sie darauf, alle Pflichtfelder zu setzen.
   - Eine neue Zutat ("Mehl") mit `objects.create()` erstellen.
   - Alle Zutaten ausgeben, deren Name mit "M" beginnt (case-insensitive).
   - Den Titel des "Pancakes"-Rezepts ändern.
   - Eine testweise erstellte Zutat wieder löschen.