Съдържание

[1. Увод 2](#_Toc194392432)

[2. Концепция на работа с бази данни в Django 3](#_Toc194392433)

[2.1 Основи на работа с бази данни в Django 3](#_Toc194392434)

[1.Модели 3](#_Toc194392435)

[2. Връзки между модели 4](#_Toc194392436)

[3. Реализация на обектно-релационни модели в Django 6](#_Toc194392437)

[Въведение 6](#_Toc194392438)

[3.1 Основи на обектно-релационните модели 6](#_Toc194392439)

[1. Дефиниране на модели 6](#_Toc194392440)

[2. Връзки между модели 7](#_Toc194392441)

[**Работа с модели в Django** 8](#_Toc194392442)

[4. Видове полета и заявките в Django 9](#_Toc194392443)

[Въведение 9](#_Toc194392444)

[4.1 Видове полета в Django 9](#_Toc194392445)

[2. Допълнителни типове полета 10](#_Toc194392446)

[4.2 Заявки в Django 11](#_Toc194392447)

[3. Заключение 13](#_Toc194392448)

[4. Литература: 14](#_Toc194392449)

# Увод

Описание на целите и задачите на проекта.

Значимост на избраната тема.

# Концепция на работа с бази данни в Django

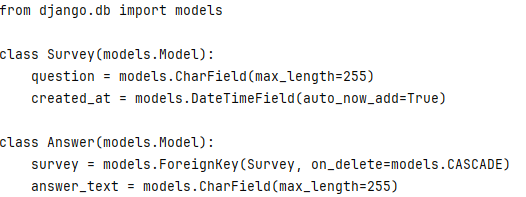
**Въведение**

Работата с бази данни е основен аспект на повечето уеб приложения. При разработката на приложения с Django, един от най-силните аспекти на фреймуорка е неговата интеграция с бази данни. Django предоставя мощни инструменти за работа с релационни бази данни, което позволява на разработчиците лесно да извършват операции по създаване, четене, актуализиране и изтриване на данни (CRUD операции).

## Основи на работа с бази данни в Django

### 1.Модели

В Django, базите данни се представят чрез модели. Моделите определят структурата на данните и как те ще бъдат организирани в базата данни. Django използва проект, наречен Object-Relational Mapping (ORM), който позволява на разработчиците да работят с базата данни чрез Python класове вместо чрез SQL команди. Всеки модел е свързан с таблица в базата данни и атрибутите на модела съответстват на колоните в таблицата.Пример за дефиниране на модел в Django:



В примера по-горе, дефинираме два модела: Survey и Answer. Тези класове ще позволят на приложението да работи с таблици, свързани с анкети и техните отговори.

### 2. Връзки между модели

Django дава възможност за създаване на различни типове връзки между моделите:

* **Едно-към-едно (One-to-One):** Позволява свързване на два модела по такъв начин, че за всяка единица от единия модел, може да съществува само една единица от другия.
* **Едно-към-много (One-to-Many):** Най-често срещаният тип връзка, където един запис от един модел може да се свърже с множество записи от друг модел.
* **Много-към-много (Many-to-Many)**: Позволява на множество записи от един модел да съществуват в отношения с множество записи от друг модел.

Чрез използването на ForeignKey и ManyToManyField полета във модела, Django автоматично управлява сложността на релациите.

**3. Видове полета**

Django предлага разнообразие от типове полета, които могат да се използват в моделите:

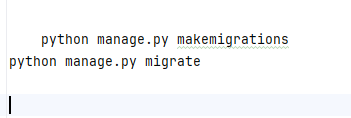
* CharField: За текстови низове с ограничена дължина.
* TextField: За по-дълги текстови стойности.
* IntegerField: За съхранение на цели числа.
* DateTimeField: За съхранение на дати и часове.
* BooleanField: За булеви стойности (истина/лъжа).

Изборът на правилния тип поле е важен, тъй като той диктува как ще бъдат съхранявани и манипулирани данните в базата.

**Работа с бази данни**

**1. Инициализация и миграции**

След дефинирането на моделите, следва следващата стъпка — инициализация на базата данни. Django използва система за миграции, която проследява промените в моделите и синхронизира базата данни с новите структури. Миграциите позволяват на разработчиците да добавят, променят или изтриват таблици и полета, без да губят данни.



Командите по-горе генерират скриптове за миграции и прилагат промените в базата данни.

**2. Извършване на заявки с ORM**

Django ORM позволява извършване на заявки чрез използване на Python код. Разработчиците могат да извършват CRUD операции по различни начини, без да пишат директно SQL. Например, за да получим всички анкети, можем да напишем:



За да добавим нов отговор:



Тази простота е основна причина Django да бъде предпочитан при разработка на приложения, свързани с бази данни.

# Реализация на обектно-релационни модели в Django

## Въведение

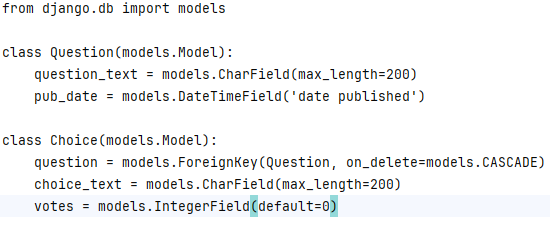
Обектно-релационното моделиране (ORM) е концепция, която свързва обектно-ориентираното програмиране с релационните бази данни. В Django, ORM позволява на разработчиците да взаимодействат с базата данни, използвайки Python класове вместо SQL заявки. Това подобрява производителността и намалява сложността при работа с данни. В този раздел ще разгледаме как се реализират обектно-релационни модели в Django, как се създават модели, как взаимодействат различните модели помежду си, и как ORM опростява манипулацията на данни.

## Основи на обектно-релационните модели

### 1. Дефиниране на модели

В Django, всеки модел представлява таблица в базата данни. Моделите се дефинират като Python класове, наследяващи на models.Model, и всеки атрибут на класа представлява колона в таблицата. При дефиниране на моделите, можем да зададем различни типове полета, които описват вида на данните, които ще се съхраняват.

Пример:



В примера, Question и Choice дефинират две таблици в базата данни, свързани помежду си чрез ForeignKey, което показва множество отговори (избори) за един въпрос.

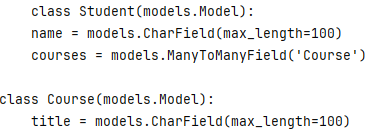
### 2. Връзки между модели

Django предоставя различни механизми за създаване на връзки между модели, включително:

**Едно-към-едно**: Дефинира се с OneToOneField. Ползва се, когато искаме един запис от единия модел да съответства на точно един запис от другия модел.

**Едно-към-много**: Дефинира се с ForeignKey. Например, в горния код, Choice има ForeignKey към Question, което означава, че един въпрос може да има множество избори.

**Много-към-много**: Дефинира се с ManyToManyField. Този тип връзка е полезен, когато искаме множество записи от един модел да са свързани с множество записи от друг модел.



Тук, един студент може да бъде записан в множество курсове и всеки курс може да има много студенти.

**Работа с модели в Django**

1. **Създаване и миграции**

След като сме дефинирали моделите, следва следващата стъпка — миграции. Миграциите са инструмент за синхронизиране на моделите с базата данни. Django автоматично генерира миграции въз основа на промените в моделите.

****

Командата makemigrations създава нова миграция, а migrate прилага промените в базата данни, създавайки новите таблици или актуализирайки съществуващите.

1. **Манипулиране на данни**

След като моделите са създадени и мигрирани, можем да взаимодействаме с тях чрез Django ORM. Основните операции, които можем да извършваме, включват:

**Създаване** на нови записи:

****

**Четене на записи:**

****

**Актуализиране на записи:**

****

Тези операции ни позволяват лесно да управляваме данни, без да пишем SQL заявки.

# Видове полета и заявките в Django

## Въведение

Django е мощен уеб фреймуорк, който предоставя интуитивни механизми за работа с релационни бази данни. Важно е да разберем различните типове полета, които можем да използваме в моделите, и как да извършваме заявки за манипулация на данни чрез Django ORM. Тази част от проекта ще разгледа широко наличните полета, техните характеристики и типичните заявки, които можем да извършваме

## Видове полета в Django

#### ****1. Основни типове полета****

* Django предоставя разнообразие от полета, с които можем да моделираме данни в базата. Основните типове полета включват:

**CharField**: Използва се за кратки текстови стойности.

Например:



**TextField**: За по-дълги текстове, когато е необходимо:



**IntegerField**: За цели числа:



**FloatField**:

* Използва се за дробни числа.
* Пример:



**BooleanField**:

* Използва се за логически стойности (True/False).
* Пример:



**DateField и DateTimeField**:

* Използват се за дати и дати с времева информация.
* Пример:



**EmailField**:

* Използва се за съхраняване на имейл адреси.
* Пример:



## ****2. Допълнителни типове полета****

**FileField**:

* Използва се за качване на файлове
* Пример:



**ImageField**:

* Използва се за качване на изображения.
* Пример:



**URLField**:

* Използва се за съхраняване на URL адреси.
* Пример:



## Заявки в Django

Заявките в Django се извършват чрез ORM, което позволява на разработчиците да работят с бази данни чрез Python обекти, вместо да пишат SQL заявки. Това не само опростява работата с базата данни, но също така я прави по-сигурна.

1. **Основни операции**

# Заключение

# Литература:

1. Колисниченко, Денис, Адаптивен уеб дизайн с Bootstrap, Асеневци, С 2019
2. D.K Academy, Python – практическо програмиране. Асеневци, С 2022
3. Донълдсън, Тоби. Бързо ръководство:Програмиране с Python.АлексСофт, С 2017
4. Есканази, Аврам. Софтуерни техологии. КЛМН, С 2006
5. https://softuni.bg/blog/what-is-django
6. 2. https://cynoteck.com/bg/blog-post/flask-vs-django/
7. 3. https://bg.savtec.org/articles/coding/the-basics-of-rest-and-restful-api-development.html
8. 4. https://urocibg.eu/
9. 5. https://www.django-rest-framework.org/
10. 6. https://softuni.bg/blog/vue-js-january-2021
11. 7. https://bg.wikipedia.org/wiki/MySQL
12. 8. https://priobshti.se/article/strategii-v-pomosht-na-prepodavaneto/taksonomiya-na-blum-ili-kak-da-napravim-uroka-v
13. 9. https://html.w3schools.bg
14. 10. https://www.inventum.bg/web-design/css/