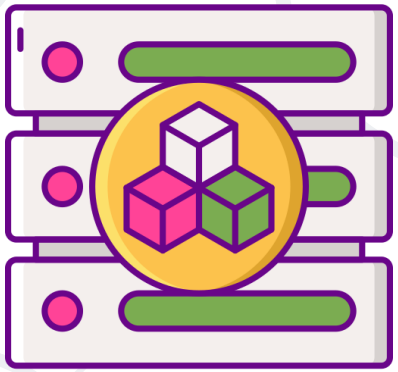
Django Models Basics



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

#python-db

Table of Contents

Introduction to Models

Django Models

- Models define the **structure** of **stored data**
  - Containing the essential **fields** and **behaviors** of the data
- Each model maps to a single **database table**
- Django Model is a Python class that subclasses **`django.db.models.Model`**
- Each attribute of the model represents a **database field**

Models Benefits

- Work with database data **using Python code**
  - Don't have to write **low-level SQL** queries
  - Focus on the **data** and the **business logic**
  - Django **automatically** creates the needed queries and executes them

Defining Models

Defining a Model

- Each Django application has a **models.py** file
- Create your **model** there. You need to subclass **models.Model**

tasks/models.py

```
from django.db import models

class Task(models.Model):
    title = models.CharField(max_length=50)
    text = models.TextField()
```

Model Name

Field Types

Fields

Fields

- The **most important** and **required** part of a model
  - Field names should not conflict with **reserved words**
  - Field names cannot have **more than one underscore** in a row and cannot **end with** an **underscore**
- Each field is **an instance** of the appropriate **Field class**

```
class Employee(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=40)
```

Field Types

- They determine the **column type** in a database table (e.g., INTEGER, VARCHAR, TEXT)
- Django has dozens of **built-in field** types
- Technically, they are defined in **django.db.models.fields**
- For convenience they're imported into **django.db.models**

```
from django.db import models

class Employee(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=40)
```

Standard convention

String Field Types

- `CharField`
  - Appropriate for small- to large-sized strings
  - Has one extra argument - **`max_length`** (**required** for all database backends included with Django **except** PostgreSQL)
- `TextField`
  - Appropriate for large texts
  - When specifying max length, it **won't be enforced** at the model or database level

Numeric Field Types

- `IntegerField`
  - Stores integers
- `PositiveIntegerField`
  - Stores integers that could be either **positive** or **zero**
- `FloatField`
  - Stores **floating-point** numbers
- `DecimalField`
  - Stores **fixed-precision** decimal numbers
  - Two required arguments - **`max_digits`** and **`decimal_place`**

Date/Time Field Types

- `DateField` - stores a date
- `TimeField` - stores a time
- `DateTimeField` - stores a date and a time
- They have two extra field arguments (not required):
  - `auto_now`
    - Sets the field to now **every time the object is saved**
  - `auto_now_add`
    - Sets the field to now when the object is **first created**

More Useful Field Types

- `BooleanField`
  - Stores Booleans - either `True` or `False`
- `URLField`
  - CharField for URLs
  - `max_length` is 200 by default
- `EmailField`
  - CharField that **checks** if the value is a **valid email address**
  - `max_length` is 254 by default

More Field Types: *https://docs.djangoproject.com/en/5.0/ref/models/fields/#field-types*

Field Arguments

- A certain set of **field-specific** or **common arguments**
  - **max_length** argument specifies the size of the VARCHAR field. It is a **field-specific, required** argument
  - **null**, **blank**, **default**, **primary_key**, etc. are **common optional** arguments
- If you do not specify **primary_key=True** for any field in your model, Django will automatically add an **IntegerField** to hold the primary key

Problem: Employee Model

- Create a model called "**`Employee`**" with the following fields:
  - **`name`**: char field; max length of 30 chars
  - **`email_address`**: email field
  - **`photo`**: URL field
  - **`birth_date`**: date field
  - **`works_full_time`**: Boolean field
  - **`created_on`**: date and time field; set to now when the object is first created

Solution: Employee Model

```
class Employee(models.Model):
    name = models.CharField(max_length=30)
    email_address = models.EmailField()
    photo = models.URLField()
    birth_date = models.DateField()
    works_full_time = models.BooleanField()
    created_on = models.DateTimeField(auto_now_add=True)
```

Model vs SQL Query

- Creating model **Employee** in the app **employees**

```
class Employee(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=40)
```

- It will create a database table like the following

```
CREATE TABLE employees_employee (
"id" BIGINT NOT NULL PRIMARY KEY,
"first_name" VARCHAR(30) NOT NULL,
"last_name" VARCHAR(40) NOT NULL
);
```

id is added automatically

Model Field Options in Details

Field Options

- Common SQL **constraints** but in Python code
- Available for **all** field types
- All of them are **optional**

```
class Employee(models.Model):
    ...
    email_address = models.EmailField(unique=True)
```

field option

- **Note**: they are **NOT field-specific** arguments

Default vs Unique

- `default`
  - A **default value** or a **default callable object** for the field
- `unique`
  - False by default
  - If True, this **field must be unique** for the table column

```
class Employee(models.Model):
    ...
    works_full_time = models.BooleanField(default=True)
    job_level = models.CharField(max_length=30, default='Junior')
    business_account = models.CharField(max_length=30, unique=True)
```

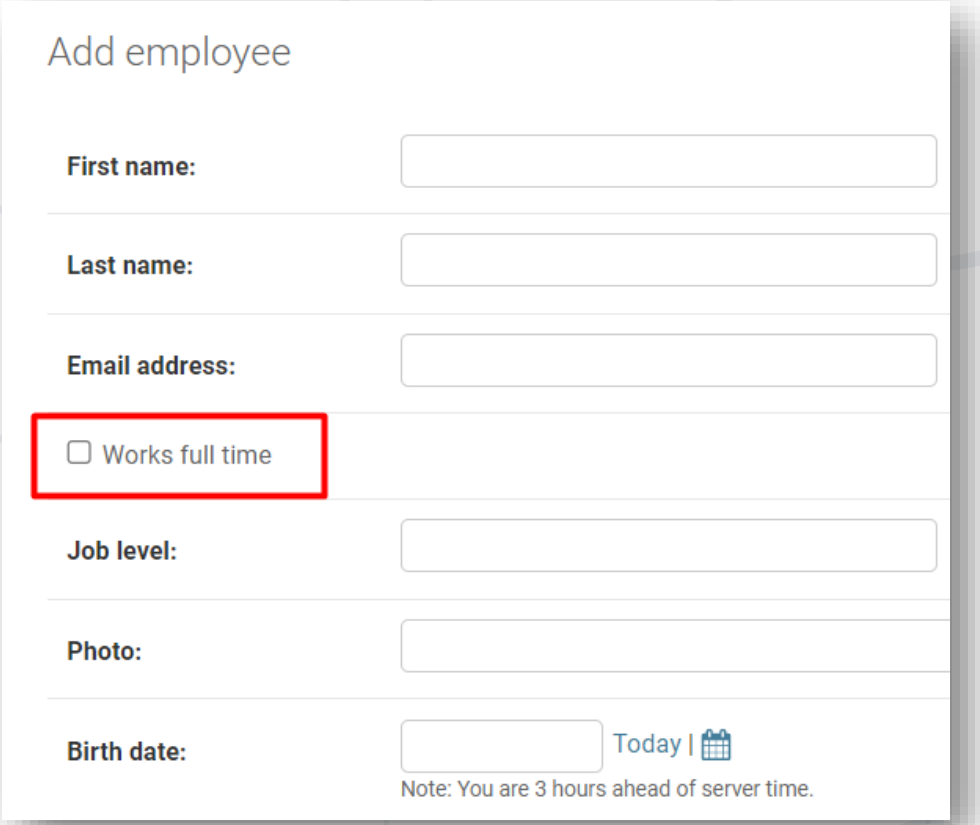
Null vs Blank

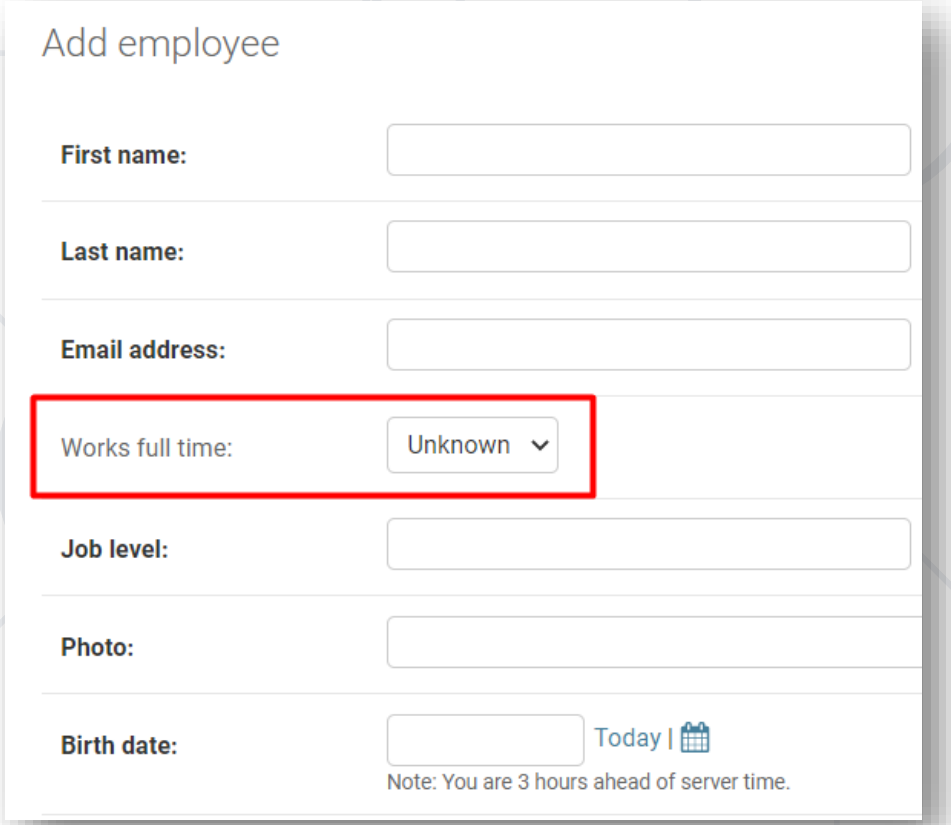
- `null` - database-related
  - **False** by default. If **True**, empty values will be stored as **NULL**
  - Use for **non-string fields** such as integers, Booleans, and dates
- `blank` - validation-related
  - **False** by default. If **True**, the field is allowed to be blank

```
class Employee(models.Model):
    ...
    second_email_address = models.EmailField(blank=True)
    photo = models.URLField(default='default-picture-url', blank=True)
    birth_date = models.DateField(null=True, blank=True)
```

Blank|Null BooleanField

- If a **`BooleanField`** is set to **allow empty values**, it changes from a checkbox to a select box





Primary Key Option

- `primary_key`
  - If `True`, the field becomes the primary key for the model
  - Used to **override** the default primary-key behavior
- The primary key field is **read-only**
- **Note:** If you change the value of the primary key on an existing object and then save it, a **new object** will be created alongside the old one

Choices Option

- **choices**
  - Use a **sequence** consisting of **iterables** of **exactly two items** to create choices
  - A new migration is **automatically created** each time the list of choices changes

```
MONTHS = [
    ('Jan', 'January'),
    ('Feb', 'February'),
    ('Mar', 'March'),
    ...
]
```

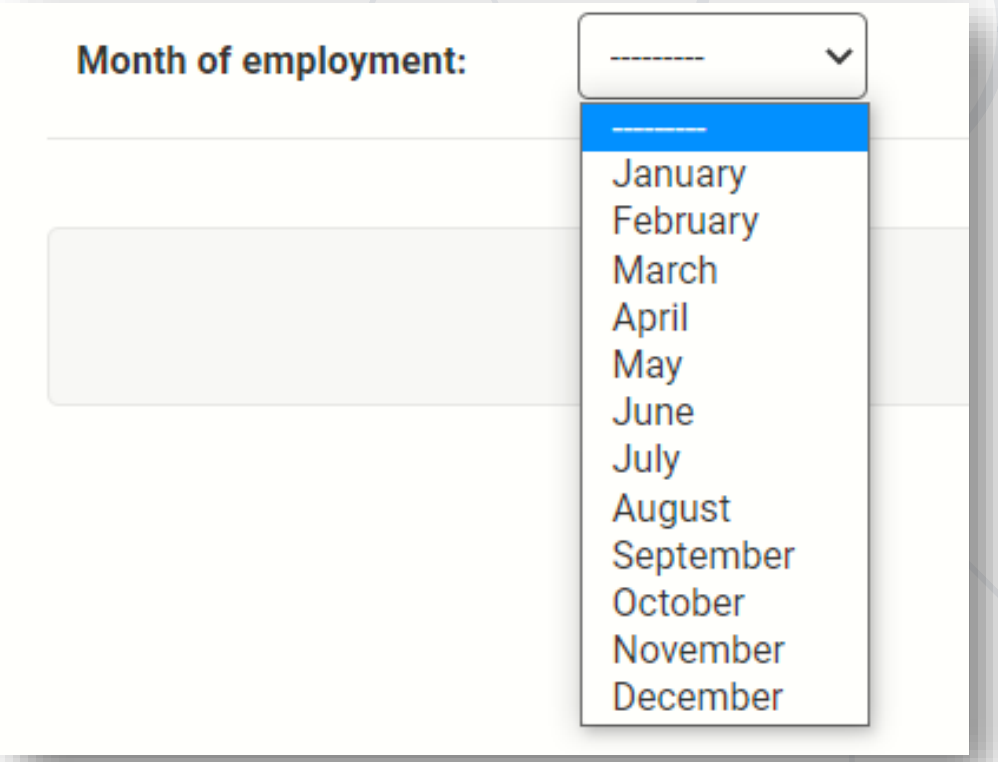
value to be set on the model

human-readable name

Choices Option

- It appears as a **select box** with the created choices instead of a standard text field

```
class Employee(models.Model):
    ...
    month_of_employment = \
        models.CharField(
            max_length=3,
            choices=MONTHS)
```



Verbose Name Option

- **verbose_name**
  - Most field types take it as an optional **first positional** argument
  - If it isn't given, Django **automatically** creates it using the **field's attribute name**, converting **underscores to spaces**

```
class Employee(models.Model):
    first_name = models.CharField(
        "First Name", max_length=30)
    last_name = models.CharField(
        "Family Name", max_length=40)
    email_address = models.EmailField(
        unique=True)
```

"First Name"

"Family Name"

"Email address"

Editable Option

- `editable`
  - `True` by default
  - If `False`, it modifies the field so:
    - It is **not able to be filled/ edited**
    - It **disappears** from all forms

```
class Employee(models.Model):
    ...
    email_address = models.EmailField(editable=False)
```

- Used to hide some fields such as encrypted code, verifications, etc.

Problem: Department Model

- Create a model called "**`Department`**" with the following field: **`code`**, **`name`**, **`employees_count`**, **`location`**, **`last_edited_on`**
- A full description of the problem can be found in the Lab document **here**

Solution: Department Model

```
CITIES = (("Sofia", "Sofia"),
          ("Plovdiv", "Plovdiv"),
          ("Burgas", "Burgas"),
          ("Varna", "Varna"))

class Department(models.Model):
    code = models.CharField(max_length=4, primary_key=True,
                            unique=True)
    name = models.CharField(max_length=50, unique=True)
    employees_count = models.IntegerField(default=1,
                                          verbose_name='Employees Count')
    location = models.CharField(max_length=20, choices=CITIES,
                                null=True, blank=True)
    last_edited_on = models.DateTimeField(auto_now=True, editable=False)
```

Models Migration Basics

How Models Turn into DB Tables

- Use models to create a **database schema** for your app
- Use **migrations** to **propagate changes** you make in your **models** (add, delete, modify fields, etc.)
  - First, **create migrations**
    - `makemigrations` command
  - Next, **apply those changes** to the database
    - `migrate` command

Migrations

- Use to **add changes** made to the models into the database
- Django **creates migrations** for you
  - Just type the appropriate **commands in the terminal**
- You can use many database systems with Django
  - However, **PostgreSQL** is the **most capable** of all in terms of schema support

Migration Commands

- **Creating** new migrations
  - Pack the changes into migration files

```
python manage.py makemigrations
```

- **Applying** the created migrations to the database
  - Use after the migration files are created

```
python manage.py migrate
```

Problem: Migrate the Models

- **Migrate** the created models named **`"Employee"`** and **`"Department"`** to the database
- **Check** the created database tables using **`dbshell`**
- **Submit** your project to the Judge system

Solution: Migrate the Models

- Open the terminal and run the command

```
python manage.py makemigrations
```

- Then, apply the migrations to the database

```
python manage.py migrate
```

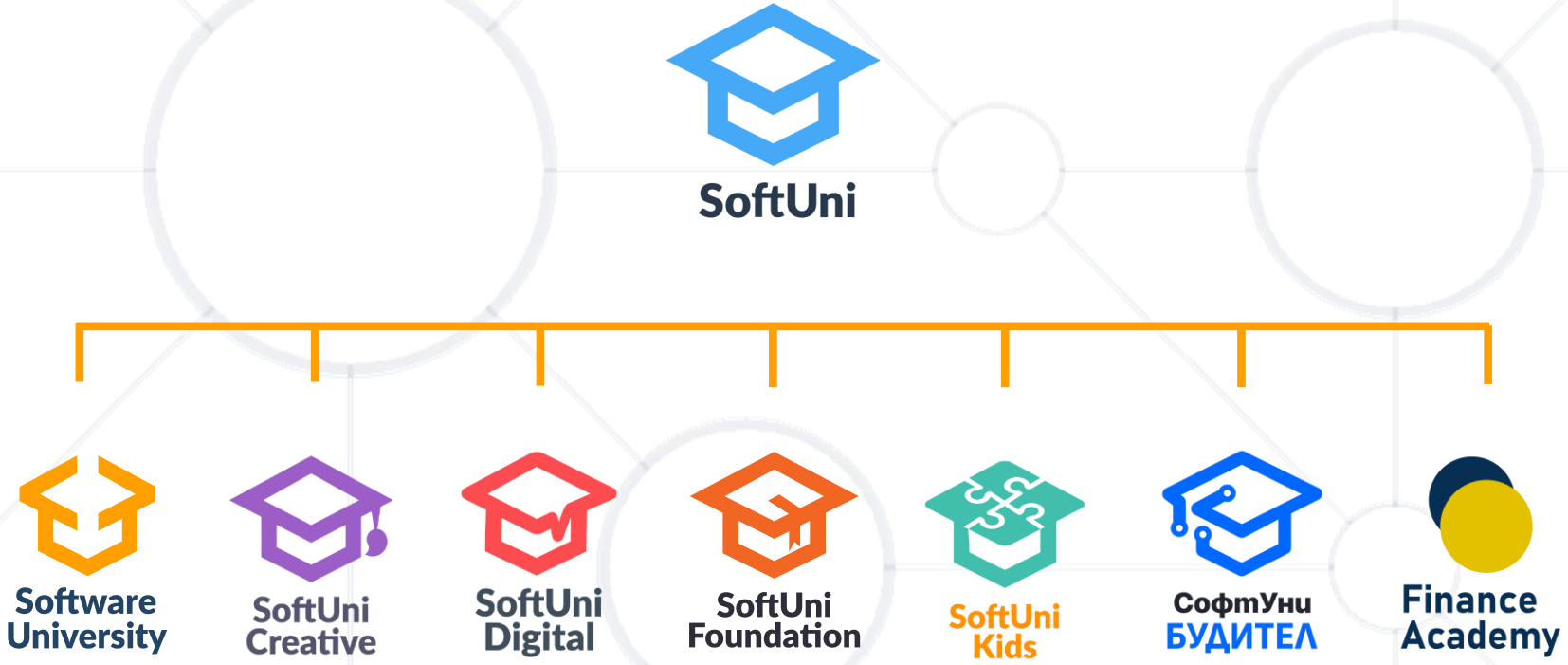
Problem: Project Model

- Create a model called "**`Project`**" with the following field: **`name`**, **`description`**, **`budget`**, **`duration_in_days`**, **`estimated_hours`**, **`start_date`**, **`created_on`**, **`last_edited_on`**
- **Migrate** the created model
- **Submit** your project to the Judge system
- A full description of the problem can be found in the Lab document **here**

Summary

- **Models** allow us to work with data using Python code
- We could specify DB column constraints using model **field options**
- Django **automatically** generates **migration** files and **SQL queries**

Questions?



SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank
Решения за твоето утре

DRAFT KINGS

VIVACOM

AMBITIONED

INDEAVR
Serving the high achievers

SOFTWARE GROUP

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos, and other assets) is **copyrighted content**
- Unauthorized copy, reproduction, or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg