Django

Getting Python in the web

Outline

- Introduction
- Project vs. App structure in Django
- Views and Templates
- Models and Databases
- Forms and User Input
- Admin
- Authentication and User Management
- Advanced Concepts

Models

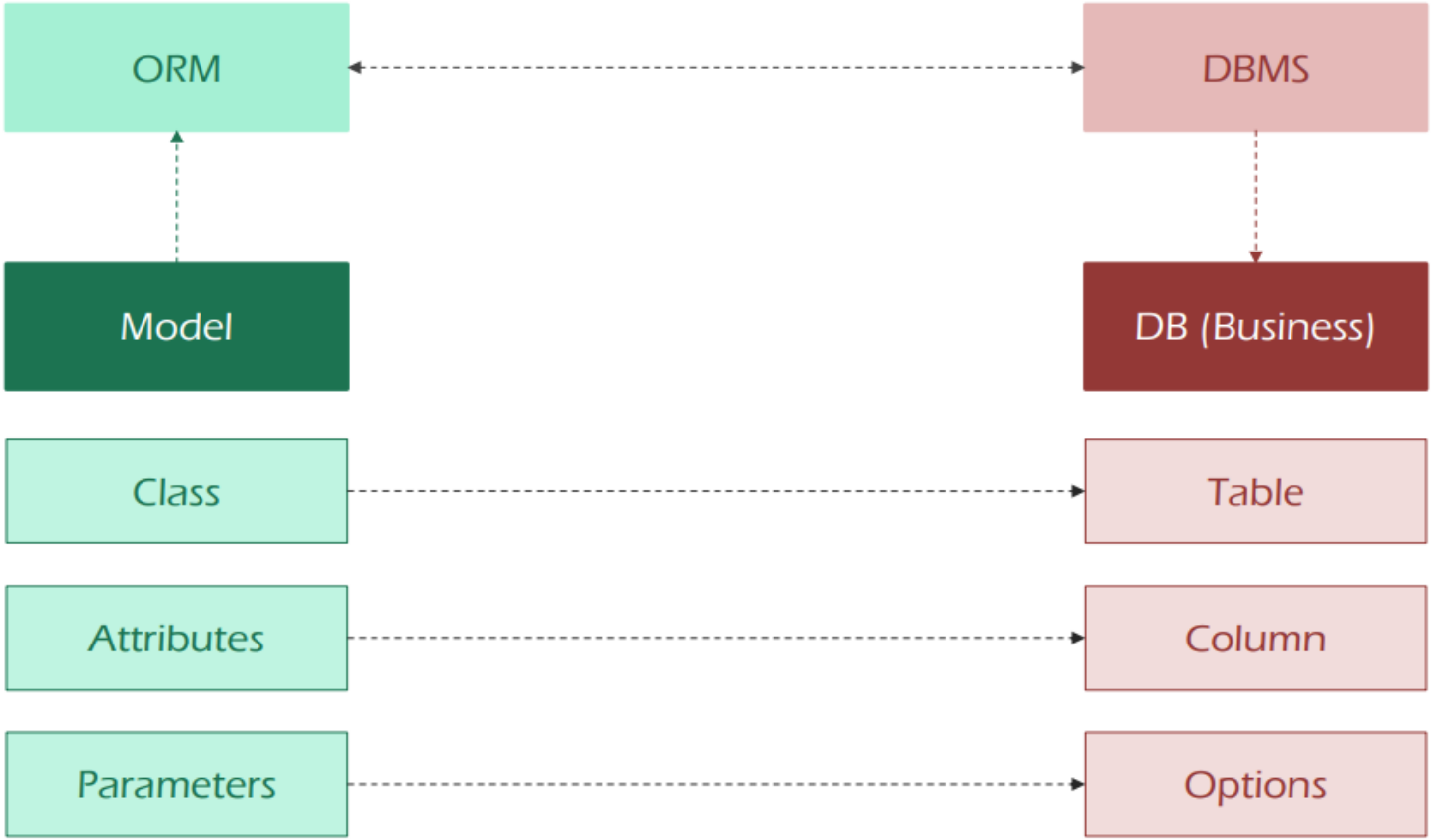
model

Python classes that define the structure and behavior of your application's data.

They represent database tables and provide a convenient

way to interact with the underlying database

Mapping


ORM
DBMS
Model
DB (Business)
Class
Table
Attributes
Column
Parameters
Options

Key concepts and features of Django models:

- **Fields**: Django provides various field types (e.g., CharField, TextField, DateTimeField, etc.) to define the data types and constraints for your model's fields. These fields map to corresponding database columns.

- **Relationships**: You can define relationships between models using fields like ForeignKey, OneToOneField, or ManyToManyField. These fields establish relationships

Key concepts and features of Django models:

- Model **Methods**: You can define methods within your models to perform operations or calculations specific to the model. In the example above, the __str__ method is defined in the Author model to represent the model instance as a string (useful for display purposes).

- Model **Meta**: The Meta class allows you to specify additional metadata for a model, such as the ordering of query results or the default ordering for the model's objects.

- Database **Migrations**: Django provides a built-in migration system to handle changes to your models over time. Migrations allow you to synchronize your database schema with the changes made to your models without losing existing data.

Settings

- Install the mysqlclient package, which is the MySQL database connector for Python. Open your terminal or command prompt and run the following command:
    - pip install mysqlclient
- Or for the psycopg2 package, which is the PostgreSQL database connector for Python. Open your terminal or command prompt and run the following command:
    - pip install psycopg2

Settings-MYSQL

- In your Django project's settings file settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'your_database_name',
        'USER': 'your_mysql_username',
        'PASSWORD': 'your_mysql_password',
        'HOST': 'localhost',  # Or your MySQL server's host IP address
        'PORT': '3306',  # Or your MySQL server's port
    }
}
```

Settings-PostgreSQL

- In your Django project's settings file settings.py

```
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.postgresql',
      'NAME': 'your_database_name',
      'USER': 'your_mysql_username',
      'PASSWORD': 'your_mysql_password',
      'HOST': 'localhost',  # Or your MySQL server's host IP address
      'PORT': '5432',  # Or your MySQL server's port
   }
}
```

Migrate commands

- **makemigrations** :create new database migration files based on the changes made to your models, **does not** apply the changes to the database immediately. It only **generates** the migration files.
    - python manage.py makemigrations **your_app_label1 your_app_label2**
- **sqlmigrate display** the SQL statements for a specific migration **without** applying the migration to the database
    - python manage.py sqlmigrate **your_app_label migration_name**
- **migrate** **apply** database migrations and update the database schema based on the migration files.
    - python manage.py migrate

Field Types

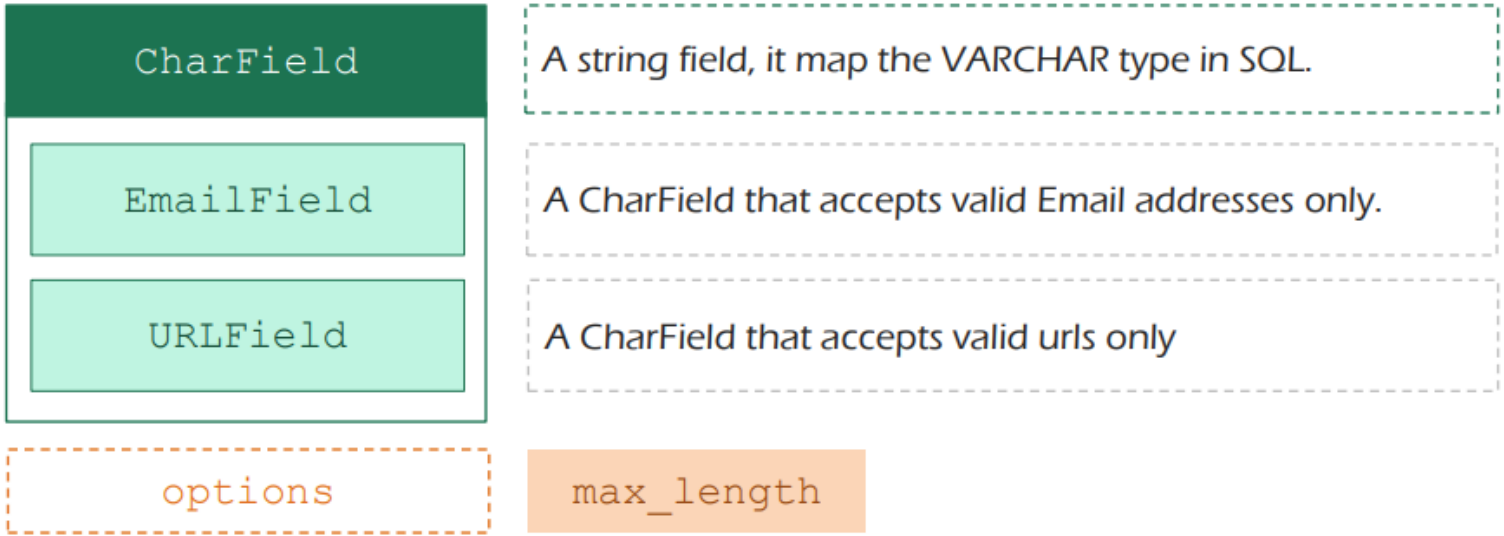
Field base class for all built-in field types.
provides common behavior and attributes that are shared among different field types in Django.

Field

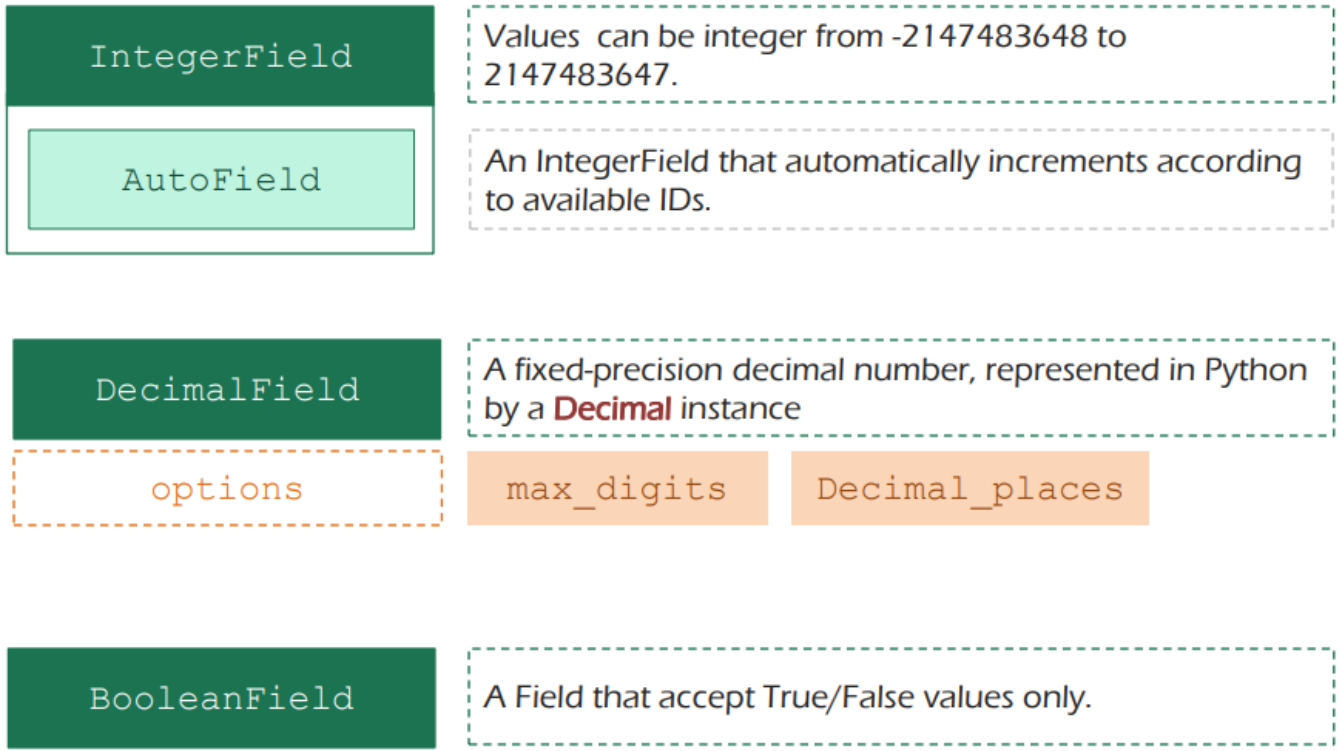
options

null	If True , It is allowed fro the Field to be null.
choices	A Tuple of choices that Field value can be.
db_column	The name of the database column to use for this field
default	The default value for field if not given
primary_key	If True, Field will be the primary key of the table in DB.
unique	If True, Filed values must be unique.

Character & Text



Numeric & Boolean



Date & Time


DateField
A date, represented in Python by a datetime.date instance
TimeField
A date, represented in Python by a datetime.time instance
DateTimeField
A date, represented in Python by a datetime.datetime instance
options
auto_now
auto_now_add

Relationship

Relationship Fields

- Establish relationships **between** different **models**. They allow you to define how one model is related to another

- **ForeignKey:** Represents a one-to-many relationship where each instance of the model with the ForeignKey field belongs to a single instance of another model

- **OneToOneField**: Represents a one-to-one relationship where each instance of the model with the OneToOneField field is associated with a single instance of another model. Options are similar to ForeignKey.

- **ManyToManyField :** Represents a many-to-many relationship where each instance of the model with the ManyToManyField field can be associated with multiple instances of another model. A separate intermediary table is created to manage the relationship. I

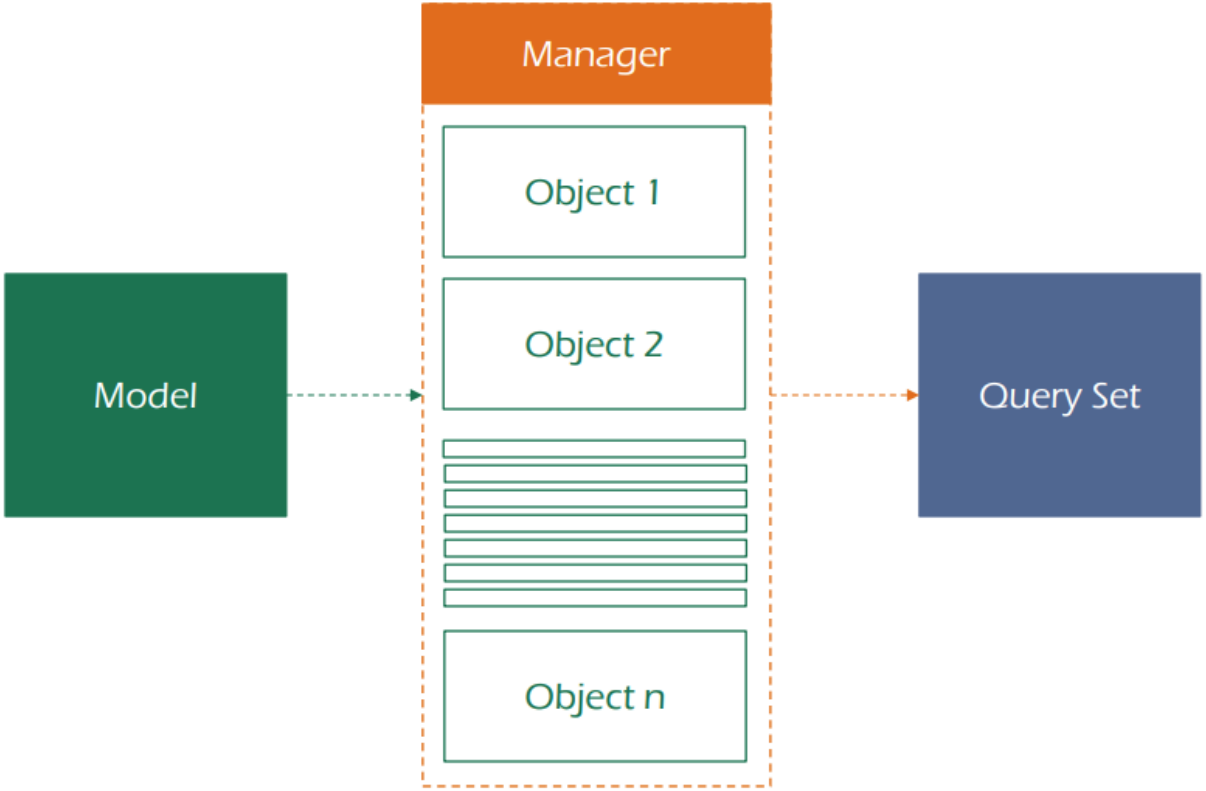
Example

```
from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()

    def __str__(self):
        return self.name

class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
    publication_date = models.DateField()

    def __str__(self):
        return self.title
```

Example

```
class Publisher(models.Model):
    name = models.CharField(max_length=100)
    books = models.ManyToManyField(Book)

    def __str__(self):
        return self.name
```

Model Operations




```
Post . objects . all()
```

Create

```
# Create an author
author = Author.objects.create(name='John Smith', email='john@example.com')

# Create a book and associate it with the author
book = Book.objects.create(title='Sample Book', author=author,
publication_date='2022-01-01')

# Access the author of a book
book_author = book.author
```

Retrieve Objects

SELECT … WHERE

```
User.objects.create(first_name='Ahmed', last_name='Moawad')

User.objects.create(first_name='Mohamed', last_name='Saeed')

User.objects.create(first_name='Omar', last_name='Saeed', age=12)

User.objects.all()

<QuerySet [<User: User object>, <User: User object>],<User: User object>]>

# To filter the resulting QuerySet based on condition, use filter

User.objects.filter(last_name='Saeed')

<QuerySet [<User: User object>, <User: User object>] >

# To Retrieve single record, use get

User.objects.get(age = 12)

<User: User object>
```

Update Objects

```
User.objects.create(first_name='Ahmed', last_name='Moawad')

User.objects.create(first_name='Mohamed', last_name='Saeed')

User.objects.create(first_name='Omar', last_name='Saeed', age=12)

User.objects.filter(last_name='Saeed').update(age = 19)
```

Delete Objects

```
User.objects.create(first_name='Ahmed', last_name='Moawad')

User.objects.create(first_name='Mohamed', last_name='Saeed')

User.objects.create(first_name='Omar', last_name='Saeed', age=12)

User.objects.filter(last_name='Saeed').delete()
```

Lookups

```
User.objects.create(first_name='Ahmed', last_name='Moawad')          #1
User.objects.create(first_name='Mohamed', last_name='Saeed')         #2
User.objects.create(first_name='Omar', last_name='saeed', age=12)    #3
User.objects.filter(first_name__in =['Ahmed', 'Omar'])             [#1,#3]
User.objects.filter(last_name__iexact ='saeed')                    [#2,#3]
User.objects.filter(age__gt =13)                                   [#1,#2]
```

Aggregate

```
from django.db.models import Avg

User.objects.create(first_name='Ahmed', last_name='Moawad')          #1

User.objects.create(first_name='Mohamed', last_name='Saeed')        #2

User.objects.create(first_name='Omar', last_name='saeed', age=12)   #3

User.objects.count() #3

User.objects.all().aggregate(Avg('age'))

{'age__avg': 14}
```

Raw SQL

```
User.objects.raw('SELECT * FROM users_user')
```

Admin

Fully Implemented Admin Panel , easily be integrated with your project business

Create Super User

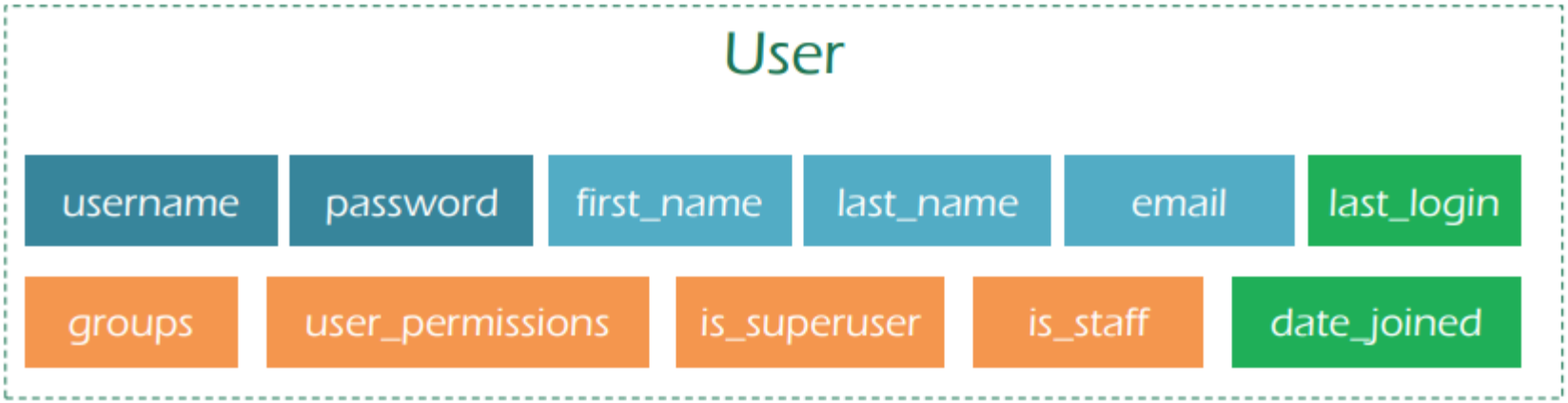
```
$ cd mysite
$ python manage.py createsuperuser
```

Register a model to Admin Panel

library/admin.py

```
from django.contrib import admin


from .models import Book


admin.site.register(Book)
```

Authentication




```
from django.contrib.auth.models import User

user = User.objects.create_user('Ahmed','am@gmail.com','120829')
```

User Authentication Example

```
from django.shortcuts import render, redirect
from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth import login, logout

def login_view(request):
    if request.method == 'POST':
        form = AuthenticationForm(request, data=request.POST)
        if form.is_valid():
            user = form.get_user()
            login(request, user)
            return redirect('home')
    else:
        form = AuthenticationForm()
    return render(request, 'login.html', {'form': form})
```

User Authentication Example

```
def any_view(request):
 if request.user.is_authenticated():
 # Do actions for Logged in Users
 else:
# Do actions for Guests


def logout_view(request):
    logout(request)
    return redirect('login')
```

Lab

template inhertance