# Q.1>Why Django? What are the main advantages?

Django is an open-source web development framework supported by the Django Software Foundation.

It’s written in Python,

is designed to encourage “rapid development

Big giantess Google, Youtube, and Instagram use this framework.

Such large websites built on Django ,Pinterest, Disqus, Mozilla, National Geographic.

**framework**

framework is a predefined structure and set of rules that provide the foundational for developing a web application.

And offering standardized way to bulid and organize code . for developing web app .

It serves as a foundation, so you’re not starting entirely from scratch.

# Q.2>What is the official website of Django?

https://www.djangoproject.com/

# Q.3>Who developed Django?

Django was invented by Lawrence Journal-World in 2003,

# Q.4>Is Django Open Source?

yes,it is

# Q.6>Which is the latest version of Django?

4.2.1

# Q.7>How to install Django?

Globally:

pip install django

Virtually:

py -m venv myworld

in a specific location

D:\Users1\cybrom>virtualenv -p python3 venv

D:\Users1\cybrom>venv\Scripts\activate

(venv) D:\Users1\cybrom>pip install django

# Q.8>PIP full form & why it is used?

pip is the package installer for Python. You can use it to install packages from the Python Package Index and other indexes.

“pip Install Packages”

# Q.9>How to check the version of Django in cmd?

django-admin --version

# Q.10>What is Django admin?

django-admin is Django’s command-line utility for administrative tasks.

# Q.11>How to Check if Django is installed on the system or not?

django-admin --version

# Q.12>How to verify that Django can be seen by Python?

python -m django --version

# Q.13>How MVT work?

Django, a Python framework to create web applications,

is based on Model-View-Template (MVT) architecture.

MVT is a software design pattern for developing a web application.

It consists of the following three entities:

Model

View

Template

**Model –**

1.a model is a object.

2. in model define the structure of data in Django app

3.model is responsible for maintaining the entire application and which it provides various mechanisms to add, update, read and delete the data

in the database.

**View**

A View is a handler function that accepts HTTP requests, processes them, and

returns the HTTP response.

**Template**

A Template is a text file that defines the structure or layout of the user interface.

The text file can be any type of file; for example HTML, XML, etc.

It can accept data from the view and render it using jinja syntax.

# Q.14>Which server is best suited to implement you want to use Django on a production site?

For Django this will usually be the Gunicorn web application server (with a . wsgi script).

Gunicorn (‘Green Unicorn’) is a pure-Python WSGI server for UNIX. It has no dependencies

and can be installed using pip.

# Q.15>What is the role of manage.py?

In a Django project, the **manage.py** file serves as a command-line utility that allows you to interact with various aspects of your project.

You can use it to perform tasks such as running the development server, creating database migrations, managing database schema, and executing custom management commands. It acts as an entry point for many administrative and development tasks within a Django project, simplifying the process of managing and running your application.

# Q.16> Queryset?

A queryset is a list of objects from the Models.

Queryset make easier to work with data like get , filter ,orderby tasks

<QuerySet [  
  <Member: Member object (1)>,  
  <Member: Member object (2)>,  
  <Member: Member object (3)>,  
  <Member: Member object (4)>,  
  <Member: Member object (5)>  
]>

As you can see, our Member model contains 5 records, and are listed inside the QuerySet as 5 objects.

# Q.16>What is an object-relational mapper?

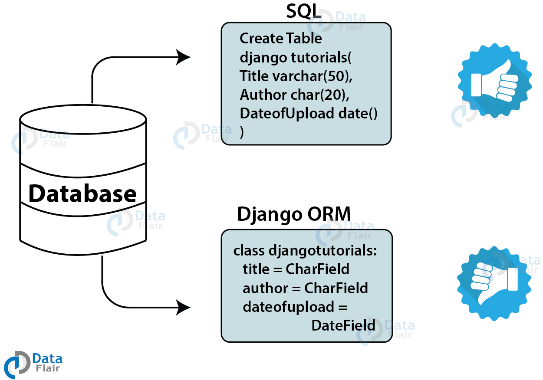
ORM –

ORMs create a database schema from defined classes/ models.

They generate SQL from Python code for a database YOU DON’T need to write sql .

ORMs let the developer build the project in one language that means Python.

table(row)==class(object)



syntax:

ModelName.objects.method\_name(arguments)

ex:

Student.objects.get(pk=id)

Student.objects.filter(name='ajay')

# Q.17>Name default installed app in Django?

rename in

settings.py

wsgi.py

asgi.py

manage.py

# Q.18>How can we see the SQL of a particular migration?

from django.db import connection

print(connection.queries)

and

py manage.py sqlmigrate app\_name migation\_file\_name

# Q.19>How to change the server's port?

py manage.py runserver 7000

# Q.20>Settings.py in used for?

settings.py is a core file in Django projects.

The **settings.py** file containing various settings that define how the Django application behaves.

It includes database configurations, security settings, middleware, static and media file paths, internationalization settings, and more.

This file allows developers to customize the behavior of their Django application without modifying the core framework files.

# Q.21>Where we can declare the urls of Django projects?

1)default\_app.urls

2)app.urls

# Q.22>What is the use of the asgi.py & wsgi.py?

WSGI (Web Server gate Interface)

WSGI is synchronous, which means it handles one request at a time.

When a request comes in, it’s processed by the web server and the WSGI application sequentially.

This makes it suitable for applications with low to moderate traffic.

ASGI (Asynchronous Server Gateway Interface)

ASGI is an asynchronous protocol, which means that it allows parallel processing of multiple requests.

It handles real-time, long-polling and Websocket connections efficiently.

ASGI is newer than WSGI.

ASGI is ideal for complex and high traffic applications.

It is ideal for application processing in real time like chat applications, streaming services and live notifications.

# Q.23>What is the use of the \_\_init\_\_.py?( \_\_init\_\_(self): constructor)

The \_\_init\_\_.py file is used to mark a directory as a Python package ,

making it possible to import modules and sub-packages from that directory .

# Q.24>What is \_\_pycache\_\_?

When you run a program in Python, the interpreter compiles it to bytecode first and stores it in the \_\_pycache\_\_ folder.

If you look in there you will find a bunch of files sharing the names of the .py files in your project's folder,

only their extensions will be either .pyc or .pyo. These are bytecode-compiled and optimized bytecode-compiled versions of your program's files, respectively.

As a programmer, you can largely just ignore it... All it does is make your program start a little faster.

When your scripts change, they will be recompiled,

and if you delete the files or the whole folder and run your program again, they will reappear (unless you specifically suppress that behavior).

\_\_pycache\_\_ is a directory automatically created by Python to store compiled bytecode files (.pyc) generated by the Python interpreter. These bytecode files are created to improve the performance of subsequent imports of Python modules by caching the compiled version of the source code.

When a Python module is imported, the interpreter checks if a corresponding bytecode file exists in the \_\_pycache\_\_ directory. If the bytecode file is found and is up-to-date, Python will load and execute it instead of recompiling the source code. This process can significantly reduce the startup time of Python programs and improve overall performance.

# Q.25>When you create a Django project, which files do you get in advance?

manage.py

[project\_name]/

init.py

settings.py

urls.py

wsgi.py

asgi.py

pycache/ (directory)

# Q.27>What must be the debug value while the developments of a python project?

True

# Q.28> Where we can declare the urls of Django projects?

urls.py

# Q.29>Can we delete the default app?

yes, we can but it's troublesome b'se we have to create settings,asgi,etc by ourself ,which create a problem.

# Makemigrations and migrate

makemigrations is used to create migration files based on changes to your Django models, while migrate is used to apply those migration files and synchronize the database schema with your models.

It's a two-step process: first, you generate the migration files with makemigrations, then you apply those migration files with migrate

In summary, **makemigrations** prepares migration files based on changes in your models, while **migrate** applies those changes to your database schema.

# One to one relationship

In Django, a one-to-one relationship is a type of relationship between two models where each record in one model is associated with exactly one record in another model, and vice versa.

This relationship is commonly used when you want to establish a unique, one-to-one correspondence between instances of two different models.

Let's explain this with an example of a Person model and a Profile model, where each person has exactly one profile:

# models.py

from django.db import models

class Person(models.Model):

name = models.CharField(max\_length=100)

age = models.IntegerField()

def \_\_str\_\_(self):

return self.name

class Profile(models.Model):

person = models.OneToOneField(Person, on\_delete=models.CASCADE)

bio = models.TextField()

location = models.CharField(max\_length=100)

def \_\_str\_\_(self):

return f"Profile of {self.person.name}"

Explanation of the code:

We define a Person model with fields name and age.

We define a Profile model with fields bio and location.

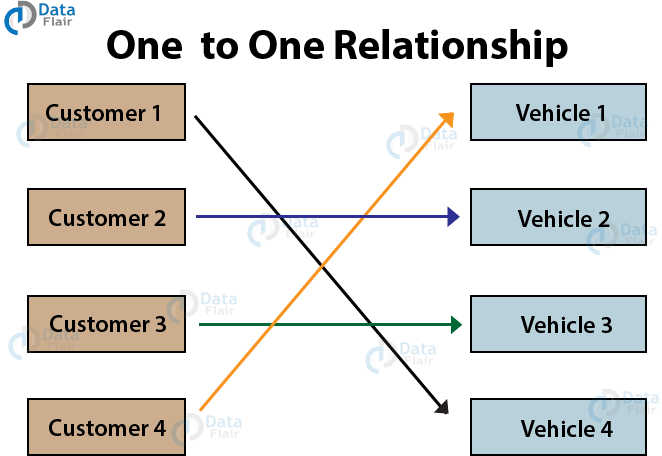
In the Profile model, we establish a one-to-one relationship with the Person model using OneToOneField.

This means that each Profile instance is associated with exactly one Person instance.

We specify on\_delete=models.CASCADE to ensure that if a Person instance is deleted, its associated Profile instance is also deleted. This maintains referential integrity in the database.

In the Profile model's \_\_str\_\_ method, we return a string representation that includes the name of the associated Person.

Now, you can create instances of Person and Profile and establish one-to-one relationships between them:



# One to Many Relationships

A one to many relationships is where one object from table1 can have multiple relations with entities in table2. Although, table2 objects will have only one relation to the object of table1.

In Django, a one-to-many relationship is a type of relationship where one instance of a model can be associated with multiple instances of another model. This is achieved by using a ForeignKey field, where the "one" side of the relationship holds the ForeignKey, and the "many" side of the relationship doesn't.

# models.py

from django.db import models

class Author(models.Model):

name = models.CharField(max\_length=100)

# Other author fields...

def \_\_str\_\_(self):

return self.name

class Book(models.Model):

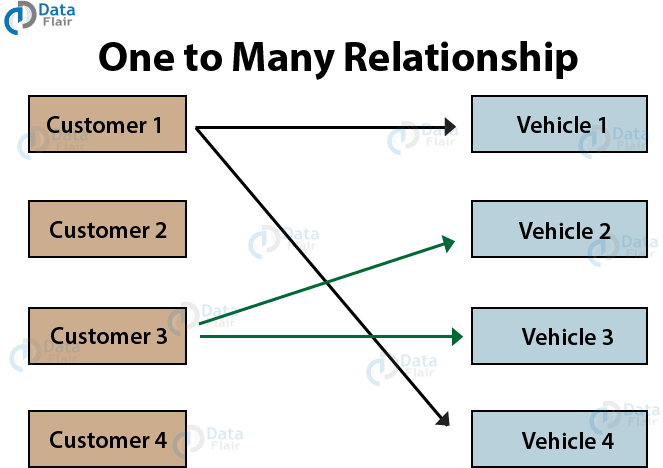
title = models.CharField(max\_length=100)

author = models.ForeignKey(Author, on\_delete=models.CASCADE)

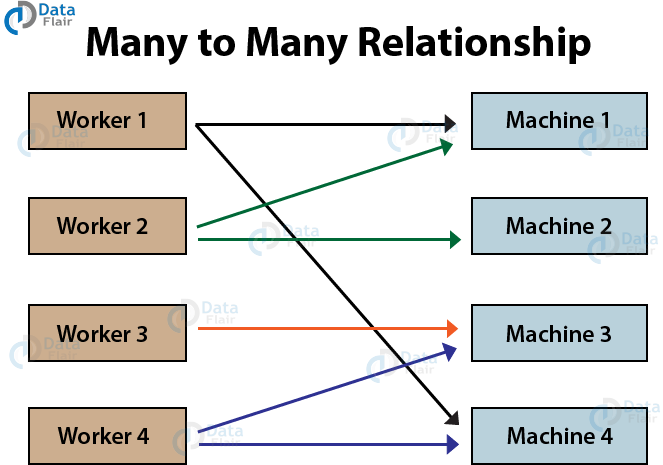
# Other book fields...

def \_\_str\_\_(self):

return self.title

[](https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/09/one-to-many-relationship.png)

# Many to Many Relationships

[](https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/09/many-to-many-relationship.png)

Here we have multiple workers for multiple machines. A worker can be assigned to operate more than one machine. Also, a machine can be operated by multiple workers one at a time.

In Django, a Many-to-Many relationship is a type of relationship between two models where each instance of one model can be associated with multiple instances of another model, and vice versa. This is commonly used when you have a situation where one entity can be related to multiple instances of another entity, and vice versa.

from django.db import models

class Author(models.Model):

    name = models.CharField(max\_length=100)

class Book(models.Model):

    title = models.CharField(max\_length=100)

    authors = models.ManyToManyField(Author)

# many to one relationship

many-to-one relationship is a type of database relationship where multiple records from one model are associated with a single record from another model. This is typically implemented using ForeignKey fields.

# ER diagram

An Entity-Relationship (ER) diagram is a visual representation of the entities (objects or concepts), their attributes, and the relationships between them within a system or a database. It's a fundamental tool used in database design to model the structure and organization of data.

ER diagrams serve as a visual tool for database designers, developers, and stakeholders to understand the structure of a database schema, including its entities, attributes, and relationships. They help in the process of conceptualizing, designing, and implementing databases, as well as in communicating database designs to others involved in the development process.

1. Install Graphviz software in your laptop

2. pip install graphviz

3. pip install django-extensions

4. Go to settings.py > add 'django\_extensions', to INSTALLED\_APPS

5. Add the following code

GRAPH\_MODELS ={

'all\_applications': True,

'graph\_models': True,

}

to settings.py

6. pip install pyparsing pydot

7. py manage.py graph\_models -a > erd.dot

8.py manage.py graph\_models -a -g -o erd.png

If code give error

Do this

conda install pydot-ng

And finally in my notebook I added the two lines below.

Example:

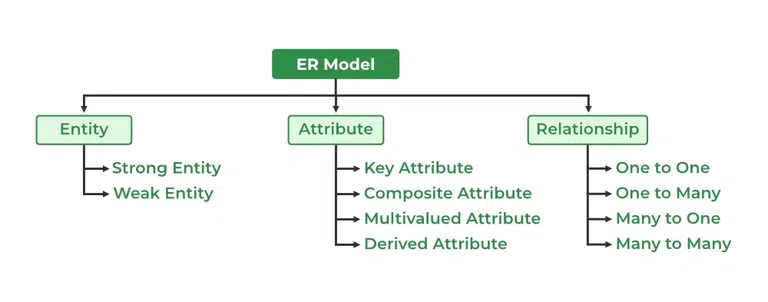
import os

os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'

add to setting.py

import os

os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin/'



# API (Application Programming Interface):

An API is a set of rules, protocols, and tools that allows different software applications to communicate with each other.

types of API

=============

public

private

web

# Web API (Web Application Programming Interface):

A Web API is an API that is accessed over the web using HTTP protocols.

It enables interaction between different web-based software systems, allowing them to exchange data and functionality.

Web APIs are commonly used for integration between web services and applications.

# REST API (Representational State Transfer API):

REST API is a type of Web API that follows the principles of REST architectural style.

It uses standard HTTP methods like GET, POST, PUT, DELETE to perform CRUD (Create, Read, Update, Delete) operations on resources.

REST APIs are stateless, meaning each request from a client contains all the information needed to process the request, and they typically return data in formats like JSON or XML.

# Serializer

A serializer converting complex data types (such as Django model instances) into native Python datatypes (like dictionaries or lists) that can be easily rendered into JSON, XML, or other content types.

\* client GET or POST

GET

serializer

complex data ------------>python native datatype------------------->Json

Student.objects.all() python dict jsonRenderer

# **Deserializer**:

A deserializer, on the other hand, is responsible for converting the native Python datatypes received from client requests (such as JSON data) into complex data types (such as Django model instances) that can be processed by the application.

Deserializers handle the validation and conversion of incoming data, ensuring that it meets the expected format and constraints before saving it to the database or performing further operations.

POST

===

deserializer

json ------------------>python native datatype------------------->complex datatype

parserer python dict

Serializers allow complex data such as querysets and model instances to be

converted to native Python datatypes that can then be easily rendered into JSON,

XML, or other content types.

Serializers also provide deserialization, allowing parsed

data to be converted back into complex types, after first validating the incoming data.