# Query set Options in Django –

**objects.all():** This method will return all the records in the database.

**filter()**: This method allows you to filter the queryset based on one or more conditions. For example, you can use **filter(name='John')** to retrieve all objects with a **name** field equal to 'John'.

A screenshot of a computer

Description automatically generated with medium confidence

**exclude()**: This method allows you to exclude objects from the queryset based on one or more conditions. For example, you can use **exclude(age\_\_lt=18)** to exclude all objects where the **age** field is less than 18.

A screenshot of a computer

Description automatically generated with medium confidence

**get()**: This method allows you to retrieve a single object from the queryset based on one or more conditions. For example, you can use **get(id=1)** to retrieve the object with an **id** field equal to 1.

A screenshot of a computer

Description automatically generated with medium confidence

**order\_by()**: This method allows you to sort the queryset based on one or more fields. For example, you can use **order\_by('name')** to sort the queryset in ascending order based on the **name** field.

Text

Description automatically generated

**values()**: This method allows you to retrieve a queryset of dictionaries containing only the specified fields. For example, you can use **values('name', 'age')** to retrieve a queryset of dictionaries containing only the **name** and **age** fields.

Text

Description automatically generated

**annotate()**: This method allows you to annotate each object in the queryset with an additional field based on an aggregation or calculation. For example, you can use **annotate(num\_friends=Count('friends'))** to annotate each object with a **num\_friends** field that represents the number of friends that object has.

Text

Description automatically generated

# Different methods to create views

In Django, there are several ways to create views for your web application, some of which include:

1. **Function-based views:** These views are created as functions that take a request object as input and return an HTTP response object. Function-based views are simple and easy to use, and can be written quickly for smaller applications.
2. **Class-based views**: These views are created as classes that inherit from Django's built-in **View** class or one of its subclasses (such as **TemplateView**, **ListView**, or **DetailView**). Class-based views provide more functionality and flexibility than function-based views, and can be easier to organize and maintain for larger applications.
3. **Generic views**: These are built-in views that Django provides to make common use cases easier to implement. Examples of generic views include **ListView** (for displaying a list of objects), **DetailView** (for displaying a single object), **CreateView** (for creating new objects), and **UpdateView** (for updating existing objects).
4. **Viewsets**: Viewsets are similar to class-based views, but they provide a more structured way of defining views for creating, reading, updating, and deleting objects in a Django application. They are often used in conjunction with Django REST framework to create RESTful APIs.
5. **API views**: These views are used specifically for creating APIs in Django, and are usually created using Django REST framework. They provide a structured way of defining views that return JSON or other data formats.

These are just a few of the methods that you can use to create views in Django. The choice of which method to use depends on your specific use case and requirements.