Cheat Sheet: Django Application Development with SQL and Databases

Estimated reading time: 12 minutes

Package/Method	Description	Code Example
all()	Retrieves all instances of the 'MyModel' model from the database.	`MyModel.objects.all()`
AVG	Calculates the average value of a column.	`SELECT AVG(column1) FROM table_name;`
Avg()	Calculates the average of a field.	`MyModel.objects.aggregate(Avg('field'))`
Basic View Function	Function-based view that returns "Hello, World!" From Django.http import HttpResponse	`def my_view(request):` `# Your view logic here` `return HttpResponse("Hello, World!")`
Bootstrap classes and components	Create visually appealing and responsive web pages without having to write CSS styles manually.	`<a href="#" class="btn btn-primary">Click Me</a>`
Bootstrap CSS	Link to include Bootstrap CSS in the base template.	`Add the following link to the <head> section of your base template (usually base.html):` `<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="st`
Bootstrap JavaScript	Script tag to include Bootstrap JavaScript library.	`Include the Bootstrap JavaScript library at the end of the <body> section to enable certain` `<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.min.js"></script`
Collecting static files	When deploying your project, you need to collect all static files into a single location.	`python manage.py collectstatic` `STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')`
Configuration – App Dirs	A configuration option used within the TEMPLATES setting. When set to TRUE, Django will look for template files within the app directories.	`Make sure the APP_DIRS setting is set to True in the TEMPLATES list. This allows Django to l` `TEMPLATES = [` `{` `# ...` `APP_DIRS': True,` `# ...` `},` `]`
Configuration – Installed apps	Defines a list of all the applications installed in the project.	`Add 'django.contrib.staticfiles' to your INSTALLED_APPS in settings.py:` `INSTALLED_APPS = [` `# ...` `django.contrib.staticfiles',` `# ...` `]`
Configuration – Static files	Django settings for static files configuration.	`In your Django settings (settings.py), define the following settings:` `STATIC_URL = 'https://prod-edx-edxapp-assets.edx-cdn.org/static/studio/edx.org-next/' # URL` `STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')] # Directory to look for static files`
contains	Checks if the value is a substring within the field.	`MyModel.objects.filter(field__contains="value")`
COUNT	Counts the number of rows or non-null values in a column.	`SELECT COUNT(*) FROM table_name; or SELECT COUNT(column1) FROM table_name;`
count()	Counts the number of objects.	`MyModel.objects.count()`
CreateView	Displays a form to create a new object.	`class MyCreateView(CreateView):` `model = MyModel` `template_name = 'my_template.html'` `fields = '__all__' # or specify a list of fields`
DELETE FROM	Deletes data from a table based on specified conditions.	`DELETE FROM table_name WHERE condition;`
delete()	Deletes an object.	`obj.delete()`
DeleteView	Displays a confirmation page to delete an object.	`class MyDeleteView(DeleteView):` `model = MyModel` `template_name = 'my_template.html'` `success_url = '/success-url/'` `pk_url_kwarg = 'my_model_id' # default: pk`
DetailView	Displays details of a single object.	`class MyDetailView(DetailView):` `model = MyModel` `template_name = 'my_template.html'` `context_object_name = 'object' # default: object` `pk_url_kwarg = 'my_model_id' # default: pk`
DISTINCT	Returns unique values from a column.	`SELECT DISTINCT column1 FROM table_name;`
django.db.models.Model	Define a model.	`from django.db import models` `class MyModel(models.Model):` `field1 = models.CharField(max_length=100)` `field2 = models.IntegerField()`
endswith	Determines whether a string ends with the specified suffix.	`MyModel.objects.filter(field__endswith="value")`
exact	Retrieves instances of the 'MyModel' model from the database where the value of the 'field' attribute is exactly equal to "value".	`MyModel.objects.filter(field__exact="value")`

Package/Method	Description	Code Example
field	Performs a filtering operation on the 'MyModel' model instances based on a related model's field value.	`MyModel.objects.filter(related_model__field="value")`

filter()	Filter objects using conditions.	`MyModel.objects.filter(field1="value")`
		`MyModel.objects.filter(field2__gt=5)`

filter(ForeignKey)	Performs conditional joins.	`MyModel.objects.filter(related_model__isnull=True)`
FROM	Specifies the table from which data is retrieved.	`SELECT column1, column2 FROM table_name;`
FULL JOIN	Returns all rows from both tables, regardless of the match.	`SELECT column1, column2 FROM table1 FULL JOIN table2 ON table1.column = table2.column;`
get()	Retrieves a single instance of the 'MyModel' model from the database where the value of 'field1' is "value".	`MyModel.objects.get(field1="value")`
GROUP BY	Groups rows based on a specified column.	`SELECT column1, COUNT(*) FROM table_name GROUP BY column1;`
gt	Checks if the value of 'field' is numerically greater than 5.	`MyModel.objects.filter(field__gt=5)`

Handle a Form Submission	Function-based view to handle form submission. From django.shortcuts import render	`def my_form_view(request):`
		`if request.method == 'POST':`
		`# Process the form data here`
		`else:`
		`# Display the form`
		`return render(request, 'my_form_template.html', context)`

Handle URL Parameters	Function-based view that accesses URL parameters.	`def my_param_view(request, param):`
		`# Access the 'param' value from the URL`

HAVING	Filters grouped data based on specified conditions.	`SELECT column1, COUNT(*) FROM table_name GROUP BY column1 HAVING COUNT(*) > 1;`
iexact	The iexact lookup is case-insensitive, meaning it will match values regardless of whether they are uppercase or lowercase and provide a case-insensitive match.	`MyModel.objects.filter(field__iexact="value")`
in	Checks if the value of the field is present in the given list of values.	`MyModel.objects.filter(field__in=["value1", "value2"])`
INNER JOIN	Returns only matching rows from both tables.	`SELECT column1, column2 FROM table1 INNER JOIN table2 ON table1.column = table2.column;`

INSERT INTO	Inserts data into a table.	`INSERT INTO table_name (column1, column2) VALUES (value1,`
		`value2);`

JOIN	Combines rows from multiple tables based on related columns.	`SELECT column1, column2 FROM table1 JOIN table2 ON`
		`table1.column = table2.column;`

LEFT JOIN	Returns all rows from the left table and matching rows from the right table.	`SELECT column1, column2 FROM table1 LEFT JOIN table2 ON`
		`table1.column = table2.column;`

ListView:	Displays a list of objects.	`class MyListView(ListView):`
		`model = MyModel`
		`template_name = 'my_template.html'`
		`context_object_name = 'object_list' # default:`
		`object_list`

lt	Checks if the value of ‘field’ is numerically less than 10.	`MyModel.objects.filter(field__lt=10)`

makemigrations/migrate	Create database tables based on models.	`python manage.py makemigrations`
		
		`python manage.py migrate`

many_to_many	Performs many-to-many join.	`obj.many_to_many_field.all()`
MAX	Finds the maximum value in a column.	`SELECT MAX(column1) FROM table_name;`
Max()	Provides the maximum value of a field.	`MyModel.objects.aggregate(Max('field'))`
MIN	Finds the minimum value in a column.	`SELECT MIN(column1) FROM table_name;`
Min()	Provides the minimum value of a field.	`MyModel.objects.aggregate(Min('field'))`

obj = MyModel(field1="value", field2=5)
 	
obj.save()
	Creates a new instance of the 'MyModel' model with the values "value" for 'field1' and 5 for 'field2', and then saves the instance to the database.	`obj = MyModel(field1="value", field2=5)`
		`obj.save()`

obj.field1 = "new value" obj.save()	Updates the value of 'field1' for the 'obj' instance to "new	`obj.field1 = "new value"`
		`obj.save()`

Package/Method	Description	Code Example
	value" and saves the changes to the database.	
obj.model_set.all()	Fetches all related objects associated with the 'obj' instance. Access related objects in reverse (ForeignKey)	`obj.model_set.all()`
obj.related_model	Retrieves the related model associated with the 'obj' instance. Access related objects (Foreign Key or OneToOneField)	`obj.related_model`
ORDER BY	Sorts the result set based on specified columns in ascending or descending order.	`SELECT column1, column2 FROM table_name ORDER BY column1 ASC;`
order_by()	Orders objects based on a field.	`MyModel.objects.order_by('field')`
order_by(-)	Order objects based on fields in descending order.	`MyModel.objects.order_by('-field')`
prefetch_related	Performs left Outer join.	`MyModel.objects.prefetch_related('related_model')`

Protecting Views (Restrict Access) using @login_required Decorator	Function-based view protected with login_required decorator. From django.contrib.auth.decorators import login_required	`@login_required`
		`def my_protected_view(request):`
		`# Your view logic here`

Redirect to a URL	Function-based view to redirect to a specific URL. From django.shortcuts import redirect	`def my_redirect_view(request):`
		`return redirect('url_name_or_path')`

Render a Template	Function-based view to render a template with context. From django.shortcuts import render	`def my_template_view(request):`
		`context = {'variable': value}`
		`return render(request, 'my_template.html', context)`

RIGHT JOIN	Returns all rows from the right table and matching rows from the left table.	`SELECT column1, column2 FROM table1 RIGHT JOIN table2 ON table1.column = table2.column;`
SELECT	Retrieves data from one or more tables based on specified columns.	`SELECT column1, column2 FROM table_name;`
select_related	Performs inner join.	`MyModel.objects.select_related('related_model')`
startswith	Determines whether a string begins with the characters of a specified string.	`MyModel.objects.filter(field__startswith="value")`
SUM	Calculates the sum of values in a column.	`SELECT SUM(column1) FROM table_name;`
Sum()	Provides the sum of a field.	`MyModel.objects.aggregate(Sum('field'))`
UPDATE	Modifies data in a table based on specified conditions.	`UPDATE table_name SET column1 = value1 WHERE condition;`

UpdateView	Displays a form to update an existing object.	`class MyUpdateView(UpdateView):`
		`model = MyModel`
		`template_name = 'my_template.html'`
		`fields = '__all__' # or specify a list of fields`
		`pk_url_kwarg = 'my_model_id' # default: pk`

Usage – Static content	Code to style the HTML templates and provide interactivity to web pages.	`<link href="{% static 'your_app/css/style.css' %}" rel="stylesheet">`
		`<script src="{% static 'your_app/js/script.js' %}"></script>`
		`<img src="{% static 'your_app/img/logo.png' %}" alt="Logo">`

WHERE	Filters data based on specified conditions.	`SELECT column1, column2 FROM table_name WHERE condition;`

Skills Network