

Topic: Custom Management Command 9: Importing Large Data Using Celery & Redis Debugging

Speaker: Udemy Instructor Rathan Kumar | Notebook: Django: Automating Common Tasks



1. After we installed, set up, and tested our Celery and Redis, it is time to use this on our IMPORTDATA function.
2. Our PROBLEM is that when we import large data like ONE MILLION records of employees, it may take some time (after testing it, it took at least 13 minutes long to save only 130K records, so it might take longer to save all the 1M records.). Our SOLUTION is to use the CELERY function to handle this time-consuming process while we see messages like 'Your data have been saved successfully' and you can do other things.
3. We create a new CELERY function in our DATAENTRY\TASKS.PY and update as:

```
1 # from the project folder in CELERY.PY, we load the variable APP
2 from autocommontasks_main.celery import app
3 import time
4 from django.core.management import call_command
5
6
7 @app.task # uses a decorator as a celery task for testing
8 def celery_test_task():
9     time.sleep(10) # simulation of any task that's going to take 10 seconds
10    return 'Task executed successfully.'
11
12
13 @app.task # uses a decorator as a celery task for importing large data
14 def import_data_task(file_path, model_name):
15     try:
16         # trigger the custom-management import data command
17         call_command('importdata', file_path, model_name)
18         # passes this custom-made message to our web page
19     except Exception as e:
20         raise e
21
```

4. In DATAENTRY\VIEWS.PY, we have to update our IMPORT_DATA function, so that the importing part is handled by CELERY instead of by DJANGO.

FROM:

```
dataentry > views.py > import_data
9 def import_data(request):
24     file_path = base_url + relative_path # absolute path of the file
25
26     # trigger the custom-management import data command
27     try:
28         call_command('importdata', file_path, model_name)
29         # passes this custom-made message to our web page
30         messages.success(
31             request, 'You have successfully imported the file.')
32     except Exception as e:
33         # raise e
34         # passes this error type to our web page
35         messages.error(request, str(e))
36
37     return redirect('import_data')
38
39 else:
40     # calls our dataentry\utils.py to extract only user-created models
41     custom_models = get_all_custom_models()
42     # print(custom_models)
43     context = {
44         'custom_models': custom_models,
45     }
46     return render(request, 'dataentry/importdata.html', context)
```

TO:

```
dataentry > views.py > import_data
1 from django.shortcuts import render, redirect
2 from .utils import get_all_custom_models
3 from uploads.models import Upload
4 from django.conf import settings
5 from django.contrib import messages
6 # imports our celery function
7 from .tasks import import_data_task
8
9
10 def import_data(request):
11     if request.method == 'POST':
12         # gets the user-submitted file using its path
13         file_path = request.FILES.get('file_path')
14         # gets the user-submitted MODEL name
15         model_name = request.POST.get('model_name')
16
17         # store this file inside the Upload model
18         upload = Upload.objects.create(file=file_path, model_name=model_name)
19
20         # construct the full path of the file with the file_path for import
21         relative_path = str(upload.file.url)
22         # gets the directory of our root directory
23         base_url = str(settings.BASE_DIR)
24
25         file_path = base_url + relative_path # absolute path of the file
26
27         # call the Celery function from dataentry\tasks.py
28         import_data_task.delay(file_path, model_name)
29
30         # show the message the user
31         messages.success(
32             request, 'Your data is being imported, you will be notified once this ia done.')
33         return redirect('import_data')
34
35     else:
36         # calls our dataentry\utils.py to extract only user-created models
37         custom_models = get_all_custom_models()
38         # print(custom_models)
39         context = {
40             'custom_models': custom_models,
41         }
42         return render(request, 'dataentry/importdata.html', context)
```

IMPORTANT REMINDER: IF YOU JUST RUN YOUR DJANGO SERVER RIGHT AWAY, YOU WONT SEE ANY UPDATE. THIS IS BECAUSE YOU NEED TO RUN YOUR CELERY COMMAND AGAIN:

← → ↻ ⓘ http://127.0.0.1:8000/admin/dataentry/student/

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

- Groups + Add
- Users + Add

DATAENTRY

- Customers + Add
- Employees + Add
- Students + Add**

UPLOADS

- Uploads + Add

☐ Scarlett Smith

☐ Alexander Johnson

☐ Ella Wilson

☐ Elijah Davis

☐ Sophia Hernandez

☐ Henry Martinez

☐ Lily Perez

☐ Christopher King

☐ Grace Turner

☐ Joseph Adams

☐ Harper Harris

☐ Daniel Green

☐ Amelia Scott

☐ Samuel Young

☐ Charlotte White

☐ Benjamin Allen

☐ Mia Martinez

☐ William Hall

☐ Ava Rodriguez

☐ Alexander Clark

☐ Isabella Garcia

☐ Ethan Martinez

☐ Sophia Taylor

☐ James Anderson

☐ Olivia Jones

☐ David Miller

☐ Jessica Lee

☐ Daniel Wilson

☐ Sarah Davis

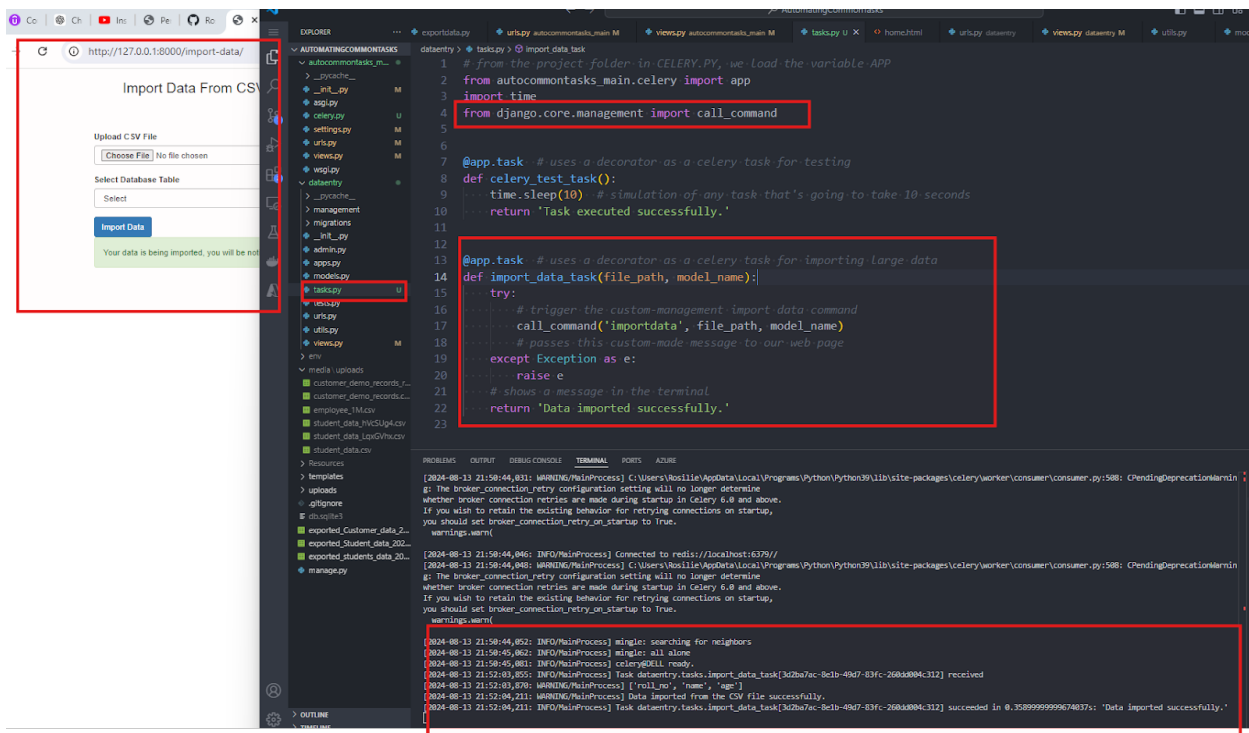
☐ Michael Brown

☐ Emily Johnson

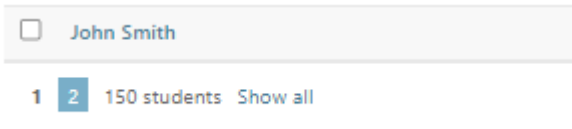
☐ John Smith

100 students

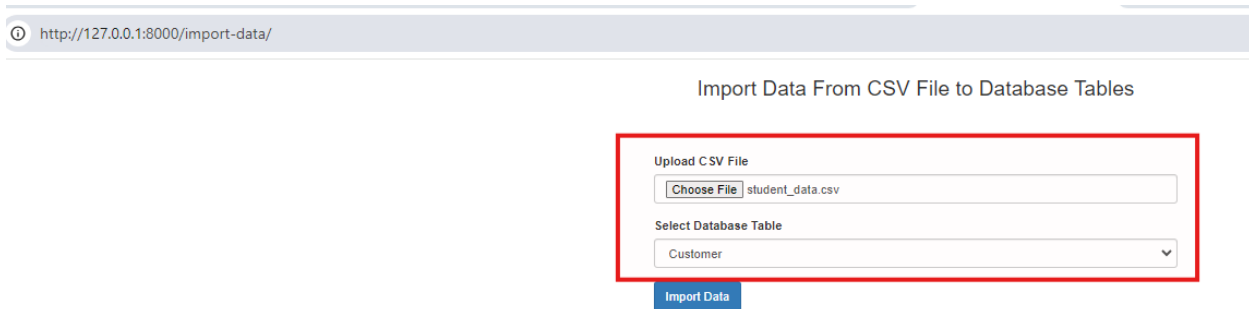
If we need to add a SUCCESS message in our CELERY FUNCTION for an upload of STUDENT CSV file again, we update and run our celery reload command again.

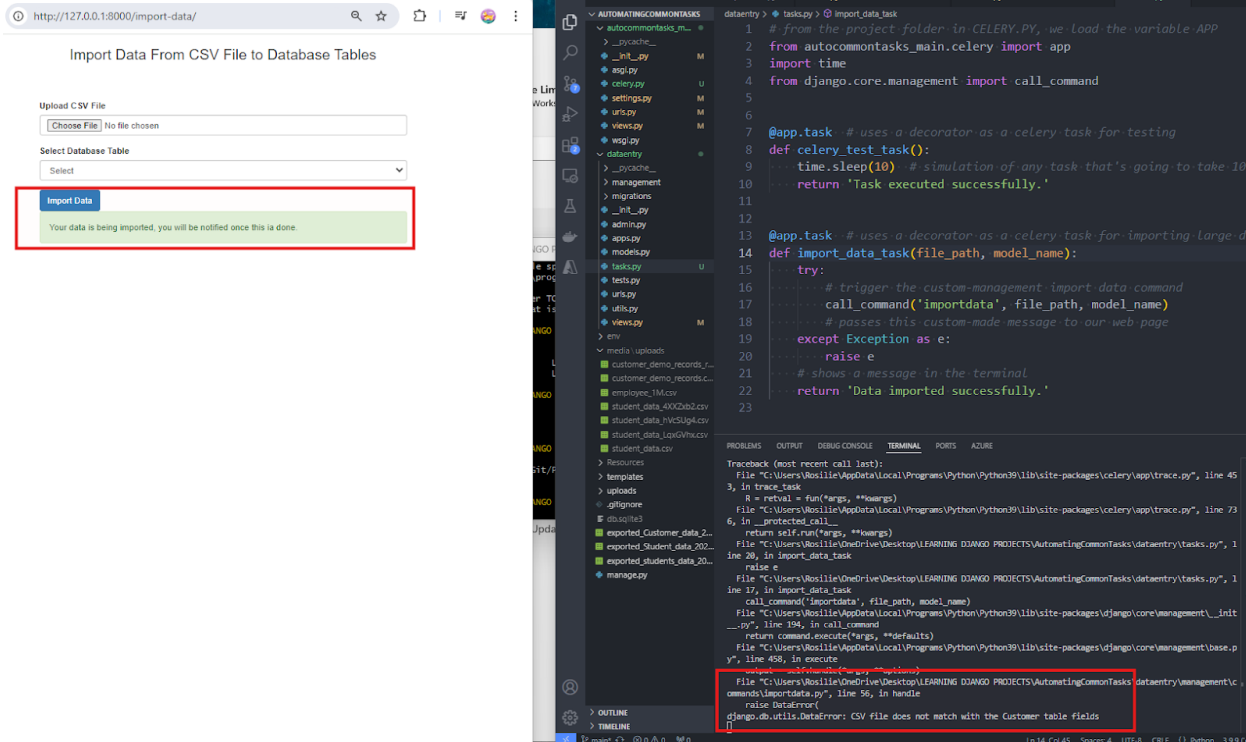


Reloading our ADMIN Dashboard for STUDENT TABLE, IT SHOULD INCREASE IN NUMBER



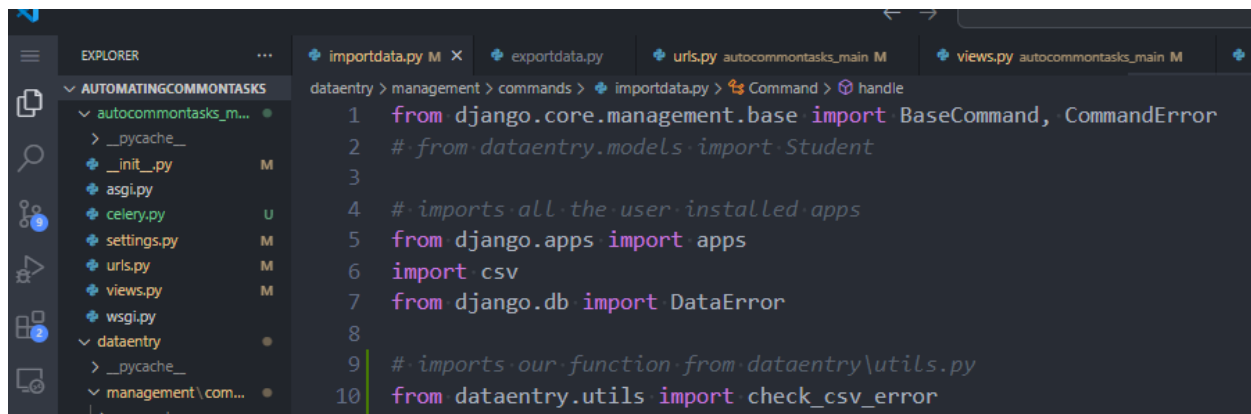
5. When we try to import a CSV file to an incorrect model, Celery will throw an error but Django won't.





To correct this, we have to an error - handling command BEFORE THE CELERY FUNCTION is being called.

So, go to DATAENTRYUTILS.PY and move some blocks from MANAGEMENTCOMMANDS\IMPORTDATA.PY. So, our IMPORTDATA.PY updated code now:



```
16 class Command(BaseCommand):
17     ...
18     ...
19     ... # accepts a user input like the filename & model_name
20     def add_arguments(self, parser):
21         parser.add_argument('file_path', type=str, help='Path to the CSV file')
22         parser.add_argument('model_name', type=str,
23                             help='name of the target model or table name')
24     ...
25     def handle(self, *args, **kwargs):
26         ... # gets the path of the source CSV file and the target model name
27         file_path = kwargs['file_path']
28         model_name = kwargs['model_name'].capitalize()
29         ...
30         ... # calls the check_csv_error function in dataentry\utils.property
31         target_model = check_csv_error(file_path, model_name)
32         ...
33         ...
34         with open(file_path, 'r') as file:
35             reader = csv.DictReader(file)
36             ... # print(reader)
37             ... # save into our desired target model if there is no error
38             for row in reader:
39                 ... # adds the row into our database Student table;
40                 ... # Student.objects.create(**row)
41                 ... # imports the CSV file into the user-provided model
42                 target_model.objects.create(**row)
43         self.stdout.write(self.style.SUCCESS(
44             'Data imported from the CSV file successfully.')
```

Our UTILS.PY updated code where we created a new function, `check_csv_error`, to allow Error Handling via UTILS.PY instead from IMPORTDATA.PY

```
1 from django.apps import apps
2 from django.core.management.base import CommandError
3 import csv
4 from django.db import DataError
5
6 # extracts only the user-created models
7 # excludes default models like Users, etc
8
9
10 def get_all_custom_models():
11     ... # lists the default models that we dont want to display in our Upload Form
12     default_models = ['ContentType', 'Session',
13                       'LogEntry', 'Group', 'Permission', 'User', 'Upload']
14
15     custom_models = []
16     for model in apps.get_models():
17         ... # print(model.__name__)
18         ... # checks if the model name is in the list of default model or not
19         if model.__name__ not in default_models:
20             custom_models.append(model.__name__)
21         ... # print(model)
22     return custom_models
23
```

```
def check_csv_error(file_path, model_name):  
    """# celery custom function to check for errors before celery does its task  
    # search for the model across all installed apps  
    target_model = None  
    for my_app_config in apps.get_app_configs():  
        """# Try to search for the target model where we will save our imported data  
        try:  
            """# returns the model name of the app  
            target_model = apps.get_model(my_app_config.label, model_name)  
            break """# stops search once the model is found  
        except LookupError:  
            """# model is not found, then keep searching in the next app  
            continue  
        """# if model is empty/not found  
        if not target_model:  
            raise CommandError(f'Model "{model_name}" not found in any app.')  
    """# get all the field names of the model that we found EXCEPT THE PK or ID  
    model_fields = [field.name for field in target_model._meta.fields if field.name !=  
                    target_model._meta.pk.name and field.name != target_model._meta.pk.name] # id  
    """# print(model_fields)  
    try:  
        """# opens the file for reading and closes it automatically  
        with open(file_path, 'r') as file:  
            """# reads the csv file including the header  
            reader = csv.DictReader(file)  
            """# gets the header names of the csv file  
            csv_header = reader.fieldnames  
            """# compare CSV header with the model's field names and throw appropriate message  
            if csv_header != model_fields:  
                raise DataError(  
                    f'CSV file does not match with the {model_name} table fields')  
            except Exception as e:  
                raise e  
    """# if there is no error, we return the model name  
    return target_model
```

6. We also updated our DATAENTRYVIEWS.PY to be able to call our error-handling function, check_for_csv,

```
1 from django.shortcuts import render, redirect  
2 from .utils import get_all_custom_models, check_csv_error  
3 from uploads.models import Upload  
4 from django.conf import settings  
5 from django.contrib import messages  
6 # imports our celery function  
7 from .tasks import import_data_task  
8
```

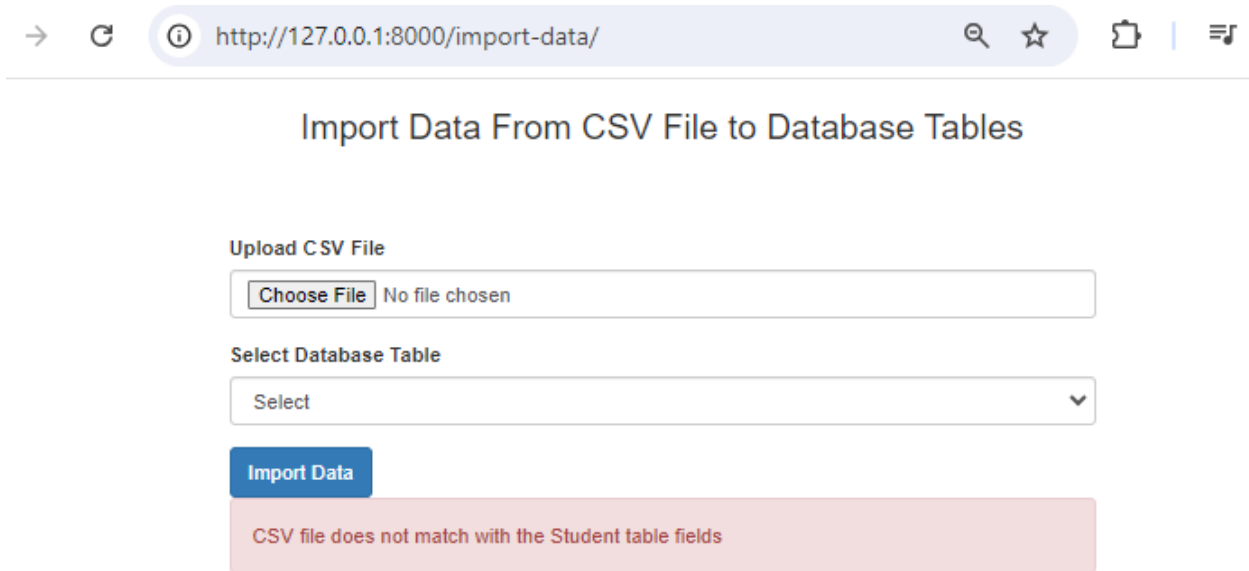


```

10 def import_data(request):
11     if request.method == 'POST':
12         # gets the user-submitted file using its path
13         file_path = request.FILES.get('file_path')
14         # gets the user-submitted MODEL name
15         model_name = request.POST.get('model_name')
16
17         # store this file inside the Upload model
18         upload = Upload.objects.create(file=file_path, model_name=model_name)
19
20         # construct the full path of the file with the file_path for import
21         relative_path = str(upload.file.url)
22         # gets the directory of our root directory
23         base_url = str(settings.BASE_DIR)
24
25         file_path = base_url + relative_path # absolute path of the file
26
27         # check for the CSV errors, if there is an error, we wont call our celery function for import
28         try:
29             # calls the check_csv_error function in dataentry\utils.property
30             check_csv_error(file_path, model_name)
31         except Exception as e:
32             messages.error(request, str(e))
33             return redirect('import_data')
34
35         # if no error, call the Celery function for data import from dataentry\tasks.py
36         import_data_task.delay(file_path, model_name)
37
38         # show the message the user
39         messages.success(
40             request, 'Your data is being imported, you will be notified once this ia done.')
41         return redirect('import_data')
42     else:
43         # calls our dataentry\utils.py to extract only user-created models
44         custom_models = get_all_custom_models()
45         # print(custom_models)
46         context = {
47             'custom_models': custom_models,
48         }
49         return render(request, 'dataentry/importdata.html', context)
50

```

7. Run your Django-server (use \$ python manage.py runserver), Redis (use \$ redis-cli ping), Celery (use \$ celery -A autocommontasks_main worker --loglevel=info --pool=solo) and upload Customer to Student Model, we get this user friendly message instead.



8. Now run this with your large data like EMPLOYEE.CSV to employee model. You should be able to see a user-friendly message instead of your webpage just loading for so long.

Import Data From CSV File to Database Tables

Upload CSV File

employee_1M.csv

Select Database Table

Employee ▼

BEFORE UPLOADING:

- Denise Morgan Library Assistant
- Susan Pope Library Assistant
- Mark Carter Library Assistant

1 2 3 4 ... 1336 1337 133700 employees

AFTER UPLOADING: Your celery must be able to receive the task request but now the user can do other things since celery handles the time-consuming data importing in the background.

