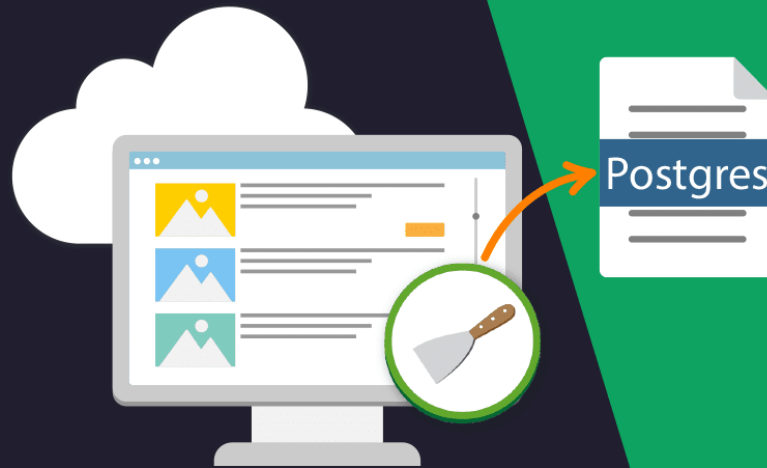


THE SCRAPY PLAYBOOK

Scrapy Saving Data to Postgres Database



Saving Scraped Data To Postgres Database With Scrapy Pipelines

If your scraping a website, you need to save that data somewhere. A great option is Postgres as it is a battletested database, trusted by thousands of companies to handle huge amounts of data.

In this guide, we will go through how to save our data to a Postgres database using Scrapy pipelines:

- [What Are Scrapy Item Pipelines?](#)
- [Setting Up A Postgres Database](#)
- [Saving Data To A Postgre Database](#)
- [Only Saving New Data](#)

First, let's go over what are **Scrapy Item Pipelines**.

Need help scraping the web?

Then check out [ScrapeOps](#), the complete toolkit for web scraping.



Proxy Manager



Scraper Monitoring



Job Scheduling

What Are Scrapy Item Pipelines?

[Item Pipelines](#) are Scrapy's way of process data scraped by spiders.

After an item has been scraped by a spider, it is sent to the Item Pipeline which processes it through a sequence of steps that can be configured to clean and process the scraped data before ultimately saving it somewhere.

You can use Item Pipelines to:

- Clean HTML data
- Validate scraped data
- Checking for and removing duplicate data
- Storing the data in database

For the purpose of this guide, we're going to focus on using Item Pipelines to store data in a Postgres database.

Setting Up A Postgres Database

To get started we first need to setup a Postgres database.

Either you can set one up on your local machine by using [one of the following downloads](#).

Or you could get a hosted version with cloud provider like [DigitalOcean](#).

Once setup you should have access to the database connection details of your database:

```
host="localhost",  
database="my_database",  
user="root",  
password="123456"
```

Saving Data to a Postgres Database

Okay, now let's now integrate saving data into our Postgres database.

1. Install psycopg2

To interact with our database we will need a library to handle the interaction. For this will install `psycopg2`.

```
pip install psycopg2
```

We will use `psycopg2` to interact with our Postgres database.

2. Setup Our Pipeline

The next step is we need to open our `pipelines.py` file and set up our pipeline.

When you open your `pipelines.py` file, the default file should look like this:

```
# pipelines.py  
  
from itemadapter import ItemAdapter  
  
class PostgresDemoPipeline:  
    def process_item(self, item, spider):  
        return item
```

Now we will configure this empty pipeline to store our data.

Note: For this guide I created a Scrapy project called **postgres_demo** (thus the default pipeline is `PostgresDemoPipeline`), and am use this spider:

```
# spiders/quotes.py

import scrapy
from postgres_demo.items import QuoteItem

class QuotesSpider(scrapy.Spider):
    name = 'quotes'

    def start_requests(self):
        url = 'https://quotes.toscrape.com/'
        yield scrapy.Request(url, callback=self.parse)

    def parse(self, response):
        quote_item = QuoteItem()
        for quote in response.css('div.quote'):
            quote_item['text'] = quote.css('span.text::text').get()
            quote_item['author'] = quote.css('small.author::text').get()
            quote_item['tags'] = quote.css('div.tags a.tag::text').getall()
            yield quote_item
```

And the Item:

```
# items.py

from scrapy.item import Item, Field

class QuoteItem(Item):
    text = Field()
    tags = Field()
    author = Field()
```

3. Connect to Postgres & Create a Table

First, we're going to `import psycopg2` into our `pipelines.py` file, and create an `__init__` method that we will use to create our database and table.

```
# pipelines.py

import psycopg2

class PostgresDemoPipeline:

    def __init__(self):
        pass

    def process_item(self, item, spider):
        return item
```

Inside the `__init__` method, we will configure the pipeline to do the following everytime the pipeline gets activated by a spider:

1. Try to connect to our database `quotes` , but if it doesn't exist create the database.
2. Create a cursor which we will use to execute SQL commands in the database.
3. Create a new table `quotes` with the columns `content` , `tags` and `author` , if one doesn't already exist in the database.

```
# pipelines.py

import psycopg2

class PostgresDemoPipeline:

    def __init__(self):
        ## Connection Details
        hostname = 'localhost'
        username = 'postgres'
        password = '*****' # your password
        database = 'quotes'

        ## Create/Connect to database
        self.connection = psycopg2.connect(host=hostname, user=username, password=password,
dbname=database)

        ## Create cursor, used to execute commands
        self.cur = self.connection.cursor()

        ## Create quotes table if none exists
        self.cur.execute("""
CREATE TABLE IF NOT EXISTS quotes(
```

```

        id serial PRIMARY KEY,
        content text,
        tags text,
        author VARCHAR(255)
    )
"""

```

```

def process_item(self, item, spider):
    return item

```

4. Save Scraped Items Into Database

Next, we're going to use the `process_item` event inside in our Scrapy pipeline to store the data we scrape into our Postgres database.

The `process_item` will be activated everytime, a item is scraped by our spider so we need to configure the `process_item` method to insert the items data in the database.

We will also the `close_spider` method, which will be called when the Spider is shutting down, to close our connections to the cursor and database to avoid leaving the connection open.

```

# pipelines.py

import psycopg2

class PostgresDemoPipeline:

    def __init__(self):
        ## Connection Details
        hostname = 'localhost'
        username = 'postgres'
        password = '*****' # your password
        database = 'quotes'

        ## Create/Connect to database
        self.connection = psycopg2.connect(host=hostname, user=username, password=password,
        dbname=database)

        ## Create cursor, used to execute commands

```

```

self.cur = self.connection.cursor()

## Create quotes table if none exists
self.cur.execute("""
CREATE TABLE IF NOT EXISTS quotes(
    id serial PRIMARY KEY,
    content text,
    tags text,
    author VARCHAR(255)
)
""")

def process_item(self, item, spider):

    ## Define insert statement
    self.cur.execute(""" insert into quotes (content, tags, author) values (%s,%s,%s)""", (
        item["text"],
        str(item["tags"]),
        item["author"]
    ))

    ## Execute insert of data into database
    self.connection.commit()
    return item

def close_spider(self, spider):

    ## Close cursor & connection to database
    self.cur.close()
    self.connection.close()

```

5. Activate Our Item Pipeline

Finally, to activate our Item Pipeline we need to include it in our `settings.py` file:

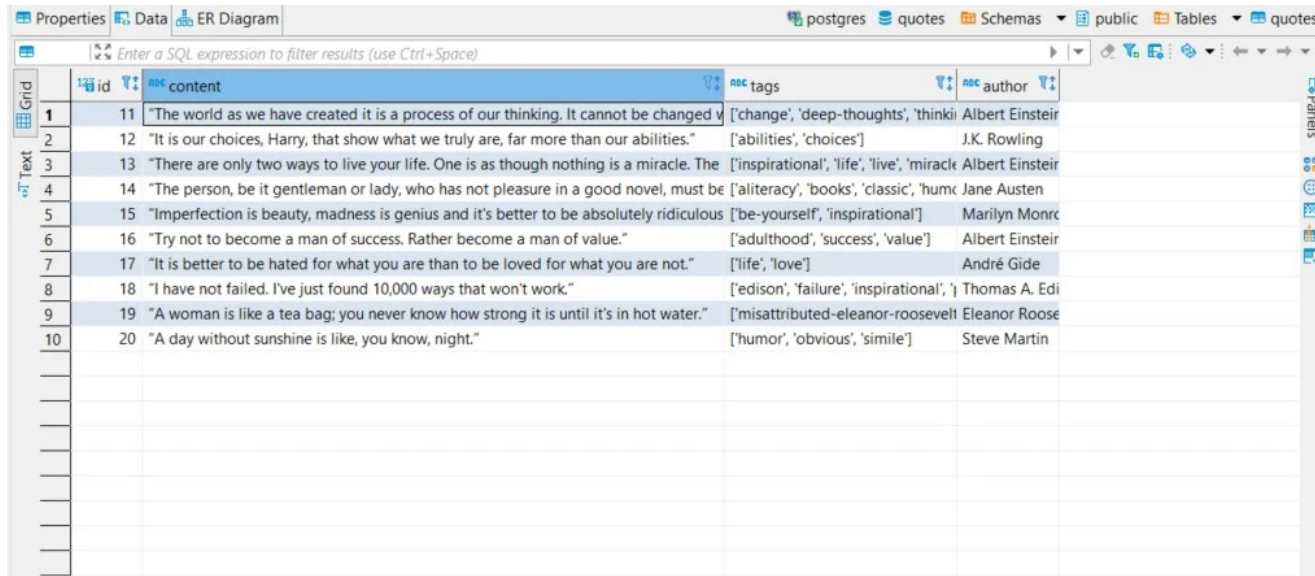
```

# settings.py

ITEM_PIPELINES = {
    'postgres_demo.pipelines.PostgresDemoPipeline': 300,
}

```

Now, when we run our **quotes** spider the **PostgresDemoPipeline** will store all the scraped items in the database.



	id	content	tags	author
1	11	"The world as we have created it is a process of our thinking. It cannot be changed v	['change', 'deep-thoughts', 'thinki	Albert Einsteir
2	12	"It is our choices, Harry, that show what we truly are, far more than our abilities."	['abilities', 'choices']	J.K. Rowling
3	13	"There are only two ways to live your life. One is as though nothing is a miracle. The	['inspirational', 'life', 'live', 'miracle	Albert Einsteir
4	14	"The person, be it gentleman or lady, who has not pleasure in a good novel, must be	['aliteracy', 'books', 'classic', 'hum	Jane Austen
5	15	"Imperfection is beauty, madness is genius and it's better to be absolutely ridiculous	['be-yourself', 'inspirational']	Marilyn Monro
6	16	"Try not to become a man of success. Rather become a man of value."	['adulthood', 'success', 'value']	Albert Einsteir
7	17	"It is better to be hated for what you are than to be loved for what you are not."	['life', 'love']	André Gide
8	18	"I have not failed. I've just found 10,000 ways that won't work."	['edison', 'failure', 'inspirational', 't	Thomas A. Edi
9	19	"A woman is like a tea bag; you never know how strong it is until it's in hot water."	['misattributed-eleanor-roosevelt	Eleanor Roose
10	20	"A day without sunshine is like, you know, night."	['humor', 'obvious', 'simile']	Steve Martin

If you don't have a SQL database viewer you can use [DBeaver](#).

Only Saving New Data

Okay, now we have a Item Pipeline that saves all scraped items to our Postgres database. However, what if we only want to save new data that we haven't scraped before.

We can easily reconfigure our pipeline to do this by having it check the database if the item is already in the database before inserting it again.

To do this, I'm going to create a new pipeline in our `pipelines.py` file called **PostgresNoDuplicatesPipeline** and change the `process_item` method so that it only inserts new data into the database.

It will look up the `item['text']` in the database first, and only if it isn't there will insert the new item.

```
# pipelines.py
```



```

import psycopg2

class PostgresNoDuplicatesPipeline:

    def __init__(self):
        ## Connection Details
        hostname = 'localhost'
        username = 'postgres'
        password = '*****' # your password
        database = 'quotes'

        ## Create/Connect to database
        self.connection = psycopg2.connect(host=hostname, user=username, password=password,
dbname=database)

        ## Create cursor, used to execute commands
        self.cur = self.connection.cursor()

        ## Create quotes table if none exists
        self.cur.execute("""
CREATE TABLE IF NOT EXISTS quotes(
    id serial PRIMARY KEY,
    content text,
    tags text,
    author VARCHAR(255)
)
""")

    def process_item(self, item, spider):

        ## Check to see if text is already in database
        self.cur.execute("select * from quotes where content = %s", (item['text'],))
        result = self.cur.fetchone()

        ## If it is in DB, create log message
        if result:
            spider.logger.warn("Item already in database: %s" % item['text'])

        ## If text isn't in the DB, insert data
        else:

            ## Define insert statement
            self.cur.execute(""" insert into quotes (content, tags, author) values (%s,%s,%s)""", (

```

```

        item["text"],
        str(item["tags"]),
        item["author"]
    ))

    ## Execute insert of data into database
    self.connection.commit()
    return item

def close_spider(self, spider):

    ## Close cursor & connection to database
    self.cur.close()
    self.connection.close()

```

To activate, this pipeline we also need to update our `settings.py` to use the `PostgresNoDuplicatesPipeline` not the previous `PostgresDemoPipeline` pipeline:

```

# settings.py

ITEM_PIPELINES = {
    ## 'postgres_demo.pipelines.PostgresDemoPipeline': 300,
    'postgres_demo.pipelines.PostgresNoDuplicatesPipeline': 300,
}

```

Now, when we run our **quotes** spider, the pipeline will only store new data that isn't already in the database.

More Scrapy Tutorials

We've covered the basics of saving data to a Postgres database with Scrapy Pipelines.

If you would like to learn more about saving data, then be sure to check out these guides:

- [Saving Data to CSV](#)
- [Saving Data to JSON](#)

- [Saving Data to SQLite Database](#)
- [Saving Data to MySQL Database](#)
- [Saving CSV/JSON Files to Amazon AWS S3 Bucket](#)

If you would like to learn more about Scrapy in general, then be sure to check out [The Scrapy Playbook](#).