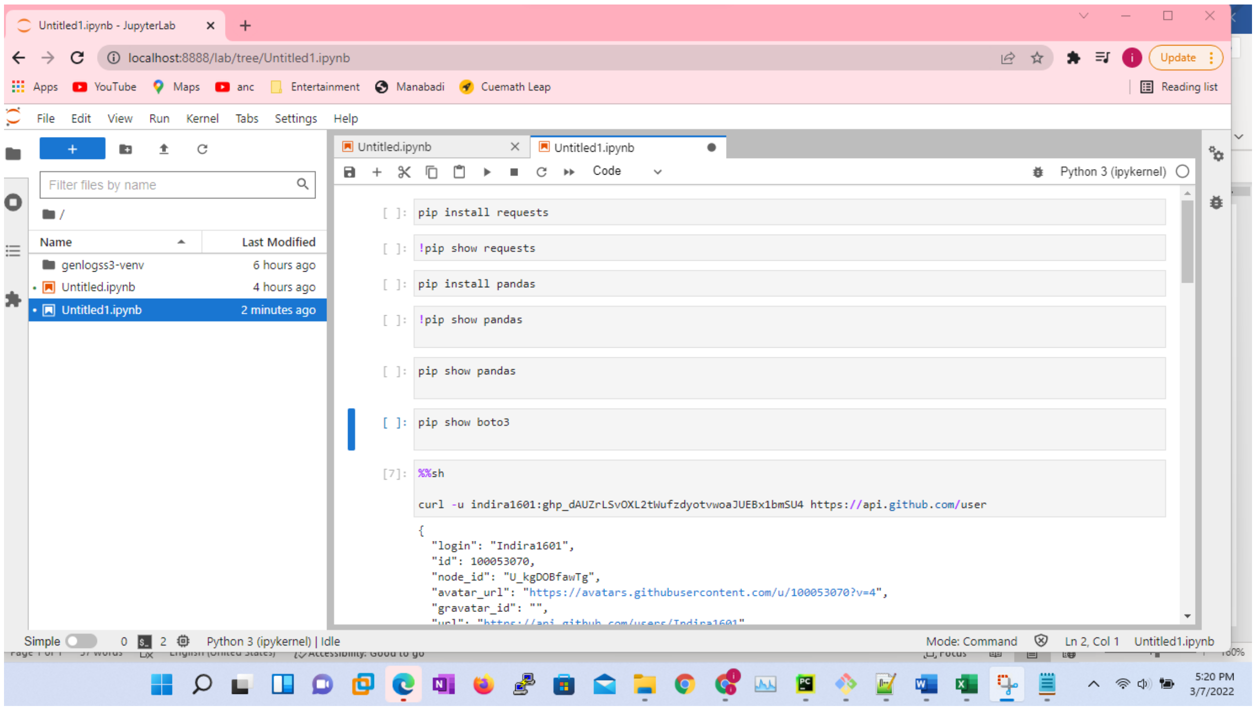
Write GitHub Data to Dynamodb

Requirements: boto3, pandas to be installed

We will be using Pandas to process the data before writing to Dynamo DB table. To write the data to AWS dynamo, we will be using boto3.

Make sure that Panda as well as boto3 installed by running commands like below.



```
[ ]: pip install requests

[ ]: !pip show requests

[ ]: pip install pandas

[ ]: !pip show pandas

[ ]: pip show pandas

[ ]: pip show boto3

[7]: %%sh

     curl -u indira1601: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 https://api.github.com/user

     {
       "login": "Indira1601",
       "id": 100053070,
       "node_id": "U_kgDOBfawTg",
       "avatar_url": "https://avatars.githubusercontent.com/u/100053070?v=4",
       "gravatar_id": "",
```

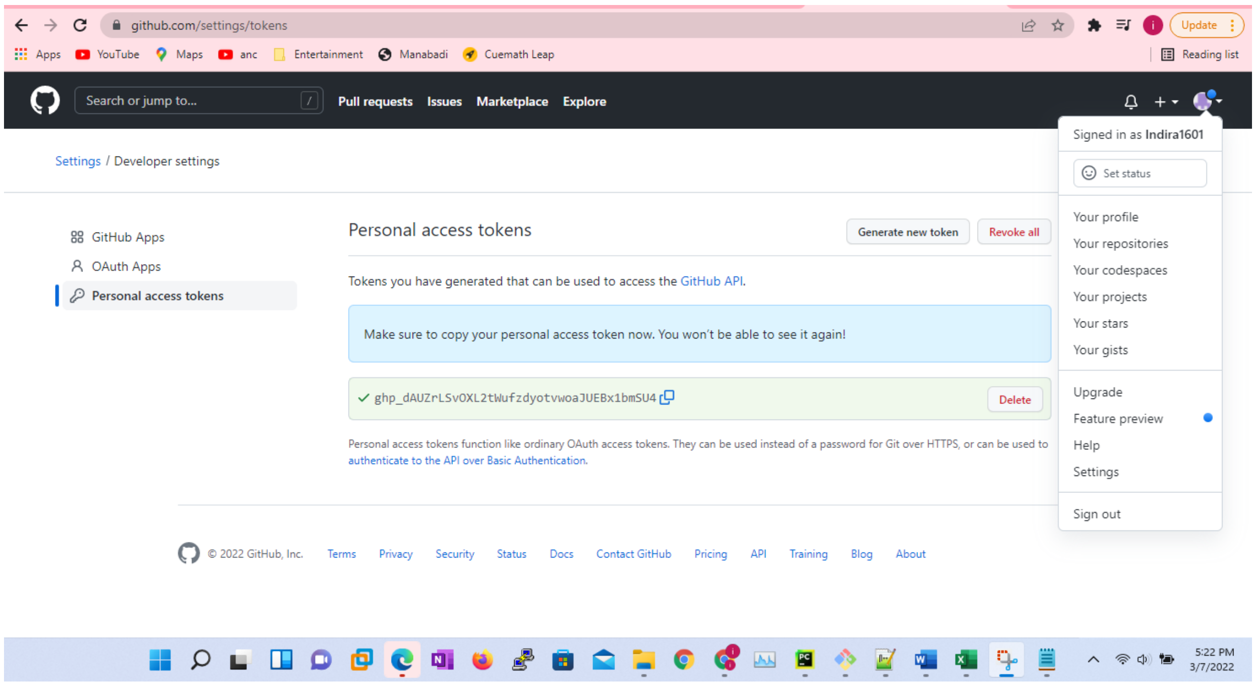
Figure 1Install the required libraries

Setting up GitHub API Token

Setup the GitHub Token so that we can make up 5000 API calls per hour.

Token can be create in GitHub account and example below.

Once you create the token either you can use `curl` or Python `requests` library to invoke the GitHub APIs to get the data.



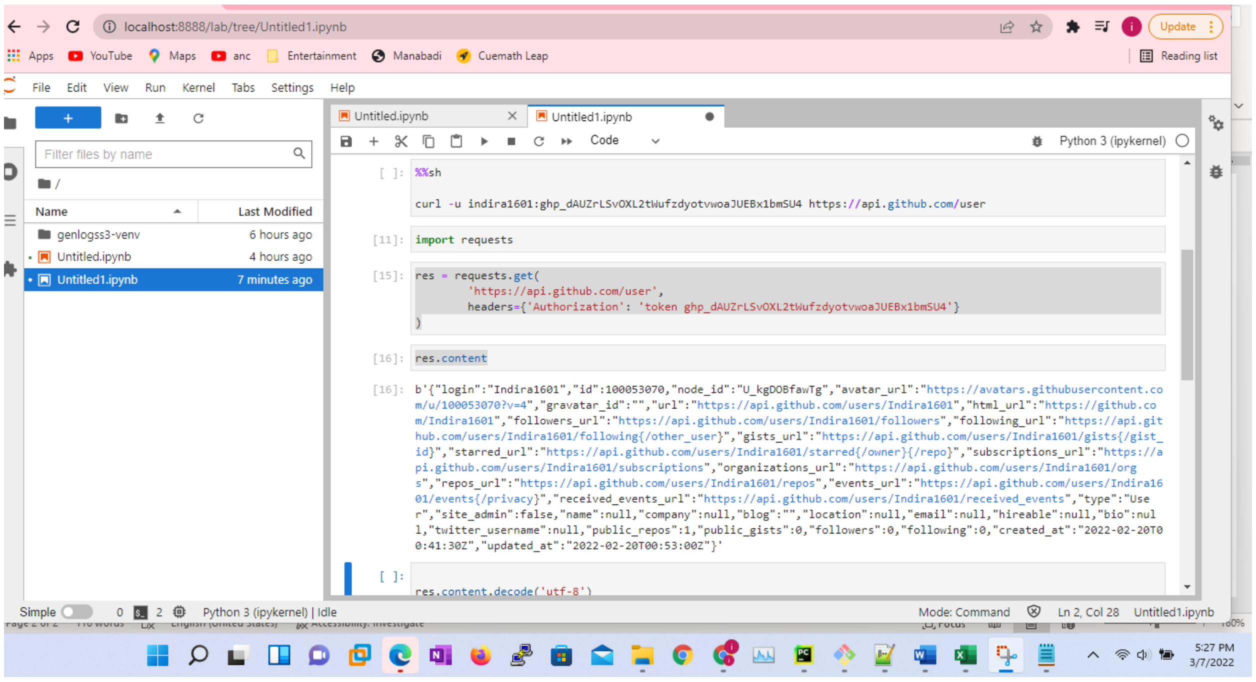
Personal access tokens

Tokens you have generated that can be used to access the GitHub API.

Make sure to copy your personal access token now. You won't be able to see it again!

```
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
```

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.



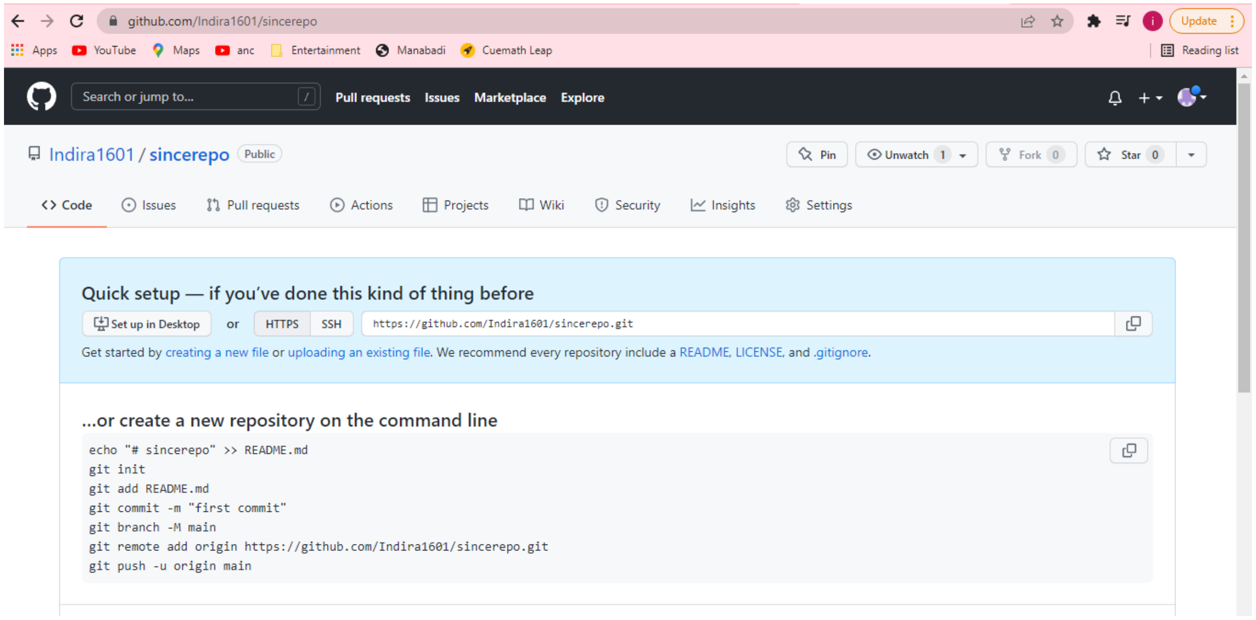
Name	Last Modified
genlogss3-venv	6 hours ago
Untitled.ipynb	4 hours ago
Untitled1.ipynb	7 minutes ago

```
[ ]: %%sh

     curl -u indira1601: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 https://api.github.com/user

[11]: import requests

[15]: res = requests.get(
              'https://api.github.com/user',
              headers={'Authorization': 'token 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'}
      )

[16]: res.content

[16]: b'{"login":"Indira1601","id":100053070,"node_id":"U_kgDOBfawTg","avatar_url":"https://avatars.githubusercontent.co
      m/u/100053070?v=4","gravatar_id":"","url":"https://api.github.com/users/Indira1601","html_url":"https://github.co
      m/Indira1601","followers_url":"https://api.github.com/users/Indira1601/followers","following_url":"https://api.git
      hub.com/users/Indira1601/following{/other_user}","gists_url":"https://api.github.com/users/Indira1601/gists{/gist_
      id}","starred_url":"https://api.github.com/users/Indira1601/starred{/owner}{/repo}","subscriptions_url":"https://a
      pi.github.com/users/Indira1601/subscriptions","organizations_url":"https://api.github.com/users/Indira1601/org
      s","repos_url":"https://api.github.com/users/Indira1601/repos","events_url":"https://api.github.com/users/Indira16
      01/events{/privacy}","received_events_url":"https://api.github.com/users/Indira1601/received_events","type":"Use
      r","site_admin":false,"name":null,"company":null,"blog":"","location":null,"email":null,"hireable":null,"bio":nul
      l,"twitter_username":null,"public_repos":1,"public_gists":0,"followers":0,"following":0,"created_at":"2022-02-20T0
      0:41:30Z","updated_at":"2022-02-20T00:53:00Z"}'

[ ]: res.content.decode('utf-8')
```

Created New Repository for "since":

In order to simulate the GitHub repo database and populate the table, we need to identify starting point and invoke list public repositories by passing it as part of "since" argument,

As GitHub is pretty heavy, you can define starting point by creating a new repository and getting id for it. Go to GitHub and create a new repository. Get the id using `requests` API and define it as starting point. We can use list repositories for user to get the repo id of just created repo. As we invoke list public repositories, we need to keep track of the last repo's id so that we can capture the information in incremental fashion.
We can use Dynamo DB to keep track of the last repo id after each call of list public repositories.

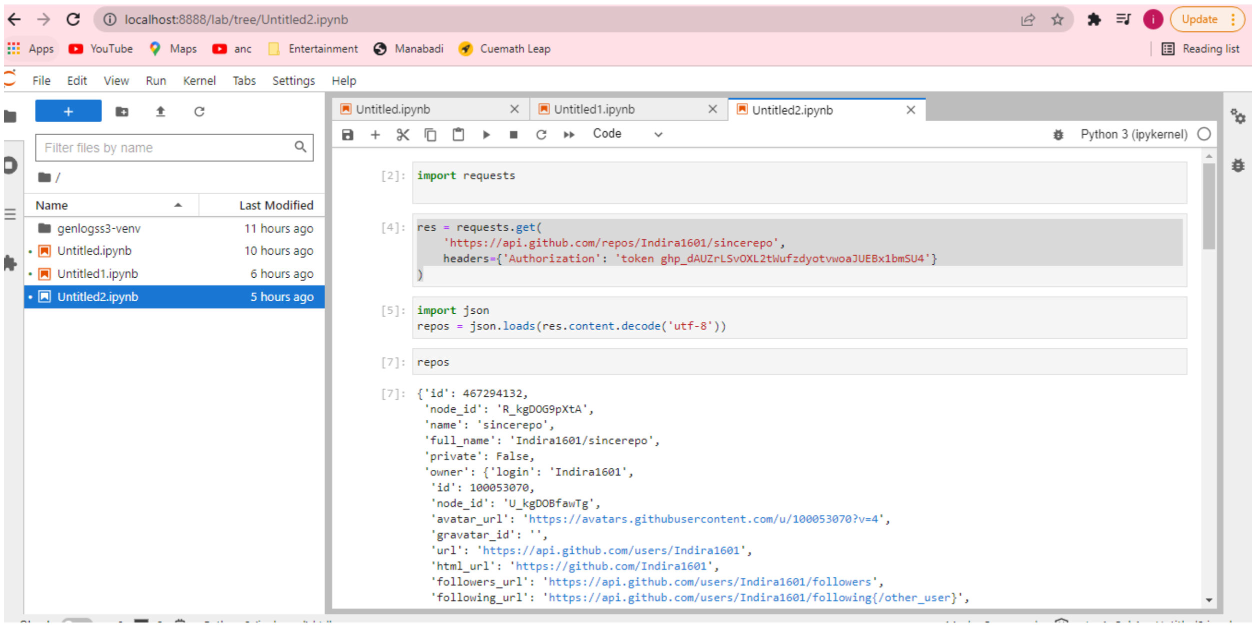

github.com/Indira1601/sincerepo
Indira1601 / sincerepo
Public
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Quick setup — if you've done this kind of thing before
Set up in Desktop
HTTPS
SSH
https://github.com/Indira1601/sincerepo.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# sincerepo" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/Indira1601/sincerepo.git
git push -u origin main
```

Figure 2Created the new repository since

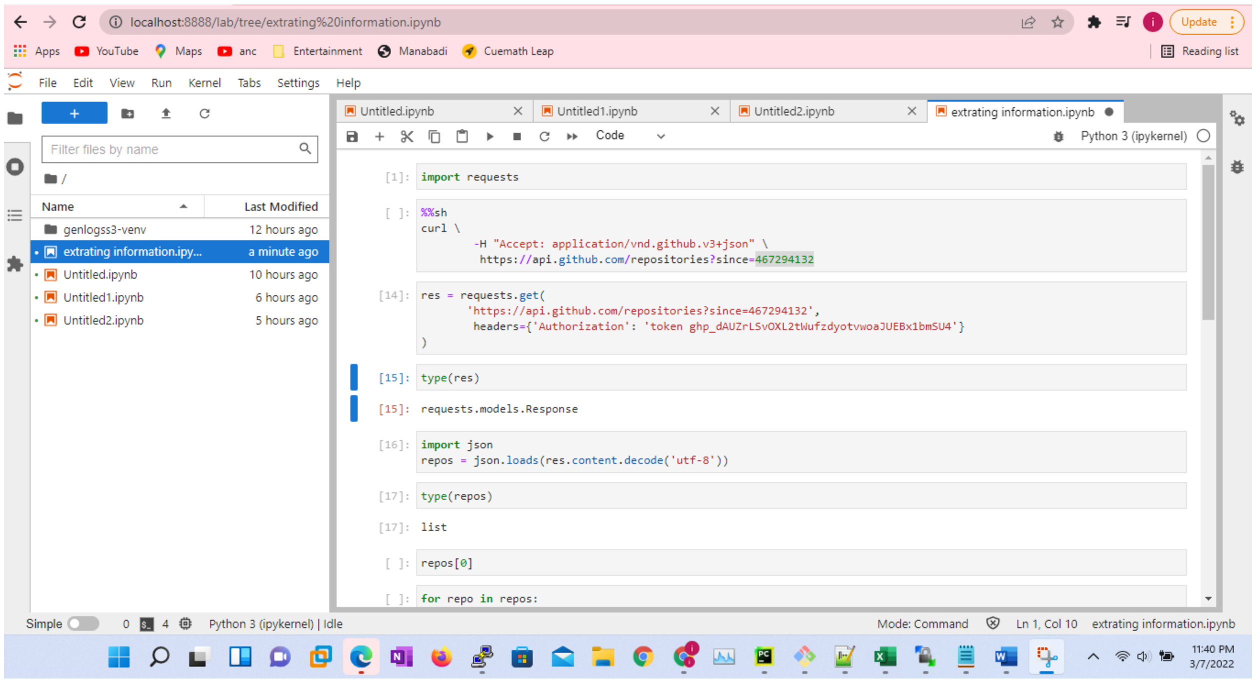


Name	Last Modified
genlogss3-venv	11 hours ago
Untitled.ipynb	10 hours ago
Untitled1.ipynb	6 hours ago
Untitled2.ipynb	5 hours ago

```
[2]: import requests

[4]: res = requests.get(
         'https://api.github.com/repos/Indira1601/sincerepo',
         headers={'Authorization': 'token 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'}
     )

[5]: import json
     repos = json.loads(res.content.decode('utf-8'))

[7]: repos

[7]: {'id': 467294132,
      'node_id': 'R_kgDOG9pXtA',
      'name': 'sincerepo',
      'full_name': 'Indira1601/sincerepo',
      'private': False,
      'owner': {'login': 'Indira1601',
       'id': 100053070,
       'node_id': 'U_kgDOBfawTg',
       'avatar_url': 'https://avatars.githubusercontent.com/u/100053070?v=4',
       'gravatar_id': '',
       'url': 'https://api.github.com/users/Indira1601',
       'html_url': 'https://github.com/Indira1601',
       'followers_url': 'https://api.github.com/users/Indira1601/followers',
       'following_url': 'https://api.github.com/users/Indira1601/following{/other_user}',
```

Extracting required information

we can extract required information using GitHub APIs. We will be using list public repositories and then get repository using id. Get list of repositories using “since”. Pick one repo details and then get details about specific repository.



```
[1]: import requests

[ ]: %%sh
     curl \
             -H "Accept: application/vnd.github.v3+json" \
              https://api.github.com/repositories?since=467294132

[14]: res = requests.get(
              'https://api.github.com/repositories?since=467294132',
              headers={'Authorization': 'token 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'}
      )

[15]: type(res)

[15]: requests.models.Response

[16]: import json
      repos = json.loads(res.content.decode('utf-8'))

[17]: type(repos)

[17]: list

[ ]: repos[0]

[ ]: for repo in repos:
```

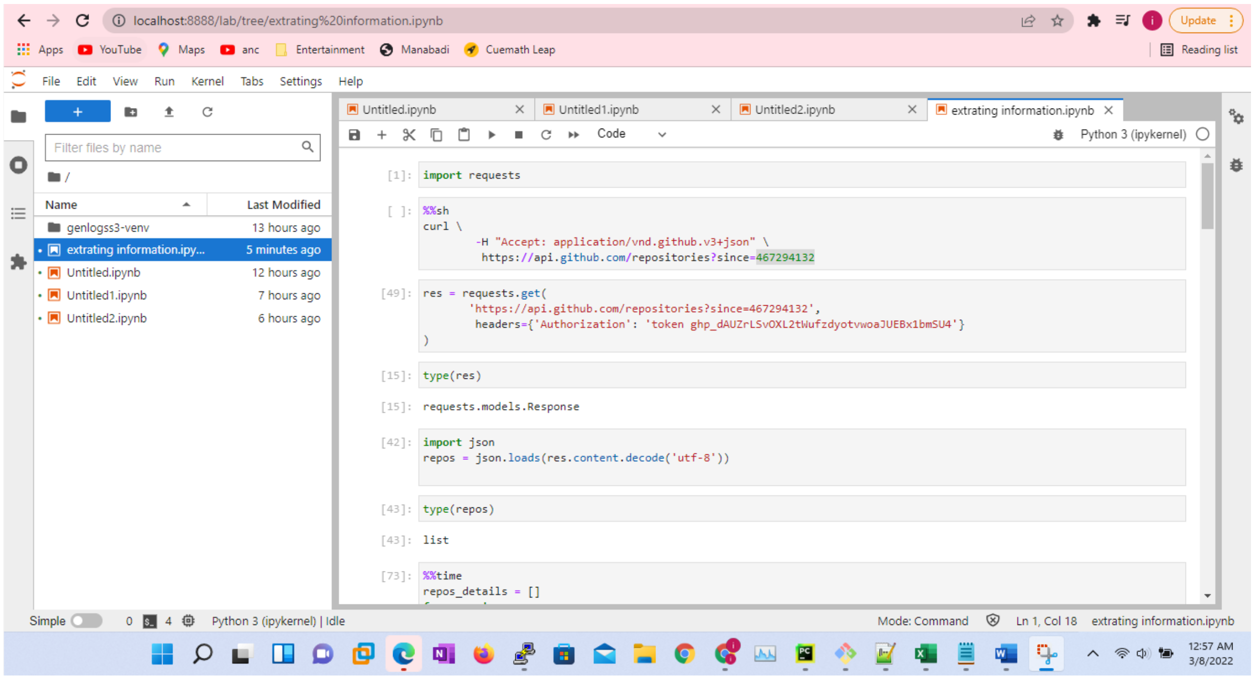
Processing Data

Process the data before storing into Dynamodb.

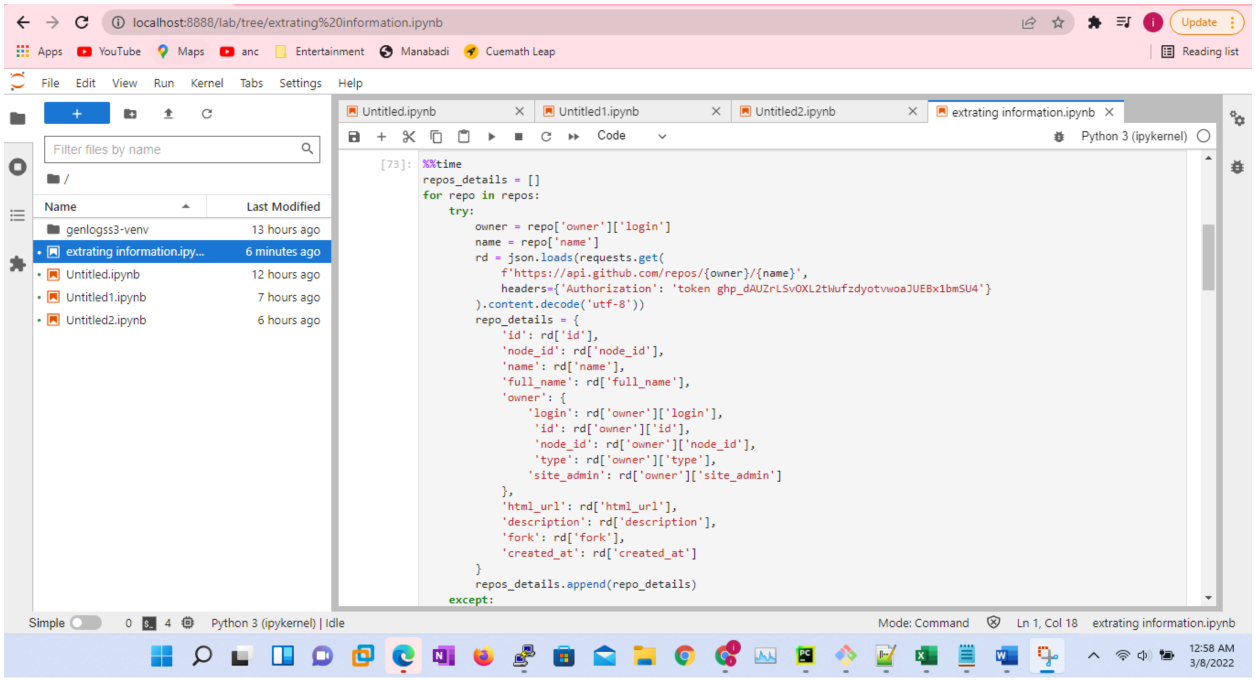
Id
node_id
name
full_name
owner.login
owner.id
owner.node_id
owner.type
owner.site_admin
html_url
description
fork
created_at

We will define owner as Map or dict with the fields we are looking for. Read the data from list public repositories up to 100.

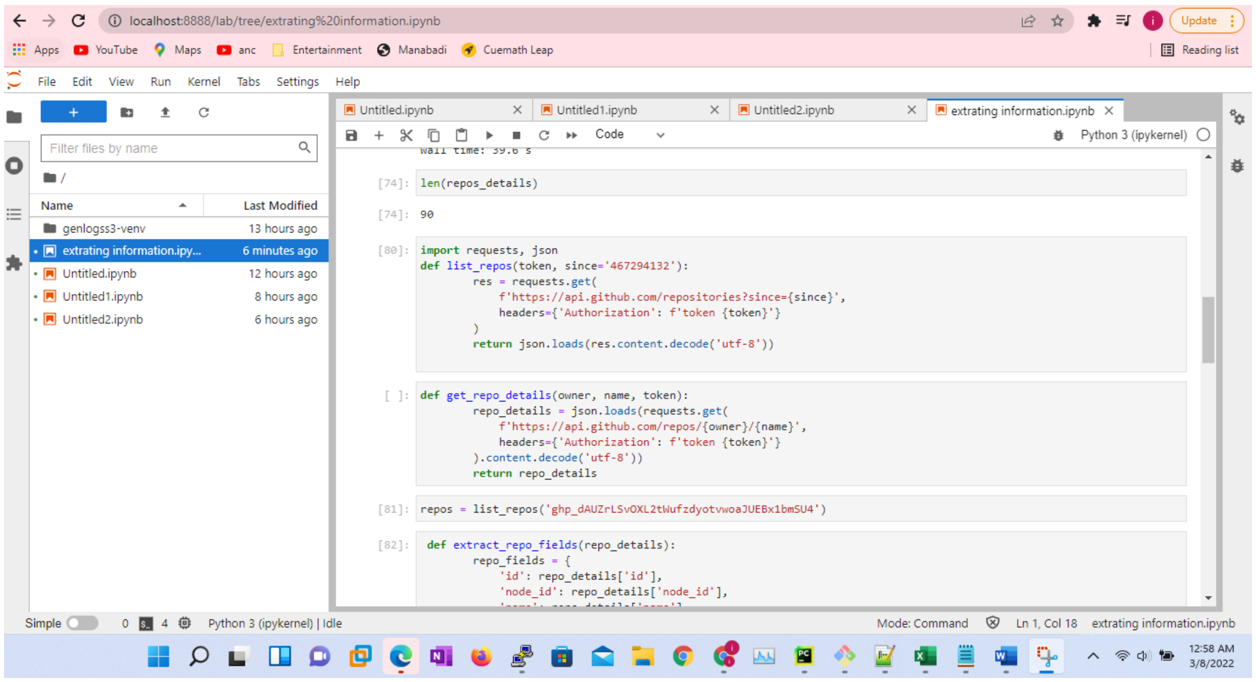
Get all the fields from get repository API. Build a collection so that we can write to the target database.



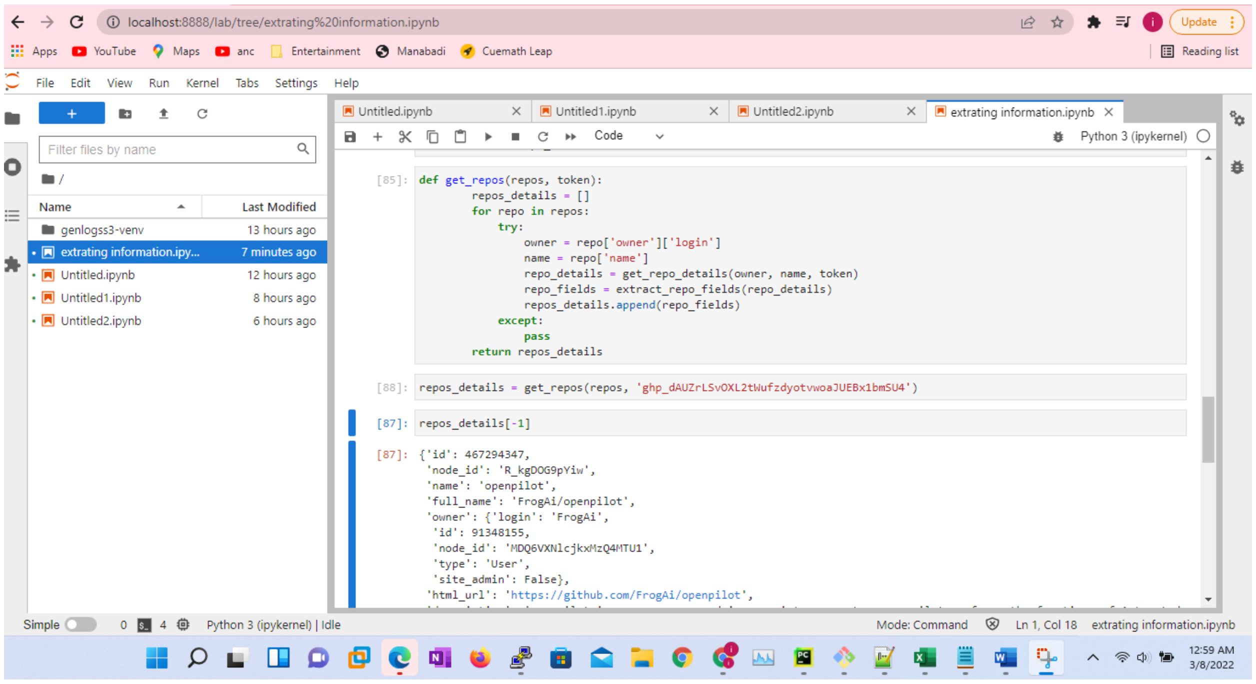
```
[1]: import requests

[ ]: %%sh
     curl \
             -H "Accept: application/vnd.github.v3+json" \
              https://api.github.com/repositories?since=467294132

[49]: res = requests.get(
              'https://api.github.com/repositories?since=467294132',
              headers={'Authorization': 'token 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'}
      )

[15]: type(res)

[15]: requests.models.Response

[42]: import json
      repos = json.loads(res.content.decode('utf-8'))

[43]: type(repos)

[43]: list

[73]: %%time
      repos_details = []
```



```
[73]: %%time
      repos_details = []
      for repo in repos:
          try:
              owner = repo['owner']['login']
              name = repo['name']
              rd = json.loads(requests.get(
                  f'https://api.github.com/repos/{owner}/{name}',
                  headers={'Authorization': 'token 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'}
              ).content.decode('utf-8'))
              repo_details = {
                  'id': rd['id'],
                  'node_id': rd['node_id'],
                  'name': rd['name'],
                  'full_name': rd['full_name'],
                  'owner': {
                      'login': rd['owner']['login'],
                       'id': rd['owner']['id'],
                       'node_id': rd['owner']['node_id'],
                       'type': rd['owner']['type'],
                      'site_admin': rd['owner']['site_admin']
                  },
                  'html_url': rd['html_url'],
                  'description': rd['description'],
                  'fork': rd['fork'],
                  'created_at': rd['created_at']
              }
              repos_details.append(repo_details)
          except:
```



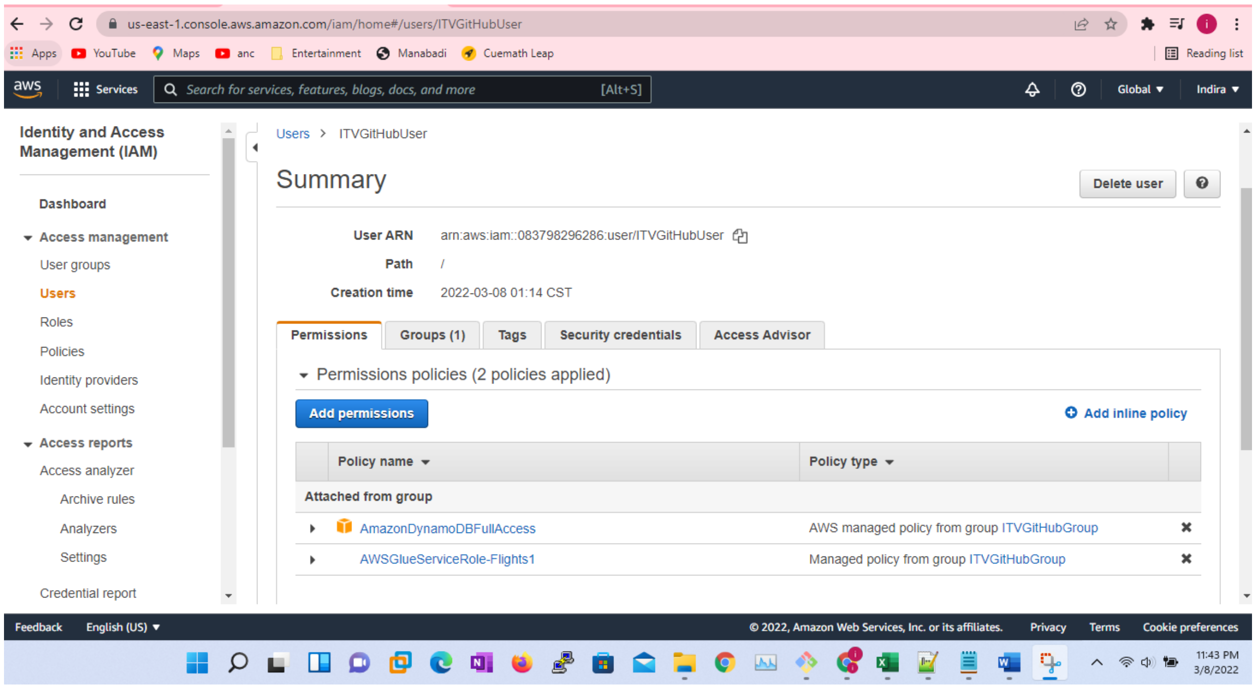
```
[74]: len(repos_details)

[74]: 90

[80]: import requests, json
      def list_repos(token, since='467294132'):
              res = requests.get(
                  f'https://api.github.com/repositories?since={since}',
                  headers={'Authorization': f'token {token}'}
              )
              return json.loads(res.content.decode('utf-8'))

[ ]:  def get_repo_details(owner, name, token):
              repo_details = json.loads(requests.get(
                  f'https://api.github.com/repos/{owner}/{name}',
                  headers={'Authorization': f'token {token}'}
              ).content.decode('utf-8'))
              return repo_details

[81]: repos = list_repos('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')

[82]:  def extract_repo_fields(repo_details):
              repo_fields = {
                  'id': repo_details['id'],
                  'node_id': repo_details['node_id'],
```



```
[85]: def get_repos(repos, token):
              repos_details = []
              for repo in repos:
                  try:
                      owner = repo['owner']['login']
                      name = repo['name']
                      repo_details = get_repo_details(owner, name, token)
                      repo_fields = extract_repo_fields(repo_details)
                      repos_details.append(repo_fields)
                  except:
                      pass
              return repos_details

[88]: repos_details = get_repos(repos, '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')

[87]: repos_details[-1]

[87]: {'id': 467294347,
       'node_id': 'R_kgDOG9pYiw',
       'name': 'openpilot',
       'full_name': 'FrogAi/openpilot',
       'owner': {'login': 'FrogAi',
        'id': 91348155,
        'node_id': 'MDQ6VXNlcjkxMzQ4MTU1',
        'type': 'User',
        'site_admin': False},
       'html_url': 'https://github.com/FrogAi/openpilot',
```

Creating Dynamodb Tables

To get the data from GitHub API into DynamoDB, need to create tables. We can create the table using boto3 and use itvgithub user to take care of tables using boto3. Also attached the required policies to the user.



Summary

User ARN arn:aws:iam::083798296286:user/ITVGitHubUser

Path /

Creation time 2022-03-08 01:14 CST

Permissions policies (2 policies applied)

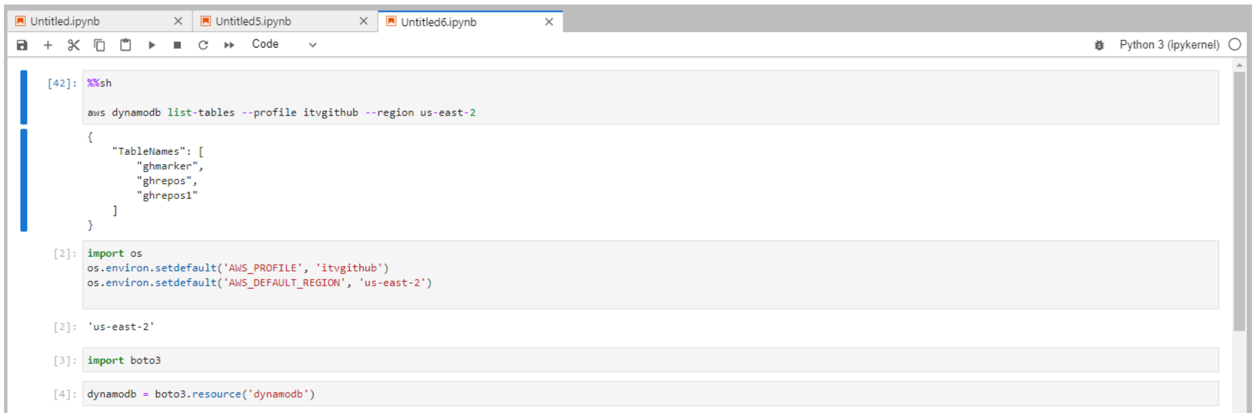
Policy name	Policy type
Attached from group	
AmazonDynamoDBFullAccess	AWS managed policy from group ITVGitHubGroup
AWSGlueServiceRole-Flights1	Managed policy from group ITVGitHubGroup

Create tables for both storing GitHub Repo data as well as the marker or bookmark. Marker or Bookmark will be used to invoke the API and get the data in incremental fashion.

- Create table called as `ghmarker`. It will only contain one record with 3 columns.
- tn (table name - ghrepos)
- marker (last id from each list all repos call). We will store it as string as we can use it for other API calls to populate other tables.
- status (success or failed)

As DynamoDB is NoSQL database, we cannot specify the column names while creating the tables. We specify the column names along with data while loading data into the table.

Connecting to DynamoDB via AWS also configured AWS profile.



```
[42]: %%sh

      aws dynamodb list-tables --profile itvgithub --region us-east-2

      {
          "TableNames": [
              "ghmarker",
              "ghrepos",
              "ghrepos1"
          ]
      }

 [2]: import os
      os.environ.setdefault('AWS_PROFILE', 'itvgithub')
      os.environ.setdefault('AWS_DEFAULT_REGION', 'us-east-2')

 [2]: 'us-east-2'

 [3]: import boto3

 [4]: dynamodb = boto3.resource('dynamodb')
```

Figure 3Connecting to DynamoDB via AWS

Created the table 'ghmaker'

```
ghmarker = dynamodb.create_table(
            TableName='ghmarker',
            KeySchema=[
                {
                    'AttributeName': 'tn',
                    'KeyType': 'HASH'
                },
            ],
            AttributeDefinitions=[
                {
                    'AttributeName': 'tn',
                    'AttributeType': 'S'
                },
            ],
            BillingMode='PAY_PER_REQUEST'
        )
```

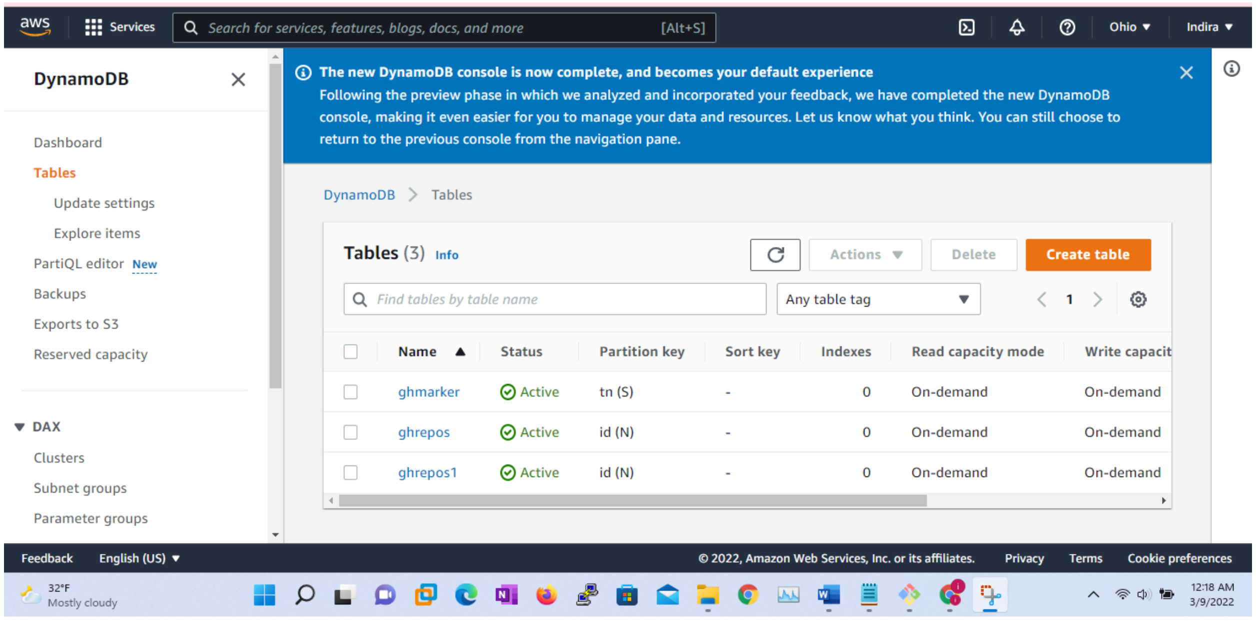
```
ghmarker.table_status
```

```
'CREATING'
```

```
ghr_table=dynamodb.Table('ghmarker')
```

```
ghmarker.table_status
```

Created the table in DynamoDB

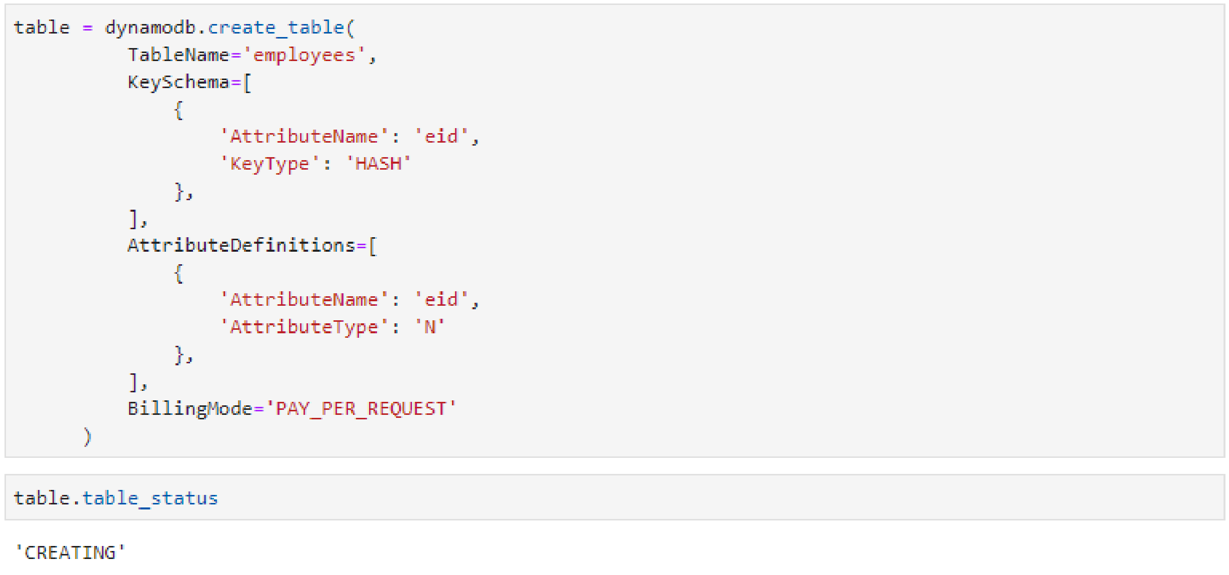


The new DynamoDB console is now complete, and becomes your default experience
Following the preview phase in which we analyzed and incorporated your feedback, we have completed the new DynamoDB console, making it even easier for you to manage your data and resources. Let us know what you think. You can still choose to return to the previous console from the navigation pane.

Tables (3) Info

Name	Status	Partition key	Sort key	Indexes	Read capacity mode	Write capacit
ghmarker	✓ Active	tn (S)	-	0	On-demand	On-demand
ghrepos	✓ Active	id (N)	-	0	On-demand	On-demand
ghrepos1	✓ Active	id (N)	-	0	On-demand	On-demand

Created another table called table with some attributes.

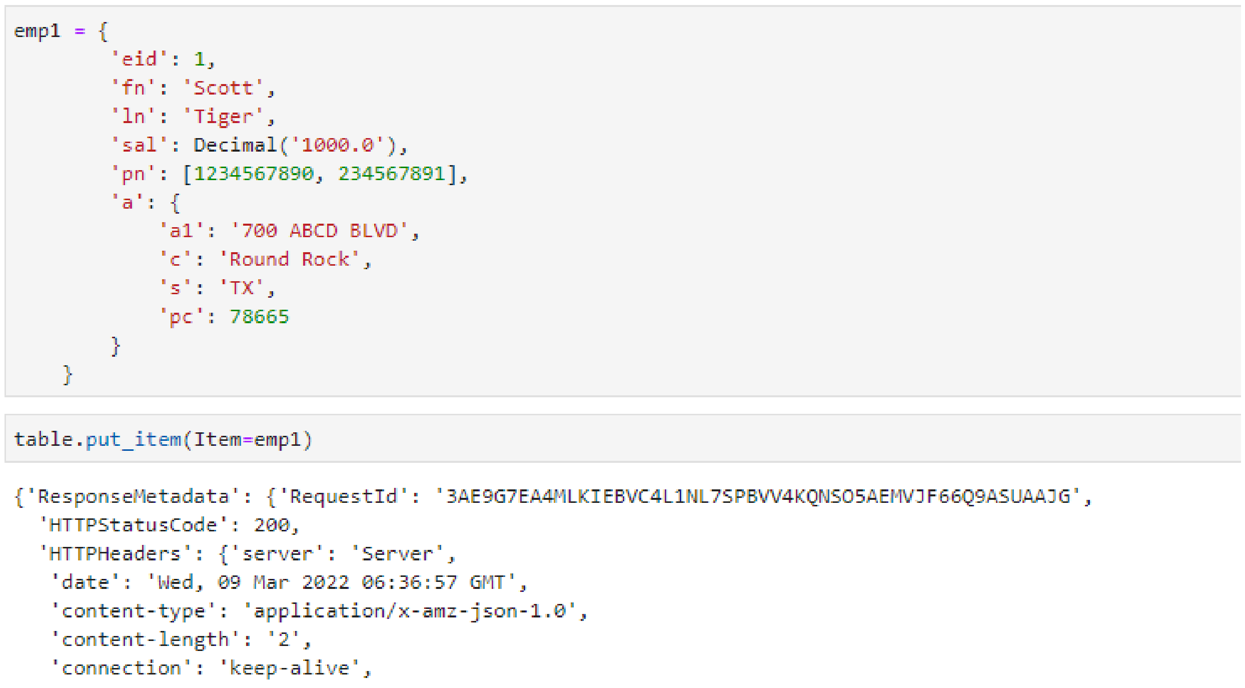


```
table = dynamodb.create_table(
          TableName='employees',
          KeySchema=[
              {
                  'AttributeName': 'eid',
                  'KeyType': 'HASH'
              },
          ],
          AttributeDefinitions=[
              {
                  'AttributeName': 'eid',
                  'AttributeType': 'N'
              },
          ],
          BillingMode='PAY_PER_REQUEST'
      )
```

```
table.table_status
```

```
'CREATING'
```

Insert first record

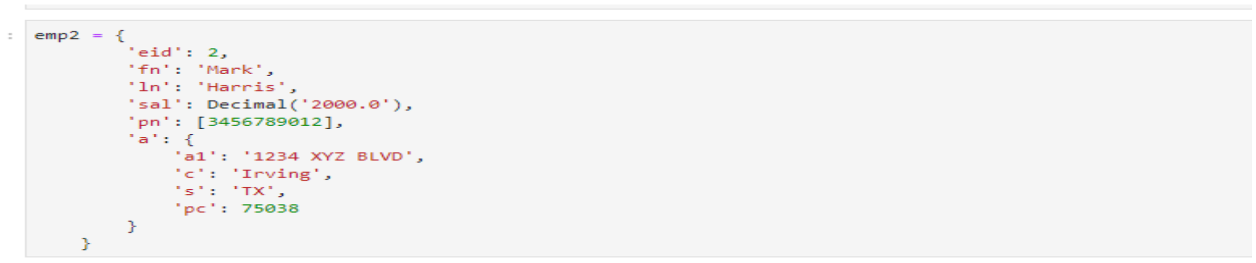


```
emp1 = {
        'eid': 1,
        'fn': 'Scott',
        'ln': 'Tiger',
        'sal': Decimal('1000.0'),
        'pn': [1234567890, 234567891],
        'a': {
            'a1': '700 ABCD BLVD',
            'c': 'Round Rock',
            's': 'TX',
            'pc': 78665
        }
    }
```

```
table.put_item(Item=emp1)
```

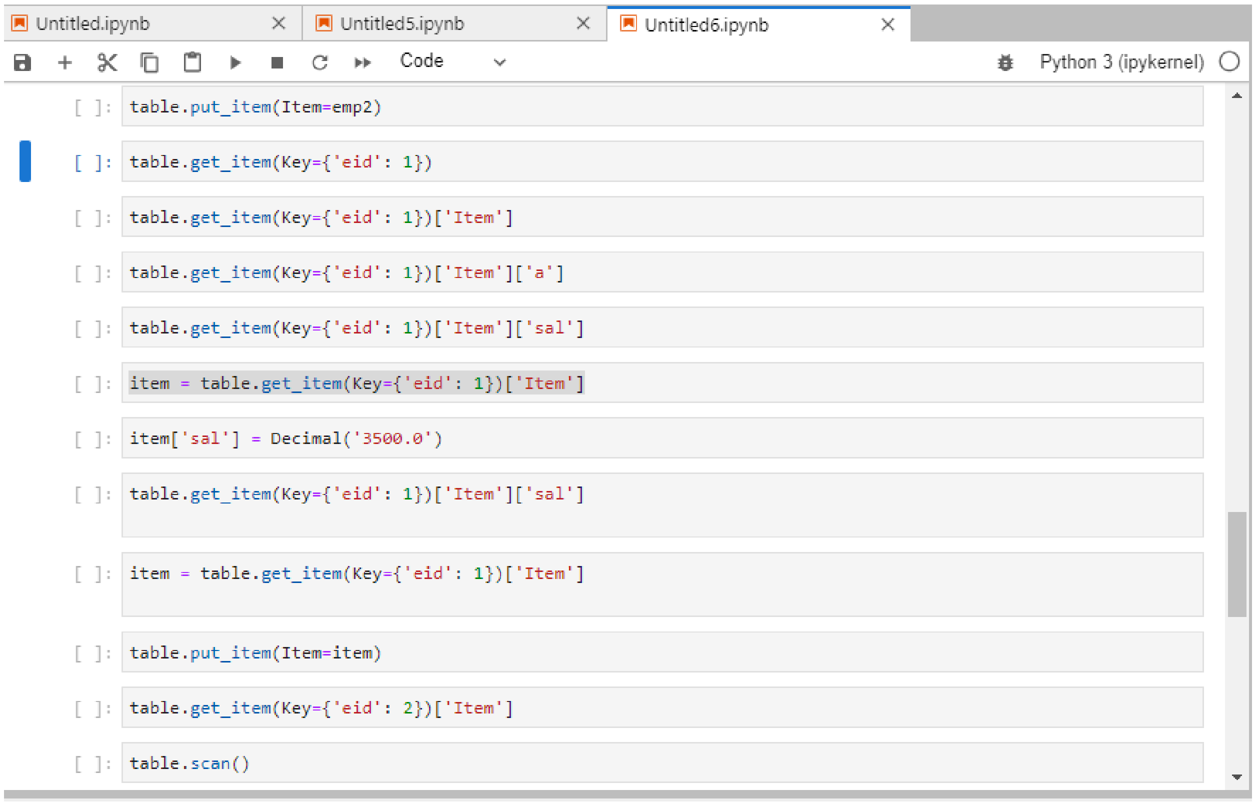
```
{'ResponseMetadata': {'RequestId': '3AE9G7EA4MLKIEBVC4L1NL7SPBVV4KQNSO5AEMVJF66Q9ASUAAJG',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'server': 'Server',
   'date': 'Wed, 09 Mar 2022 06:36:57 GMT',
   'content-type': 'application/x-amz-json-1.0',
   'content-length': '2',
   'connection': 'keep-alive',
```

Inserted the second record:



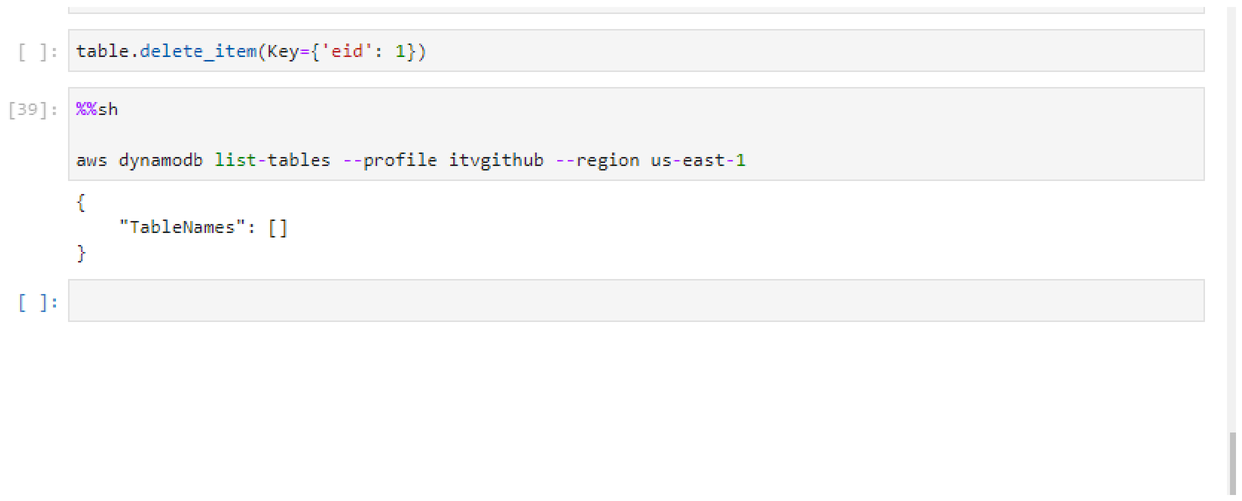
```
: emp2 = {
          'eid': 2,
          'fn': 'Mark',
          'ln': 'Harris',
          'sal': Decimal('2000.0'),
          'pn': [3456789012],
          'a': {
              'a1': '1234 XYZ BLVD',
              'c': 'Irving',
              's': 'TX',
              'pc': 75038
          }
      }
```

Read/insert the data into the records



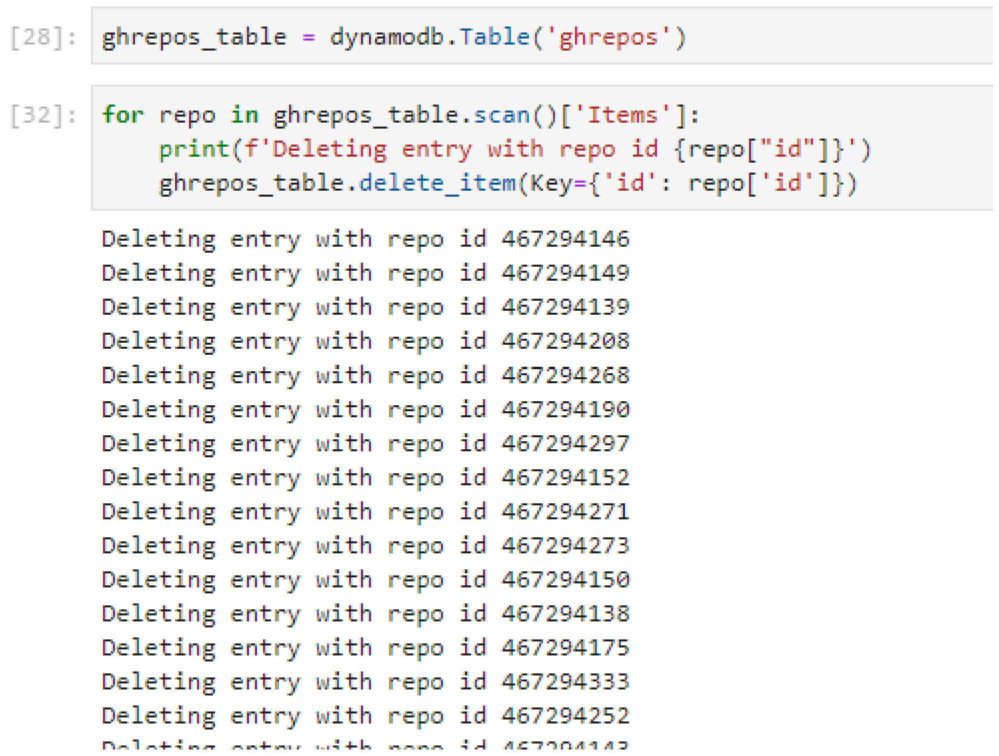
```
[ ]: table.put_item(Item=emp2)

[ ]: table.get_item(Key={'eid': 1})

[ ]: table.get_item(Key={'eid': 1})['Item']

[ ]: table.get_item(Key={'eid': 1})['Item']['a']

[ ]: table.get_item(Key={'eid': 1})['Item']['sal']

[ ]: item = table.get_item(Key={'eid': 1})['Item']

[ ]: item['sal'] = Decimal('3500.0')

[ ]: table.get_item(Key={'eid': 1})['Item']['sal']

[ ]: item = table.get_item(Key={'eid': 1})['Item']

[ ]: table.put_item(Item=item)

[ ]: table.get_item(Key={'eid': 2})['Item']

[ ]: table.scan()
```

Delete the table



```
[ ]: table.delete_item(Key={'eid': 1})

[39]: %%sh

      aws dynamodb list-tables --profile itvgithub --region us-east-1

      {
          "TableNames": []
      }

[ ]:
```

DynamoDB Batch Operation:
We can insert batch data into DynamoDB table using batch writer. Batch writer used to load the data to DynamoDB table in batches. Same way can use it for delete as well.



```
[28]: ghrepos_table = dynamodb.Table('ghrepos')
```

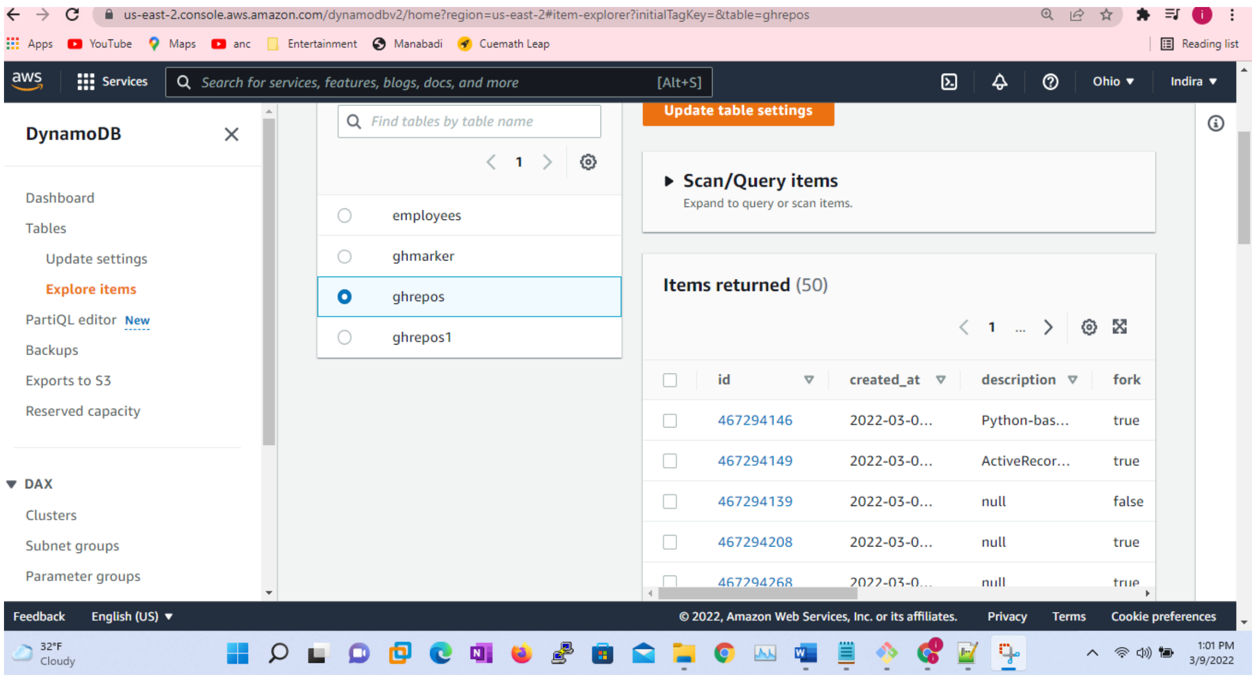
```
[32]: for repo in ghrepos_table.scan()['Items']:
          print(f'Deleting entry with repo id {repo["id"]}')
          ghrepos_table.delete_item(Key={'id': repo['id']})
```

```
Deleting entry with repo id 467294146
Deleting entry with repo id 467294149
Deleting entry with repo id 467294139
Deleting entry with repo id 467294208
Deleting entry with repo id 467294268
Deleting entry with repo id 467294190
Deleting entry with repo id 467294297
Deleting entry with repo id 467294152
Deleting entry with repo id 467294271
Deleting entry with repo id 467294273
Deleting entry with repo id 467294150
Deleting entry with repo id 467294138
Deleting entry with repo id 467294175
Deleting entry with repo id 467294333
Deleting entry with repo id 467294252
```

Insert batch_writer

Insert the batch of repos into the DynamoDB table. Created function for the same as mentioned below.

```
[39]: type(batch_writer)

[39]: boto3.dynamodb.table.BatchWriter

[41]: def load_repos(repos_details, ghrepos_table, batch_size=50):
              with ghrepos_table.batch_writer() as batch:

                  repos_count = len(repos_details)
                  for i in range(0, repos_count, batch_size):
                      print(f'Processing from {i} to {i+batch_size}')
                      for repo in repos_details[i:i+batch_size]:
                          batch.put_item(Item=repo)

[45]: list(range(0, 100, 50))

[45]: [0, 50]

[46]: %%time
      load_repos(repos_details, ghrepos_table)

      Processing from 0 to 50
      Processing from 50 to 100
      Wall time: 614 ms

[ ]:
```



us-east-2.console.aws.amazon.com/dynamodbv2/home?region=us-east-2#item-explorer?initialTagKey=&table=ghrepos

Apps · YouTube · Maps · anc · Entertainment · Manabadi · Cuemath Leap · Reading list
aws · Services · Search for services, features, blogs, docs, and more [Alt+S] · Ohio · Indira

DynamoDB

- Dashboard
- Tables
  - Update settings
  - Explore items
- PartiQL editor New
- Backups
- Exports to S3
- Reserved capacity

DAX

- Clusters
- Subnet groups
- Parameter groups

Find tables by table name

- ( ) employees
- ( ) ghmarker
- (●) ghrepos
- ( ) ghrepos1

Update table settings

Scan/Query items
Expand to query or scan items.

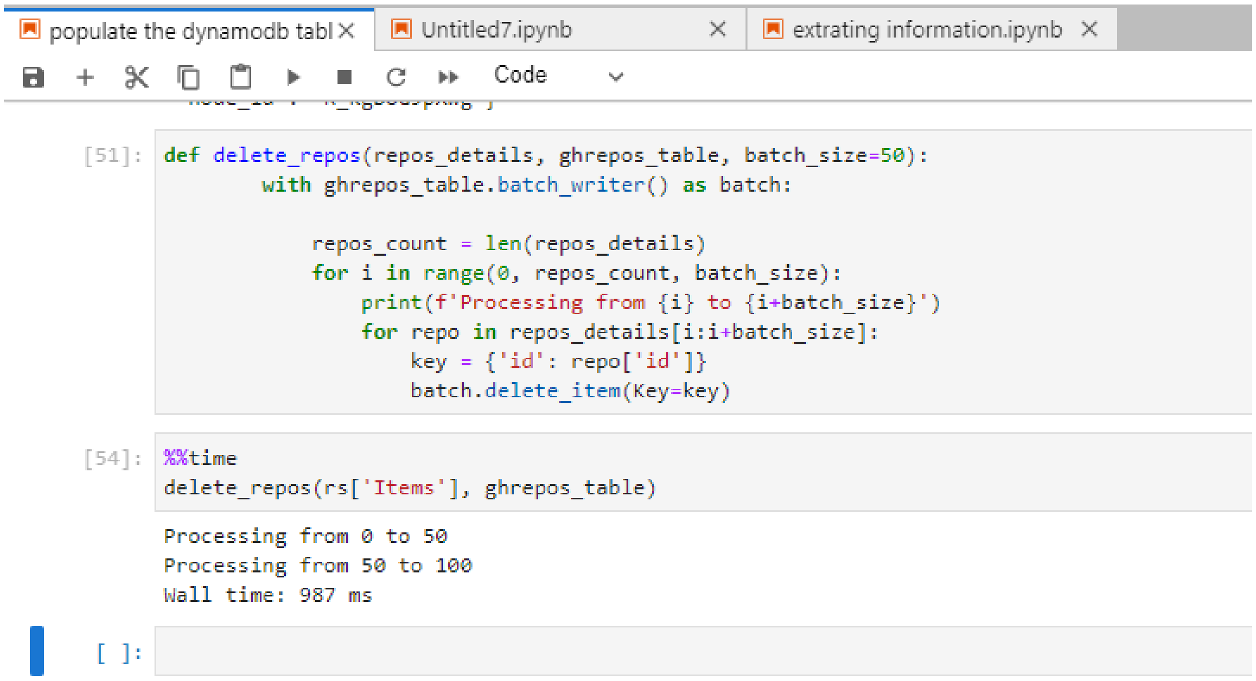
Items returned (50)

id	created_at	description	fork
467294146	2022-03-0...	Python-bas...	true
467294149	2022-03-0...	ActiveRecor...	true
467294139	2022-03-0...	null	false
467294208	2022-03-0...	null	true
467294268	2022-03-0...	null	true

Feedback · English (US)
© 2022, Amazon Web Services, Inc. or its affiliates. · Privacy · Terms · Cookie preferences
32°F Cloudy · 1:01 PM 3/9/2022

Delete batch_writer

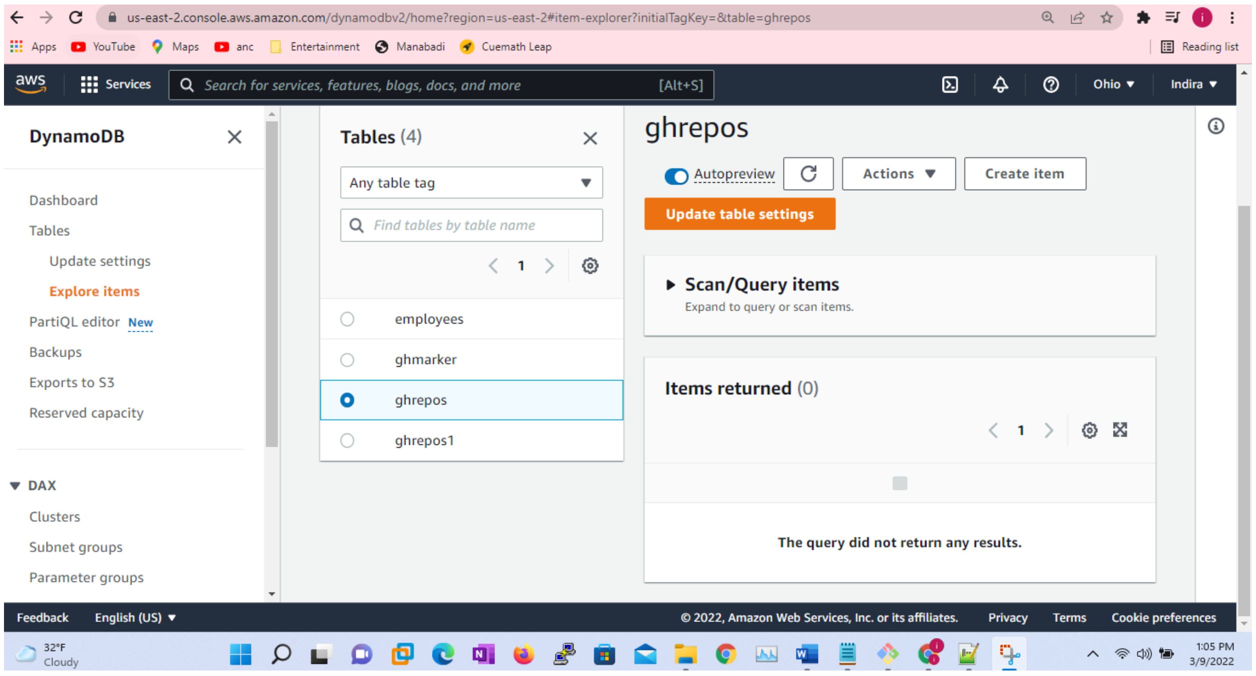
To delete the bath of repos in DynamoDB table



```
[51]: def delete_repos(repos_details, ghrepos_table, batch_size=50):
          with ghrepos_table.batch_writer() as batch:

              repos_count = len(repos_details)
              for i in range(0, repos_count, batch_size):
                  print(f'Processing from {i} to {i+batch_size}')
                  for repo in repos_details[i:i+batch_size]:
                      key = {'id': repo['id']}
                      batch.delete_item(Key=key)
```

```
[54]: %%time
      delete_repos(rs['Items'], ghrepos_table)

      Processing from 0 to 50
      Processing from 50 to 100
      Wall time: 987 ms

[ ]:
```

Validated in DynamoDB Table as shown below



us-east-2.console.aws.amazon.com/dynamodbv2/home?region=us-east-2#item-explorer?initialTagKey=&table=ghrepos