PRACTICAL: 3

AIM:

Scenario:

The café website is up and running, and the café staff noticed a significant increase in new customer visits. Multiple customers also mentioned that it would be helpful if the website had an up-to-date menu. They could then use the menu to check the availability of food items before going to the café. Frank and Martha ask Sofía to explore whether she can implement this feature for customers. Sofía is feeling more confident in her coding skills and has also been learning about different ways to store information in AWS. She knows that before they can dynamically update data on the website, she must first choose a data storage service to hold the data. She also needs to learn how to manage table data, load the product records, and create scripts to retrieve information from the data platform. A business request from the café: Store menu information in the cloud. Frank and Martha mentioned to Sofía that they want the website to dynamically update its menu information. To prepare for this new functionality, Sofía decides to store this information in DynamoDB. Café staff must be able to retrieve information from the table. Sofía decides to create one script that retrieves all inventory items from the table and another script (as a proof of concept) that uses a product name to retrieve a single record. For this first challenge, you take on the role of Sofía. You use the AWS CLI and the SDK for Python to configure and create a DynamoDB table, load records into the table, and extract data from the table.

Lab overview and objectives:

In this lab, you use Amazon DynamoDB to store and manage menu information. Using databases, such as DynamoDB, simplifies data management because you can easily query, sort, edit, and index data. You will use both the AWS Command Line Interface (AWS CLI) and the AWS SDK for Python (Boto3) to work with DynamoDB. In upcoming labs, you will use application programming interface (API) calls from the café website to dynamically retrieve and update data that's stored in a DynamoDB table.

After completing this lab, you should be able to:

- Create a new DynamoDB table.
- Add data to the table.
- Modify table items based on conditions.
- Query the table.
- Add a global secondary index to the table.

THEORY:

Amazon DynamoDB is a fast, scalable NoSQL database service ideal for dynamically managing the café's menu information. By creating a table with a defined primary key, the menu items can be stored with attributes like name, price, category, and availability. Data can be added using PutItem, modified with conditional updates using UpdateItem, and queried efficiently using Query or Scan. Adding a Global Secondary Index (GSI) allows fast searches on non-primary key attributes, like categories. With DynamoDB's scalability, low-latency access, and seamless API integration, the café can dynamically update its menu, improving customer experience and operational efficiency.

CODE:

Not_an_existing_product.json

```
{
        "product_name": {
           "S": "best pipe"
        },
        "product_id": {
           "S": "676767676767"
        }
}
```

Not_an_existing_product.json

```
{
        "product_name": {
           "S": "best pipe"
        },
        "product_id": {
           "S": "3333333333"
        }
}
```

Not_an_existing_product.json

```
{
        "product_name": {
           "S": "best pipe"
        },
        "product_id": {
            "S": "2222222222"
        }
}
```

Conditional_put.py

```
import boto3
from botocore.exceptions import ClientError
def conditional_put():
    DDB = boto3.client('dynamodb', region_name='us-east-1')
    try:
        response = DDB.put_item(
            TableName='FoodProducts',
            Item={
                'product_name': {
                    'S': 'apple pie'
                },
                'product_id': {
                    'S': 'a444'
                },
                'price_in_cents':{
                    'N': '595' #number passed in as a string (ie in quotes)
                },
                'description':{
                    'S': "It is amazing!"
                },
                'tags':{
                    'L': [{
                        'S': 'whole pie'
                    },{
                        'S': 'apple'
                    }]
                }
            },
            ConditionExpression='attribute_not_exists(product_name)'
        )
    except ClientError as e:
        # Ignore the ConditionalCheckFailedException, bubble up
        # other exceptions.
        if e.response['Error']['Code'] != 'ConditionalCheckFailedException':
            raise
if __name__ == '__main__':
    conditional_put()
    print('Done')
```

test_batch_put.py

```
import boto3, json
def batch_put(food_list):
  DDB = boto3.resource('dynamodb', region_name='us-east-1')
  table = DDB.Table('FoodProducts')
  with table.batch_writer(overwrite_by_pkeys=['product_name']) as batch:
    for food in food_list:
            product_name = food['product_name_str']
            price_in_cents = food['price_in_cents_int']
            formatted_item = {
                'product_name': product_name,
                'price_in_cents': price_in_cents  #Boto will "know" this is a number type
            }
            print("Adding food item:", formatted_item)
            batch.put_item(Item=formatted_item)
if __name__ == '__main__':
    with open("../resources/test.json") as json_file:
        food_list = json.load(json_file)
    batch_put(food_list)
```

batch_put.py

```
import boto3, json
def batch_put(food_list):
    DDB = boto3.resource('dynamodb', region_name='us-east-1')
    table = DDB.Table('FoodProducts')
    with table.batch_writer() as batch:
        for food in food_list:
            product_name = food['product_name_str']
            product_id = food['product_id_str']
            price_in_cents = food['price_in_cents_int']
            description = food['description_str']
```

```
          tags = food['tag_str_arr']
          formatted_data  = {
             'product_name': product_name,
             'product_id': product_id,
             'price_in_cents': price_in_cents,
             'description': description,
             'tags': tags
          }
          if 'special_int' in food:
             formatted_data['special'] = food['special_int']
             print("Adding special food item:", product_name, price_in_cents)
          else:
             print("Adding food item:", product_name, price_in_cents)
             pass
          batch.put_item(Item=formatted_data)
if __name__ == '__main__':
   with open("../resources/website/all_products.json") as json_file:
      food_list = json.load(json_file)['product_item_arr']
   batch_put(food_list)#
```

get_all_items.py

```
import boto3
def get_all_items():
   import boto3
```

```
    DDB = boto3.resource('dynamodb', region_name='us-east-1')
    table = DDB.Table('FoodProducts')
    response = table.scan()
    data = response['Items']
    while response.get('LastEvaluatedKey'):
        response = table.scan(ExclusiveStartKey=response['LastEvaluatedKey'])
        data.extend(response['Items'])
    print (data)
if __name__ == '__main__':
    get_all_items()
```

get_one_items.py

```
import boto3, json
from boto3.dynamodb.conditions import Key
def get_one_item(product):
    DDB = boto3.client('dynamodb', region_name='us-east-1')
    response = DDB.get_item(TableName='FoodProducts',
        Key={
         'product_name': {'S': product}
         }
        )
    data = response['Item']
    print (data)
```

```
if __name__ == '__main__':
    product = "chocolate cake"
    get_one_item(product)
```

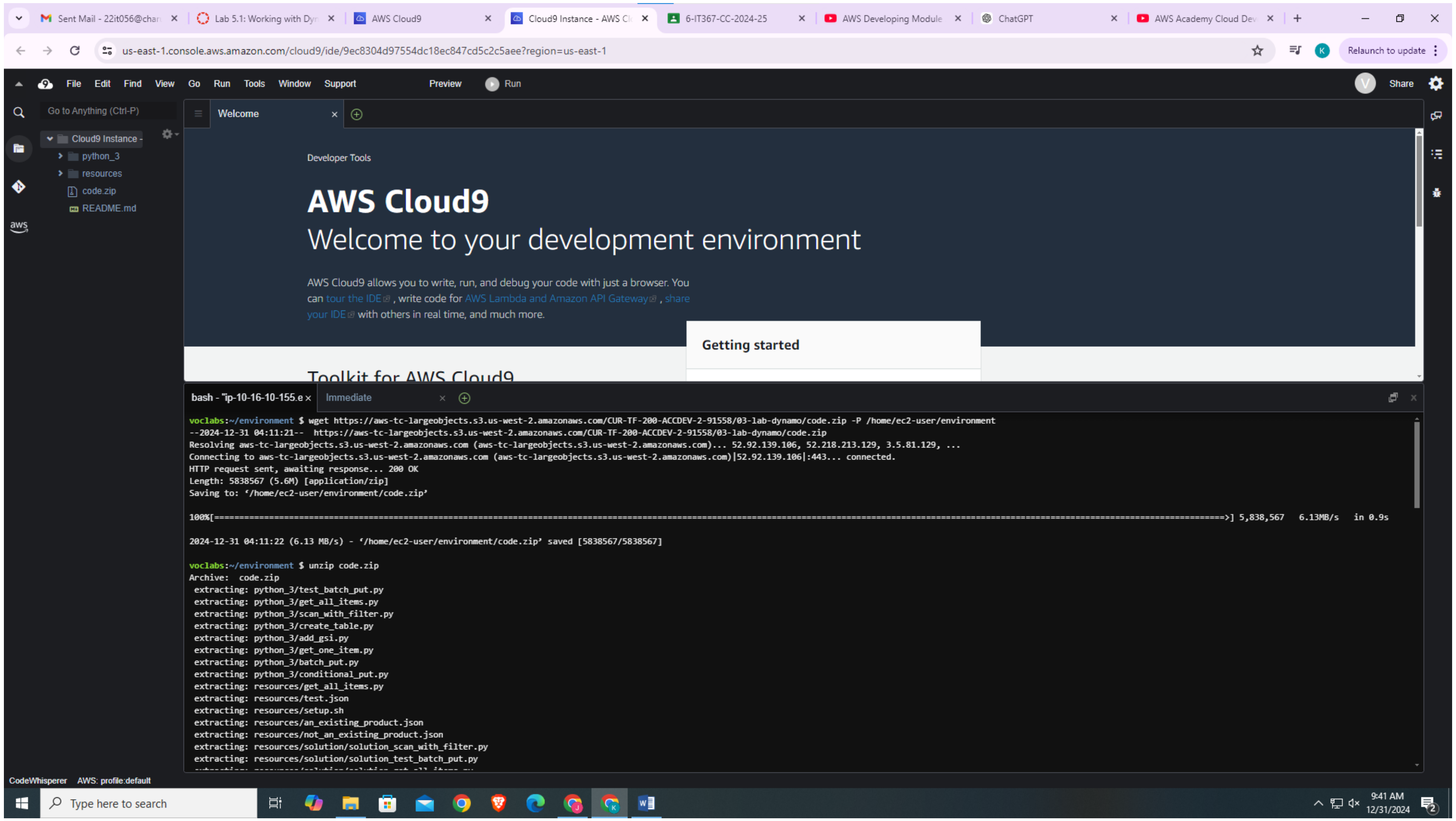
add_gsi.py

```
import boto3
from boto3.dynamodb.conditions import Key
def update_table():
    DDB = boto3.client('dynamodb', region_name='us-east-1')
    params = {
        'TableName': 'FoodProducts',
        'AttributeDefinitions': [
            {'AttributeName': 'special', 'AttributeType': 'N'}
        ],
        'GlobalSecondaryIndexUpdates': [
            {
                'Create': {
                    'IndexName': 'special_GSI',
                    'KeySchema': [
                        {
                            'AttributeName': 'special',
                            'KeyType': 'HASH'
                        }
                    ],
                    'Projection': {
                    'ProjectionType': 'ALL'
                 },
                    'ProvisionedThroughput': {
                    'ReadCapacityUnits': 1,
                    'WriteCapacityUnits': 1
                 }
              }
           }
       ]
    }
    table = DDB.update_table(**params)
    print ('Done')
if __name__ == '__main__':
    update_table()
```

scan_with_filter.py

```
import boto3, json
from boto3.dynamodb.conditions import Key
from boto3.dynamodb.conditions import Key, Attr, Not
def scan_menu_items():
    DDB = boto3.resource('dynamodb', region_name='us-east-1')
    table = DDB.Table('FoodProducts')
    response = table.scan(
                IndexName='special_GSI',
                FilterExpression=Not(Attr('tags').contains('out of stock')))
    data = response['Items']
    print (data)
if __name__ == '__main__':
    scan_menu_items()
```

OUTPUT:



Developer Tools

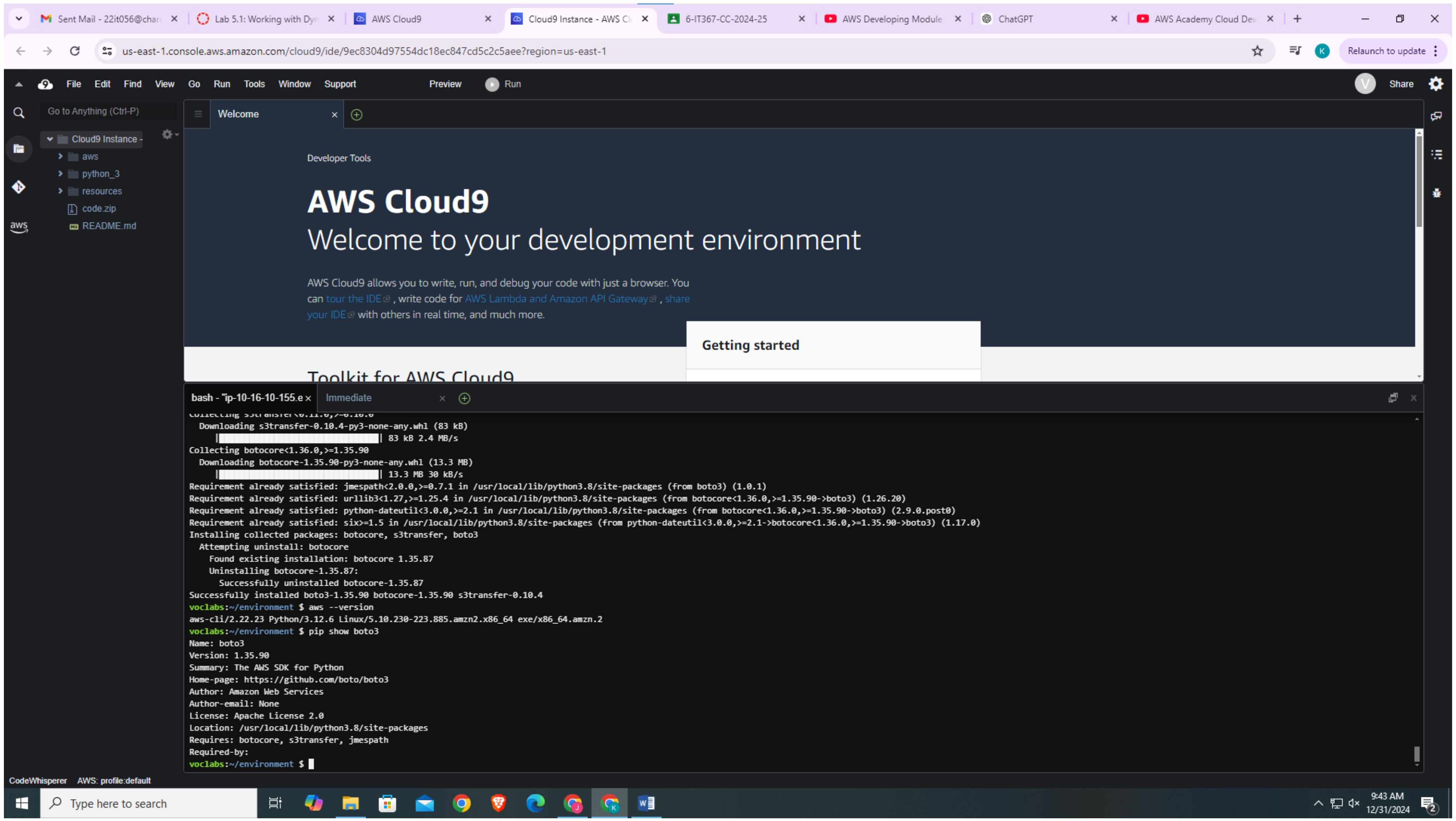
AWS Cloud9

Welcome to your development environment

AWS Cloud9 allows you to write, run, and debug your code with just a browser. You can tour the IDE , write code for AWS Lambda and Amazon API Gateway , share your IDE with others in real time, and much more.

```
voclabs:~/environment $ wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/03-lab-dynamo/code.zip -P /home/ec2-user/environment
--2024-12-31 04:11:21--  https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/03-lab-dynamo/code.zip
Resolving aws-tc-largeobjects.s3.us-west-2.amazonaws.com (aws-tc-largeobjects.s3.us-west-2.amazonaws.com)... 52.92.139.106, 52.218.213.129, 3.5.81.129, ...
Connecting to aws-tc-largeobjects.s3.us-west-2.amazonaws.com (aws-tc-largeobjects.s3.us-west-2.amazonaws.com)|52.92.139.106|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5838567 (5.6M) [application/zip]
Saving to: ‘/home/ec2-user/environment/code.zip’

100%[=====================================================================================================================================================>] 5,838,567   6.13MB/s   in 0.9s

2024-12-31 04:11:22 (6.13 MB/s) - ‘/home/ec2-user/environment/code.zip’ saved [5838567/5838567]

voclabs:~/environment $ unzip code.zip
Archive:  code.zip
 extracting: python_3/test_batch_put.py
 extracting: python_3/get_all_items.py
 extracting: python_3/scan_with_filter.py
 extracting: python_3/create_table.py
 extracting: python_3/add_gsi.py
 extracting: python_3/get_one_item.py
 extracting: python_3/batch_put.py
 extracting: python_3/conditional_put.py
 extracting: resources/get_all_items.py
 extracting: resources/test.json
 extracting: resources/setup.sh
 extracting: resources/an_existing_product.json
 extracting: resources/not_an_existing_product.json
 extracting: resources/solution/solution_scan_with_filter.py
 extracting: resources/solution/solution_test_batch_put.py
```

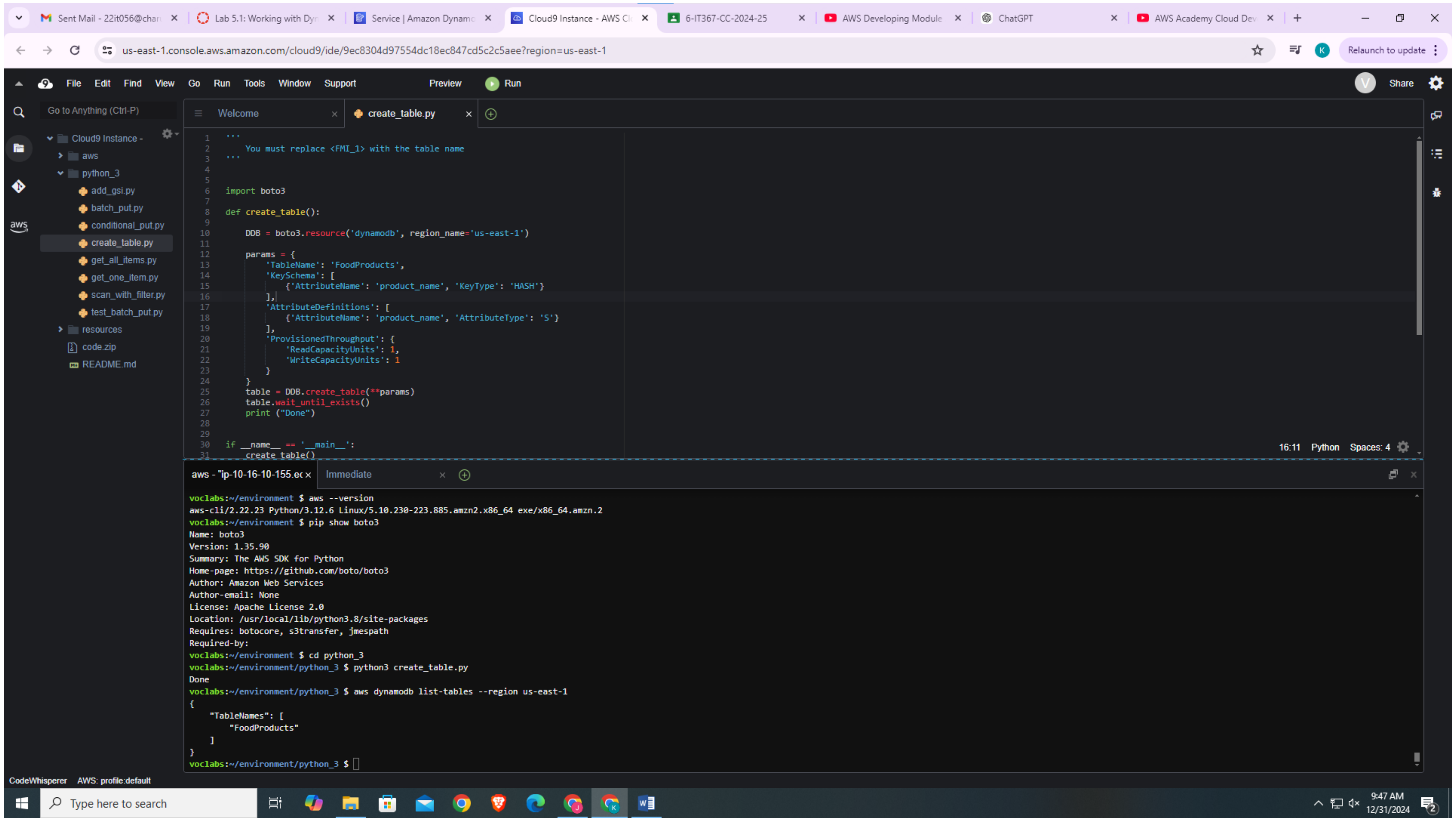
Figure 1: Download and extract zip file



Developer Tools
AWS Cloud9
Welcome to your development environment
AWS Cloud9 allows you to write, run, and debug your code with just a browser. You can tour the IDE, write code for AWS Lambda and Amazon API Gateway, share your IDE with others in real time, and much more.
Getting started
Toolkit for AWS Cloud9

```
  Downloading s3transfer-0.10.4-py3-none-any.whl (83 kB)
     |████████████████████████████████| 83 kB 2.4 MB/s
Collecting botocore<1.36.0,>=1.35.90
  Downloading botocore-1.35.90-py3-none-any.whl (13.3 MB)
     |████████████████████████████████| 13.3 MB 30 kB/s
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/local/lib/python3.8/site-packages (from boto3) (1.0.1)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/local/lib/python3.8/site-packages (from botocore<1.36.0,>=1.35.90->boto3) (1.26.20)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/local/lib/python3.8/site-packages (from botocore<1.36.0,>=1.35.90->boto3) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.36.0,>=1.35.90->boto3) (1.17.0)
Installing collected packages: botocore, s3transfer, boto3
  Attempting uninstall: botocore
    Found existing installation: botocore 1.35.87
    Uninstalling botocore-1.35.87:
      Successfully uninstalled botocore-1.35.87
Successfully installed boto3-1.35.90 botocore-1.35.90 s3transfer-0.10.4
voclabs:~/environment $ aws --version
aws-cli/2.22.23 Python/3.12.6 Linux/5.10.230-223.885.amzn2.x86_64 exe/x86_64.amzn.2
voclabs:~/environment $ pip show boto3
Name: boto3
Version: 1.35.90
Summary: The AWS SDK for Python
Home-page: https://github.com/boto/boto3
Author: Amazon Web Services
Author-email: None
License: Apache License 2.0
Location: /usr/local/lib/python3.8/site-packages
Requires: botocore, s3transfer, jmespath
Required-by:
voclabs:~/environment $
```

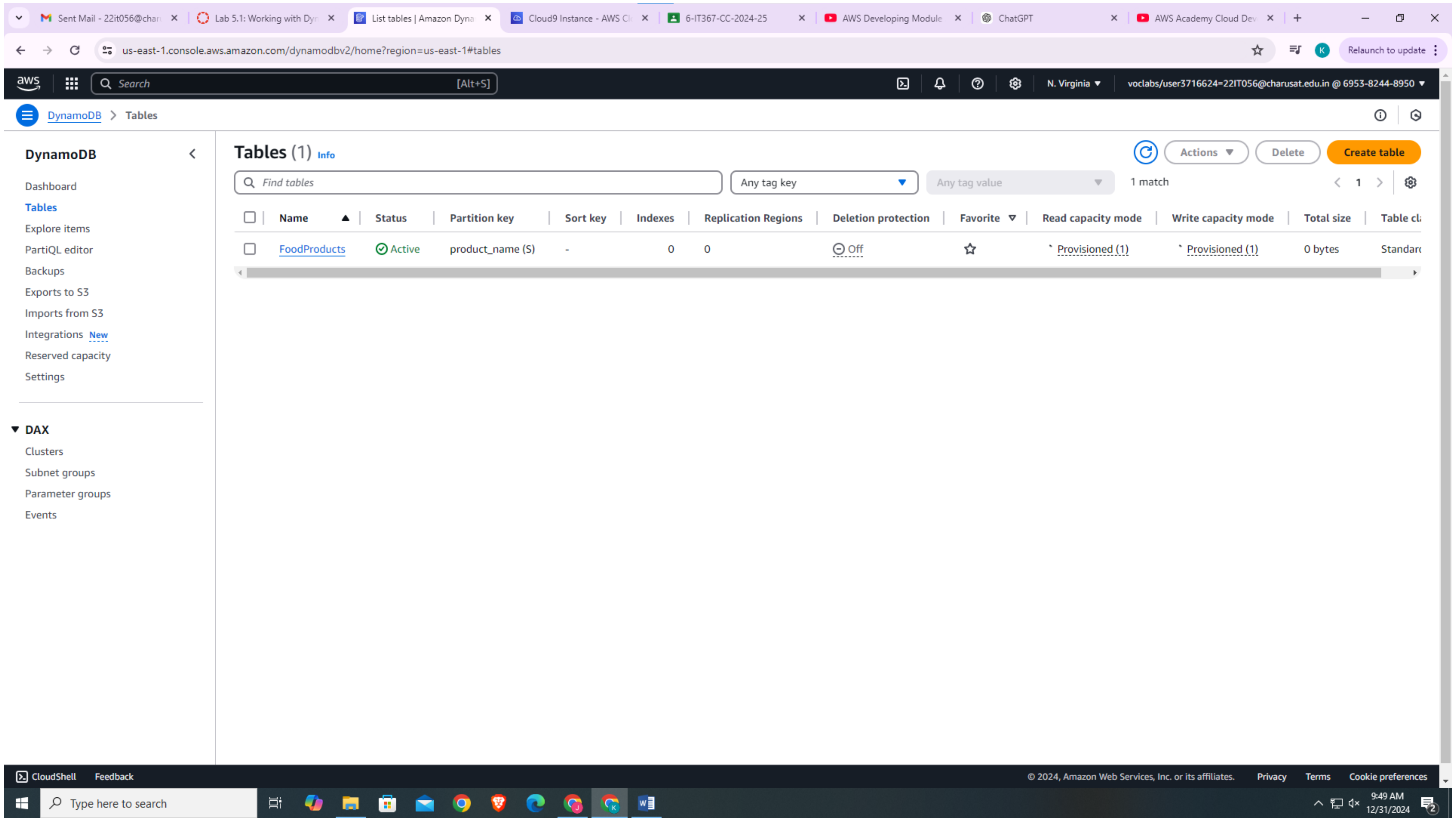
Figure 2: Set python credentials and show boto3



```python
'''
    You must replace <FMI_1> with the table name
'''


import boto3

def create_table():

    DDB = boto3.resource('dynamodb', region_name='us-east-1')

    params = {
        'TableName': 'FoodProducts',
        'KeySchema': [
            {'AttributeName': 'product_name', 'KeyType': 'HASH'}
        ],
        'AttributeDefinitions': [
            {'AttributeName': 'product_name', 'AttributeType': 'S'}
        ],
        'ProvisionedThroughput': {
            'ReadCapacityUnits': 1,
            'WriteCapacityUnits': 1
        }
    }
    table = DDB.create_table(**params)
    table.wait_until_exists()
    print ("Done")


if __name__ == '__main__':
    create_table()
```

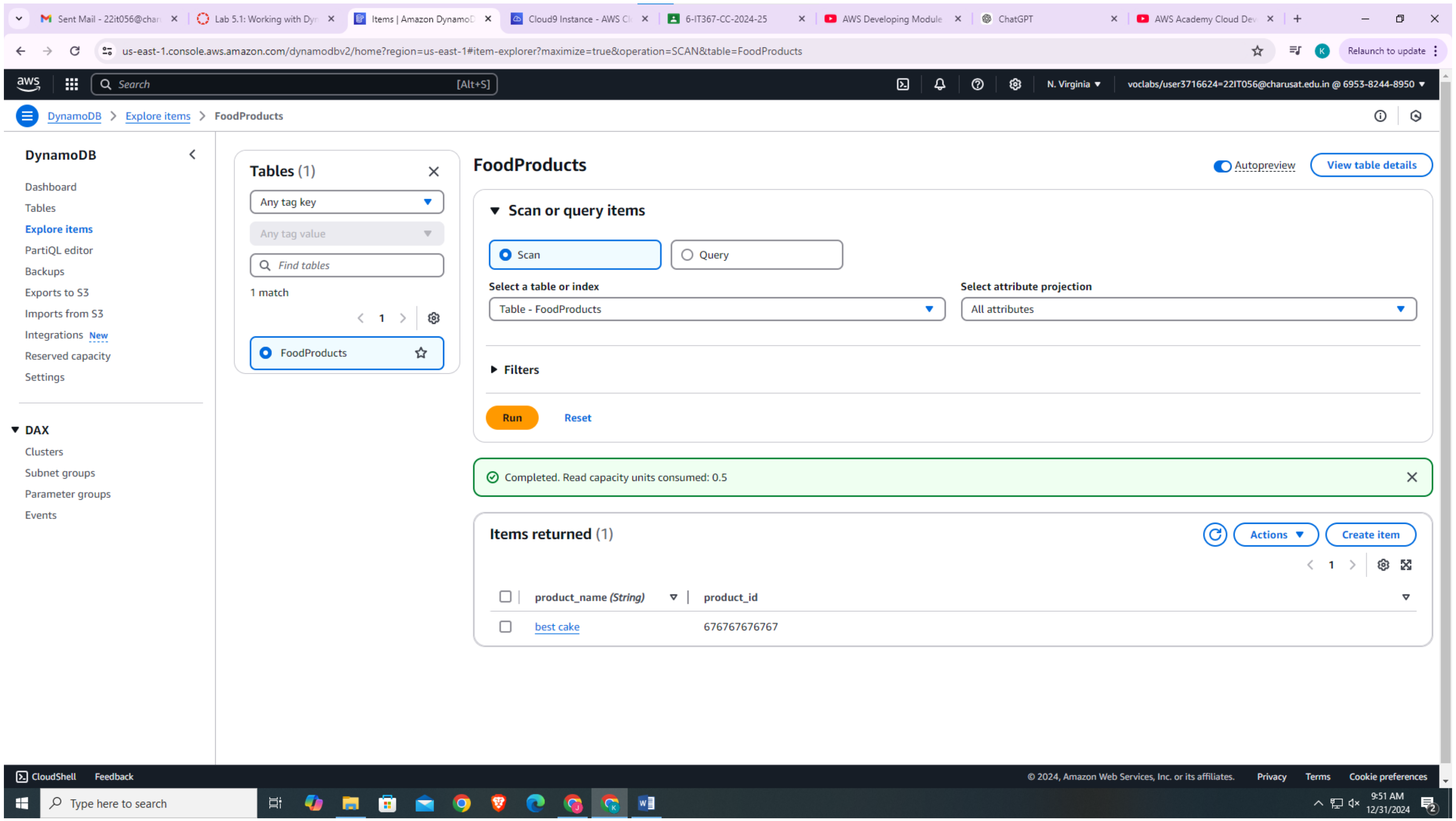
```
voclabs:~/environment $ aws --version
aws-cli/2.22.23 Python/3.12.6 Linux/5.10.230-223.885.amzn2.x86_64 exe/x86_64.amzn.2
voclabs:~/environment $ pip show boto3
Name: boto3
Version: 1.35.90
Summary: The AWS SDK for Python
Home-page: https://github.com/boto/boto3
Author: Amazon Web Services
Author-email: None
License: Apache License 2.0
Location: /usr/local/lib/python3.8/site-packages
Requires: botocore, s3transfer, jmespath
Required-by:
voclabs:~/environment $ cd python_3
voclabs:~/environment/python_3 $ python3 create_table.py
Done
voclabs:~/environment/python_3 $ aws dynamodb list-tables --region us-east-1
{
    "TableNames": [
        "FoodProducts"
    ]
}
voclabs:~/environment/python_3 $
```

Figure 3: Create table and show table data in terminal



Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacity mode	Write capacity mode	Total size	Table cl
FoodProducts	Active	product_name (S)	-	0	0	Off	☆	Provisioned (1)	Provisioned (1)	0 bytes	Standar

Figure 4: Show created table in Dynamodb



us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#item-explorer?maximize=true&operation=SCAN&table=FoodProducts

DynamoDB > Explore items > FoodProducts

FoodProducts

Scan or query items

Select a table or index: Table - FoodProducts

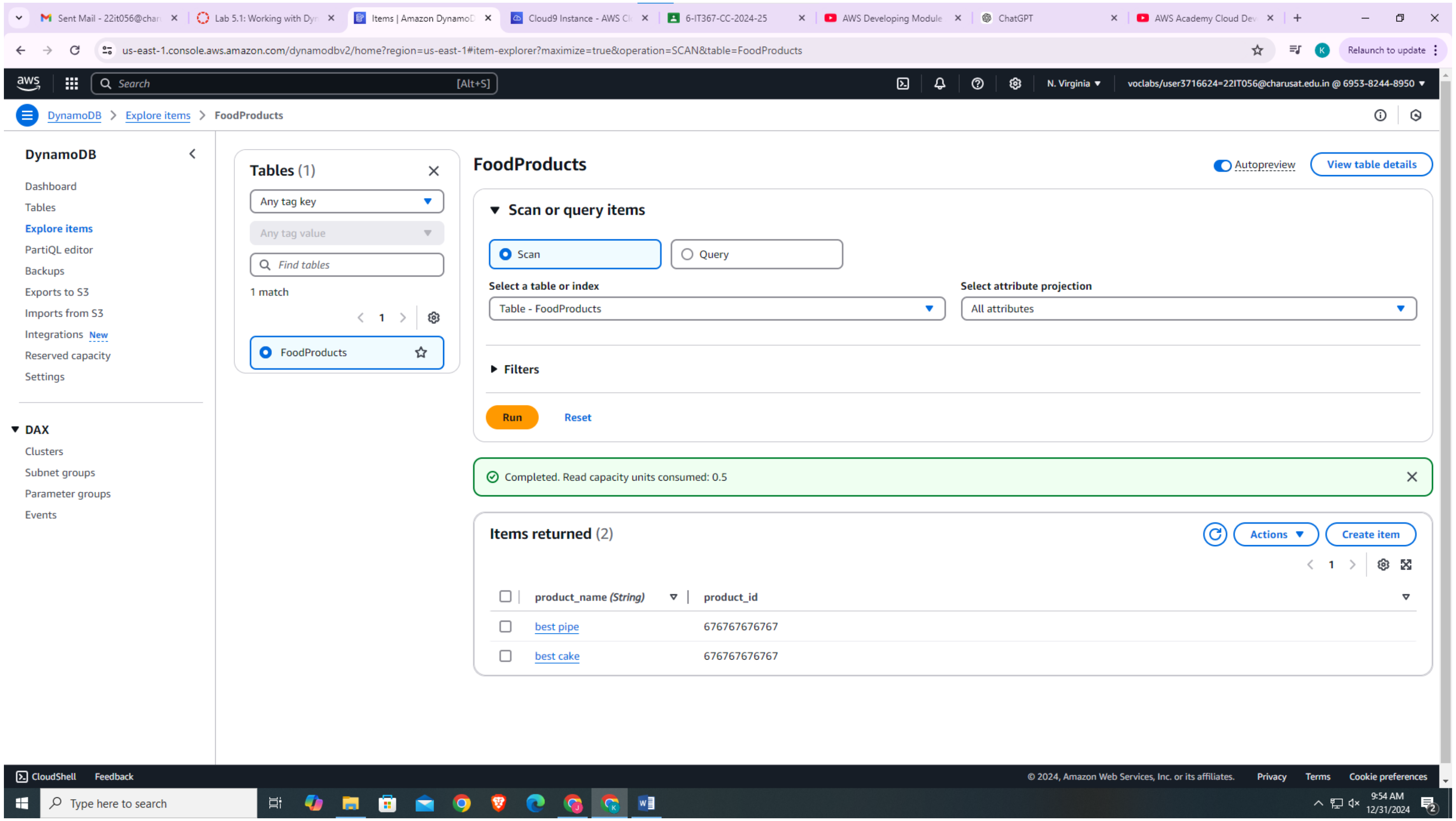
Select attribute projection: All attributes

Completed. Read capacity units consumed: 0.5

Items returned (1)

product_name (String)	product_id
best cake	676767676767

Figure 5: Show data in created table



Select a table or index: Table - FoodProducts

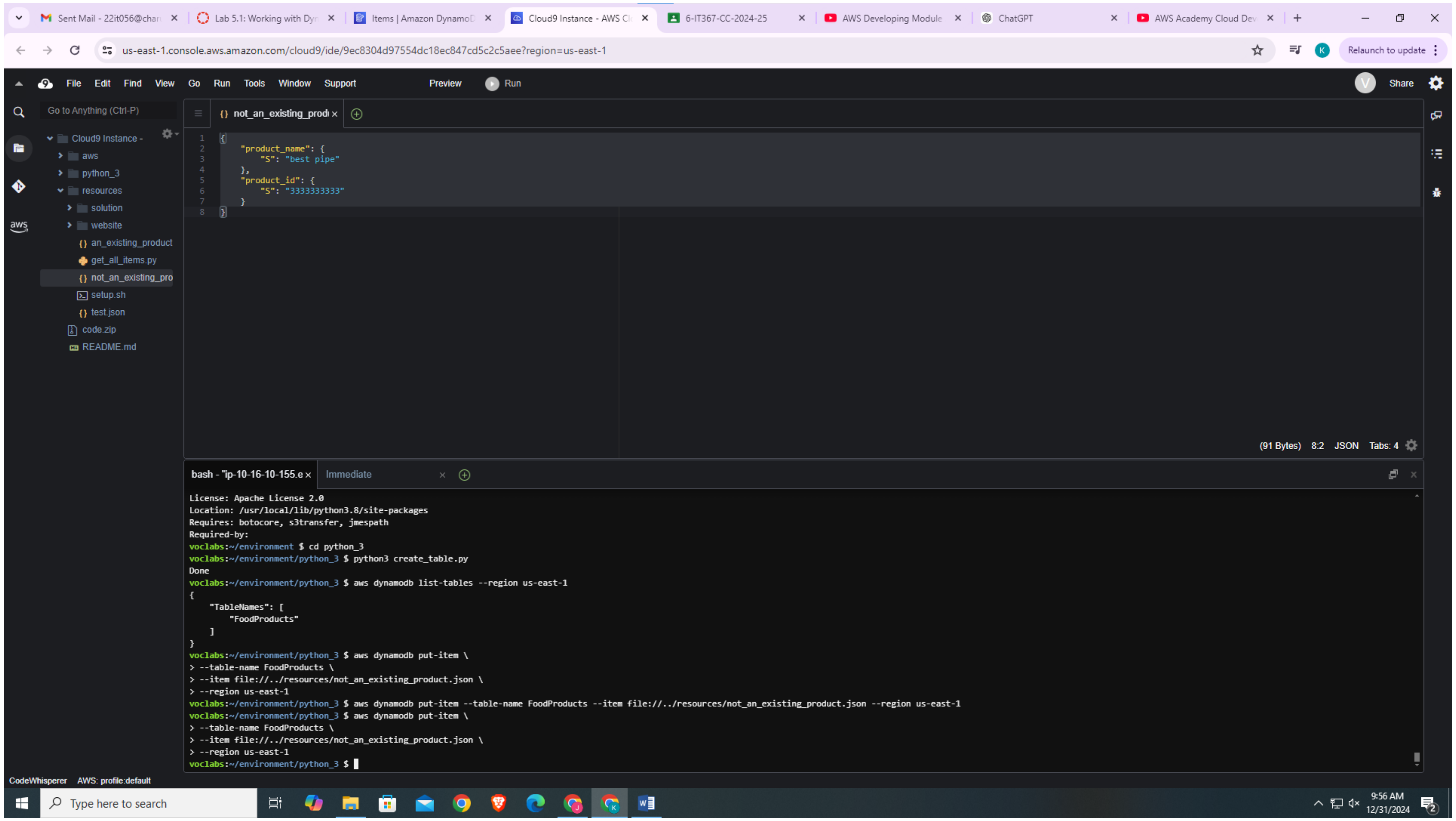
Select attribute projection: All attributes

Completed. Read capacity units consumed: 0.5

Items returned (2)

product_name (String)	product_id
best pipe	676767676767
best cake	676767676767

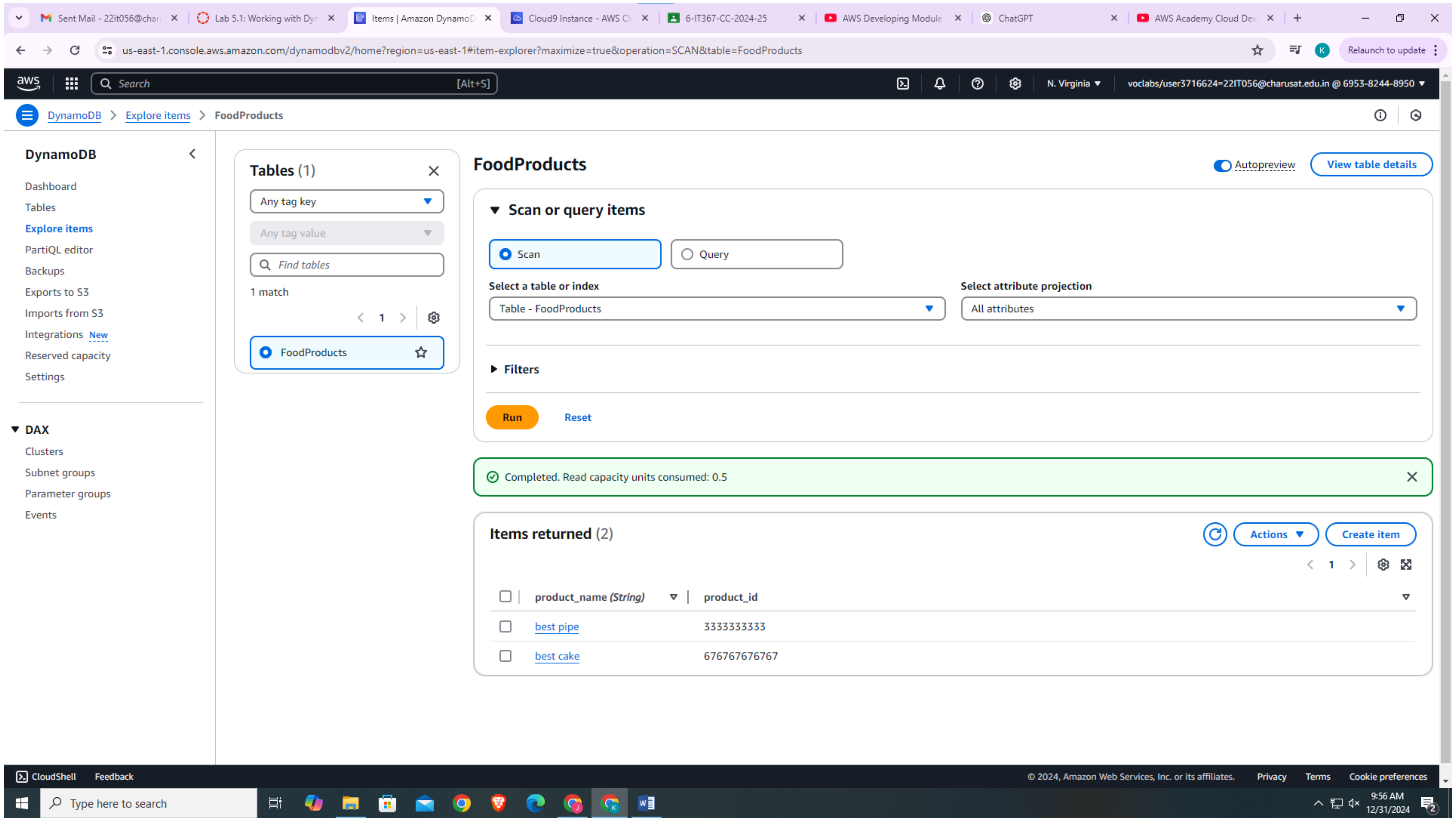
Figure 6: Add other data in table



```
{
    "product_name": {
        "S": "best pipe"
    },
    "product_id": {
        "S": "3333333333"
    }
}
```

```
License: Apache License 2.0
Location: /usr/local/lib/python3.8/site-packages
Requires: botocore, s3transfer, jmespath
Required-by:
voclabs:~/environment $ cd python_3
voclabs:~/environment/python_3 $ python3 create_table.py
Done
voclabs:~/environment/python_3 $ aws dynamodb list-tables --region us-east-1
{
    "TableNames": [
        "FoodProducts"
    ]
}
voclabs:~/environment/python_3 $ aws dynamodb put-item \
> --table-name FoodProducts \
> --item file://../resources/not_an_existing_product.json \
> --region us-east-1
voclabs:~/environment/python_3 $ aws dynamodb put-item --table-name FoodProducts --item file://../resources/not_an_existing_product.json --region us-east-1
voclabs:~/environment/python_3 $ aws dynamodb put-item \
> --table-name FoodProducts \
> --item file://../resources/not_an_existing_product.json \
> --region us-east-1
voclabs:~/environment/python_3 $ 
```

Figure 7: Change product name in .json file.



FoodProducts

Scan or query items

Scan [✓] Query

Select a table or index: Table - FoodProducts

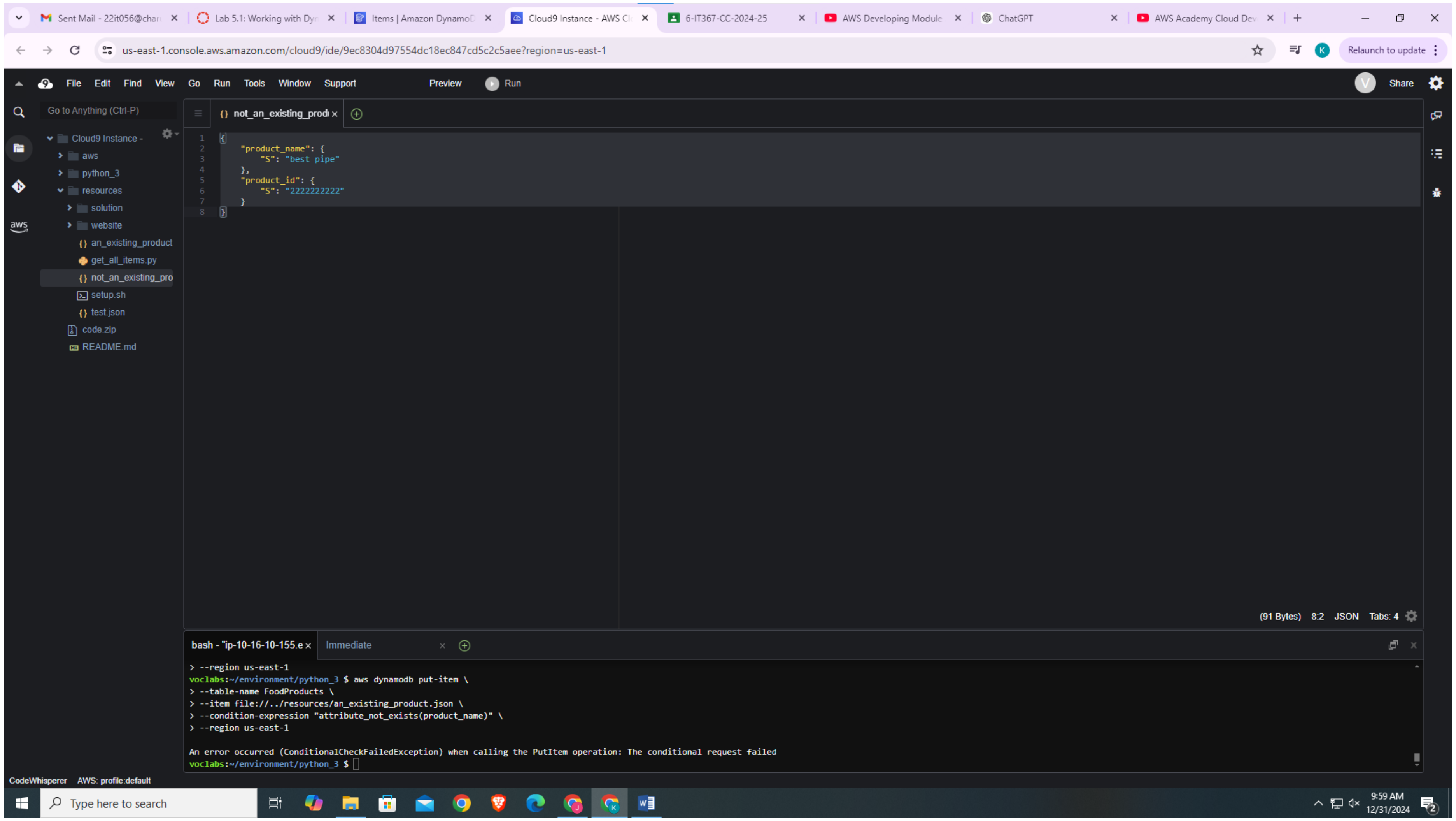
Select attribute projection: All attributes

Completed. Read capacity units consumed: 0.5

Items returned (2)

product_name (String)	product_id
best pipe	3333333333
best cake	676767676767

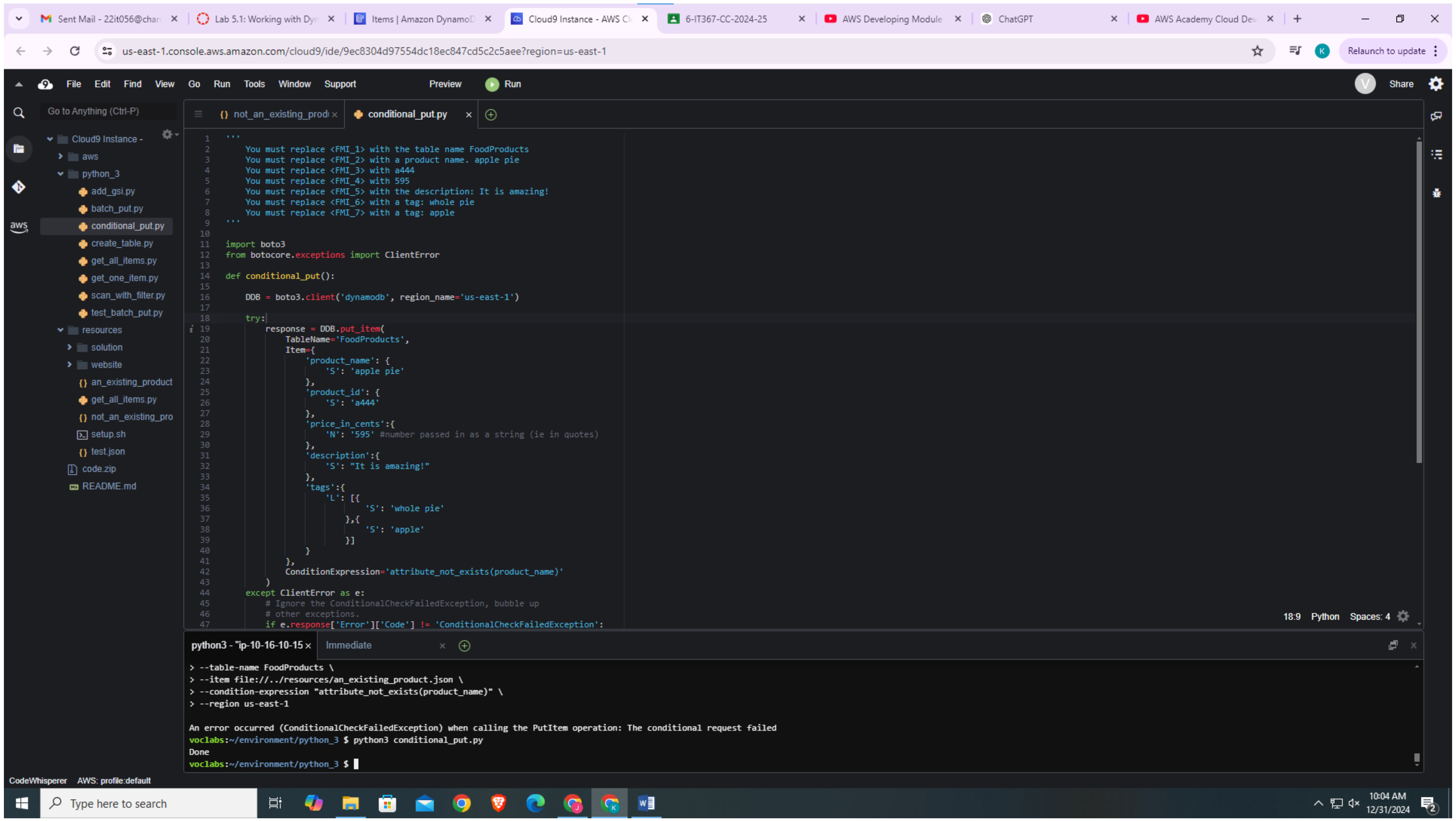
Figure 8: New record is inserted successfully.



```
{
    "product_name": {
        "S": "best pipe"
    },
    "product_id": {
        "S": "2222222222"
    }
}
```

```
> --region us-east-1
voclabs:~/environment/python_3 $ aws dynamodb put-item \
> --table-name FoodProducts \
> --item file://../resources/an_existing_product.json \
> --condition-expression "attribute_not_exists(product_name)" \
> --region us-east-1

An error occurred (ConditionalCheckFailedException) when calling the PutItem operation: The conditional request failed
voclabs:~/environment/python_3 $ 
```

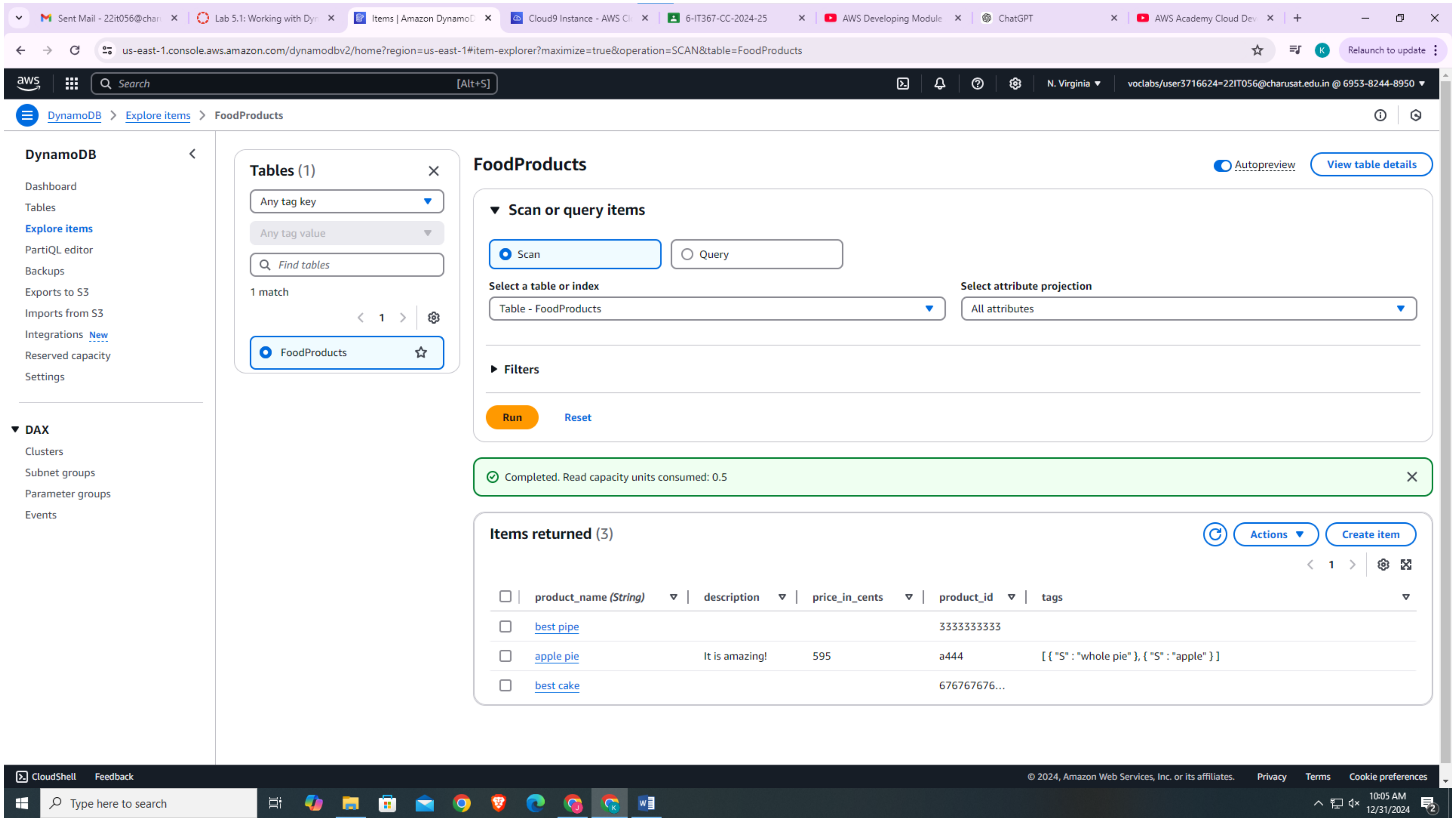
Figure 9: : Modify product id in .json file.



```python
'''
    You must replace <FMI_1> with the table name FoodProducts
    You must replace <FMI_2> with a product name. apple pie
    You must replace <FMI_3> with a444
    You must replace <FMI_4> with 595
    You must replace <FMI_5> with the description: It is amazing!
    You must replace <FMI_6> with a tag: whole pie
    You must replace <FMI_7> with a tag: apple
'''

import boto3
from botocore.exceptions import ClientError

def conditional_put():

    DDB = boto3.client('dynamodb', region_name='us-east-1')

    try:
        response = DDB.put_item(
            TableName='FoodProducts',
            Item={
                'product_name': {
                    'S': 'apple pie'
                },
                'product_id': {
                    'S': 'a444'
                },
                'price_in_cents':{
                    'N': '595' #number passed in as a string (ie in quotes)
                },
                'description':{
                    'S': "It is amazing!"
                },
                'tags':{
                    'L': [{
                            'S': 'whole pie'
                        },{
                            'S': 'apple'
                        }]
                }
            },
            ConditionExpression='attribute_not_exists(product_name)'
        )
    except ClientError as e:
        # Ignore the ConditionalCheckFailedException, bubble up
        # other exceptions.
        if e.response['Error']['Code'] != 'ConditionalCheckFailedException':
```

```
> --table-name FoodProducts \
> --item file://../resources/an_existing_product.json \
> --condition-expression "attribute_not_exists(product_name)" \
> --region us-east-1

An error occurred (ConditionalCheckFailedException) when calling the PutItem operation: The conditional request failed
voclabs:~/environment/python_3 $ python3 conditional_put.py
Done
voclabs:~/environment/python_3 $
```

Figure 10: Modify and run conditional_put.py file.


FoodProducts

Scan or query items

Scan [✓] Query

Select a table or index: Table - FoodProducts

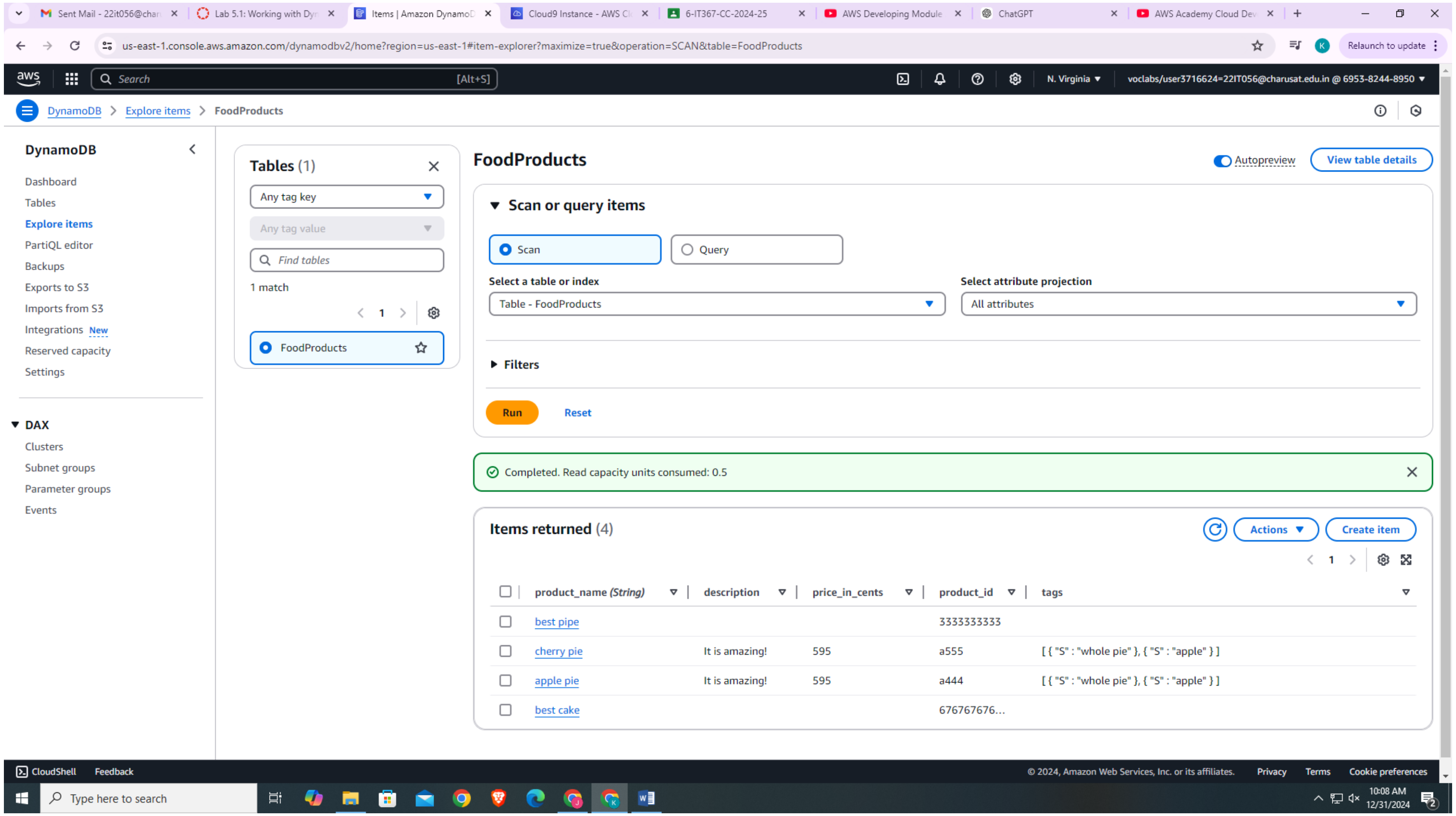
Select attribute projection: All attributes

Completed. Read capacity units consumed: 0.5

Items returned (3)

product_name (String)	description	price_in_cents	product_id	tags
best pipe			3333333333	
apple pie	It is amazing!	595	a444	[{ "S" : "whole pie" }, { "S" : "apple" }]
best cake			676767676...	

Figure 11: : Review the updated data.



FoodProducts

Scan or query items

Scan [✓] Query

Select a table or index: Table - FoodProducts

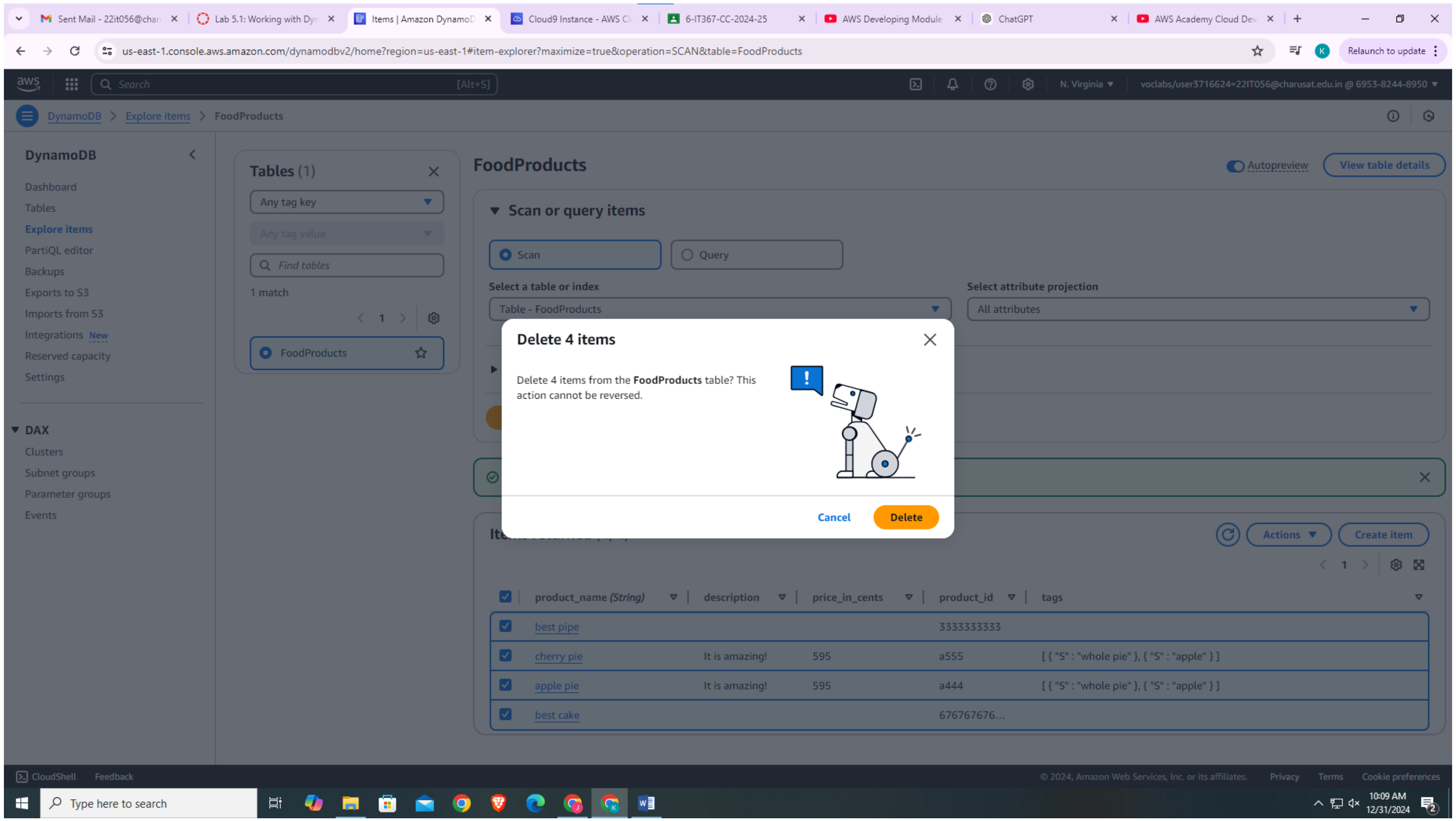
Select attribute projection: All attributes

Completed. Read capacity units consumed: 0.5

Items returned (4)

product_name (String)	description	price_in_cents	product_id	tags
best pipe			3333333333	
cherry pie	It is amazing!	595	a555	[{ "S" : "whole pie" }, { "S" : "apple" }]
apple pie	It is amazing!	595	a444	[{ "S" : "whole pie" }, { "S" : "apple" }]
best cake			676767676...	

Figure 12: New record was added to the table.



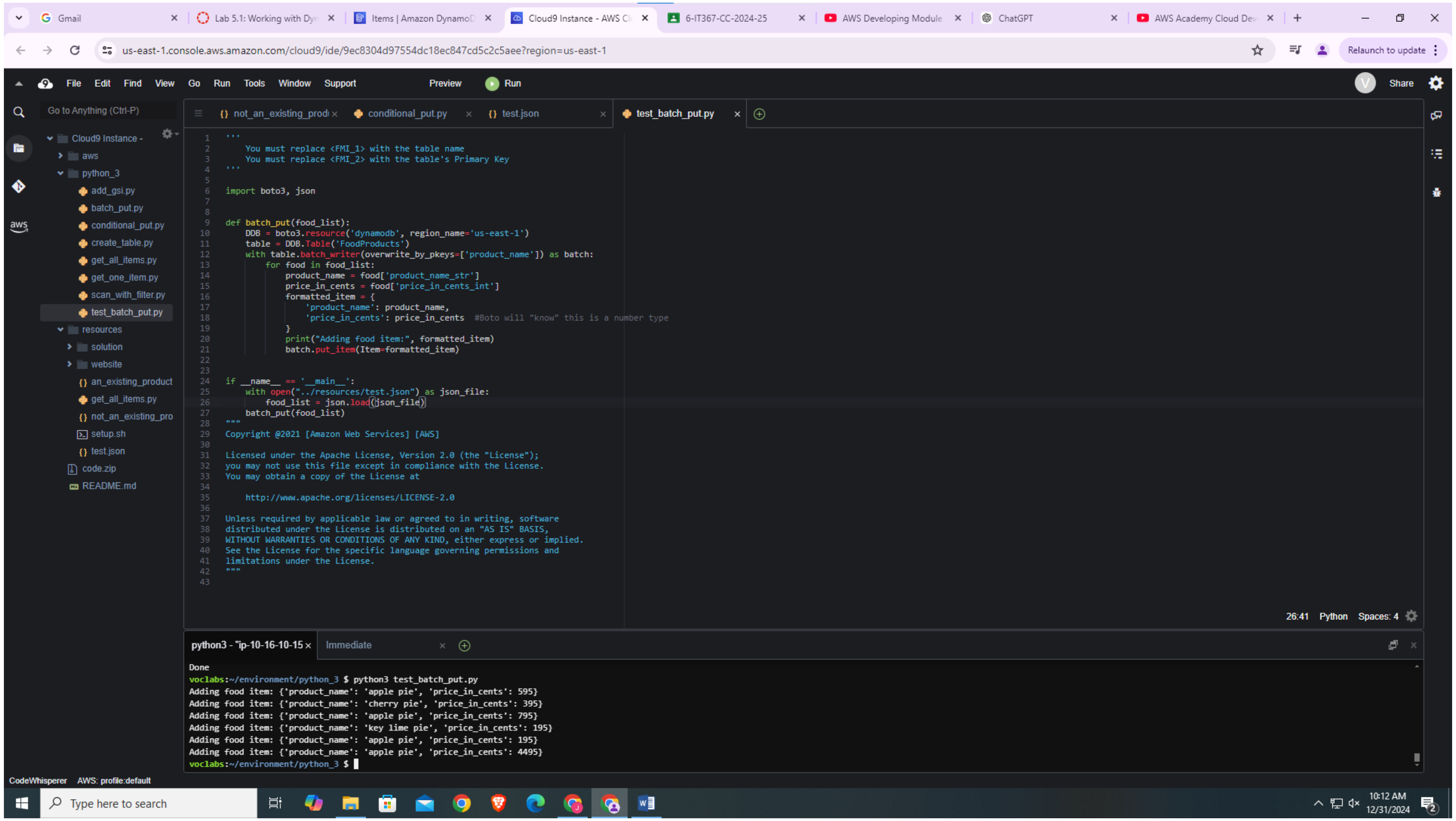
Delete 4 items

Delete 4 items from the **FoodProducts** table? This action cannot be reversed.

Cancel Delete

	product_name (String)	description	price_in_cents	product_id	tags
✓	best pipe			3333333333	
✓	cherry pie	It is amazing!	595	a555	[{ "S" : "whole pie" }, { "S" : "apple" }]
✓	apple pie	It is amazing!	595	a444	[{ "S" : "whole pie" }, { "S" : "apple" }]
✓	best cake			676767676...	

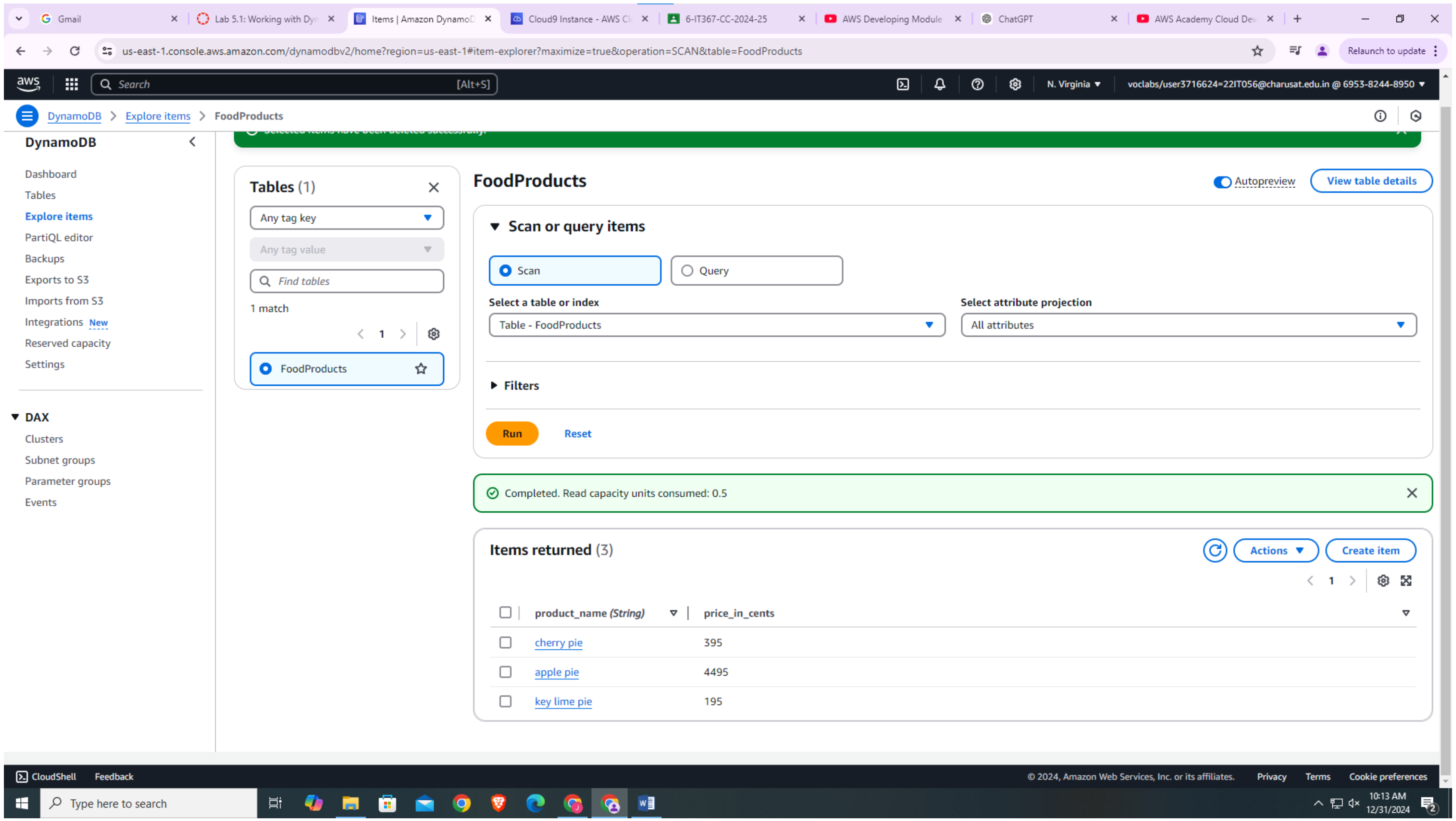
Figure 13: Successfully deleted items from table.



```
'''
    You must replace <FMI_1> with the table name
    You must replace <FMI_2> with the table's Primary Key
'''

import boto3, json


def batch_put(food_list):
    DDB = boto3.resource('dynamodb', region_name='us-east-1')
    table = DDB.Table('FoodProducts')
    with table.batch_writer(overwrite_by_pkeys=['product_name']) as batch:
        for food in food_list:
            product_name = food['product_name_str']
            price_in_cents = food['price_in_cents_int']
            formatted_item = {
                'product_name': product_name,
                'price_in_cents': price_in_cents  #Boto will "know" this is a number type
            }
            print("Adding food item:", formatted_item)
            batch.put_item(Item=formatted_item)


if __name__ == '__main__':
    with open("../resources/test.json") as json_file:
        food_list = json.load(json_file)
    batch_put(food_list)
"""
Copyright @2021 [Amazon Web Services] [AWS]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
"""
```

```
Done
voclabs:~/environment/python_3 $ python3 test_batch_put.py
Adding food item: {'product_name': 'apple pie', 'price_in_cents': 595}
Adding food item: {'product_name': 'cherry pie', 'price_in_cents': 395}
Adding food item: {'product_name': 'apple pie', 'price_in_cents': 795}
Adding food item: {'product_name': 'key lime pie', 'price_in_cents': 195}
Adding food item: {'product_name': 'apple pie', 'price_in_cents': 195}
Adding food item: {'product_name': 'apple pie', 'price_in_cents': 4495}
voclabs:~/environment/python_3 $
```

Figure 14:Modify and run test_batch_put.py file.



us-east-1.console.aws.amazon.com/dynamodbv2/home?region=us-east-1#item-explorer?maximize=true&operation=SCAN&table=FoodProducts

FoodProducts

Scan or query items

[✓ Scan] Query

Select a table or index: Table - FoodProducts

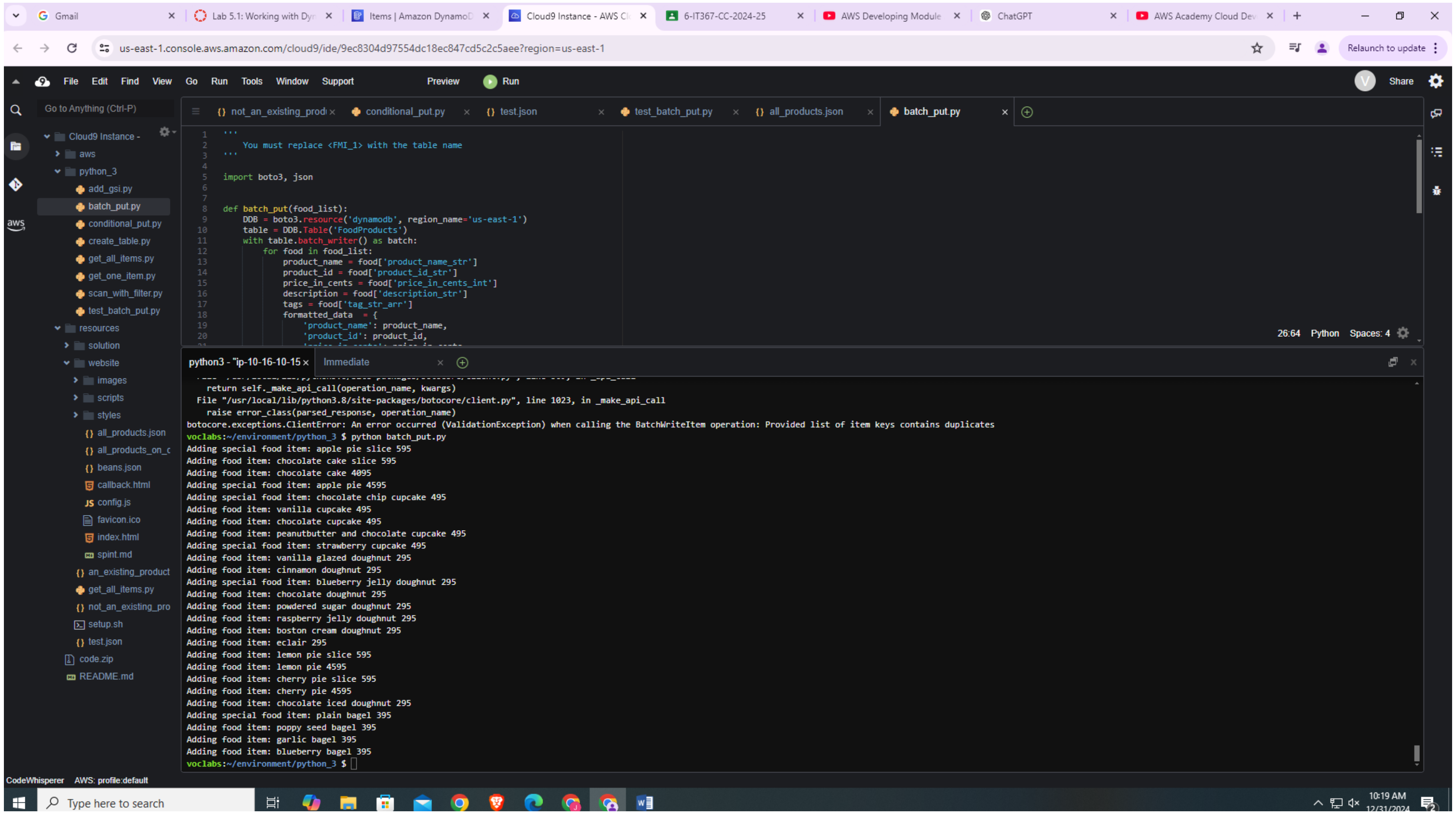
Select attribute projection: All attributes

Completed. Read capacity units consumed: 0.5

Items returned (3)

product_name (String)	price_in_cents
cherry pie	395
apple pie	4495
key lime pie	195

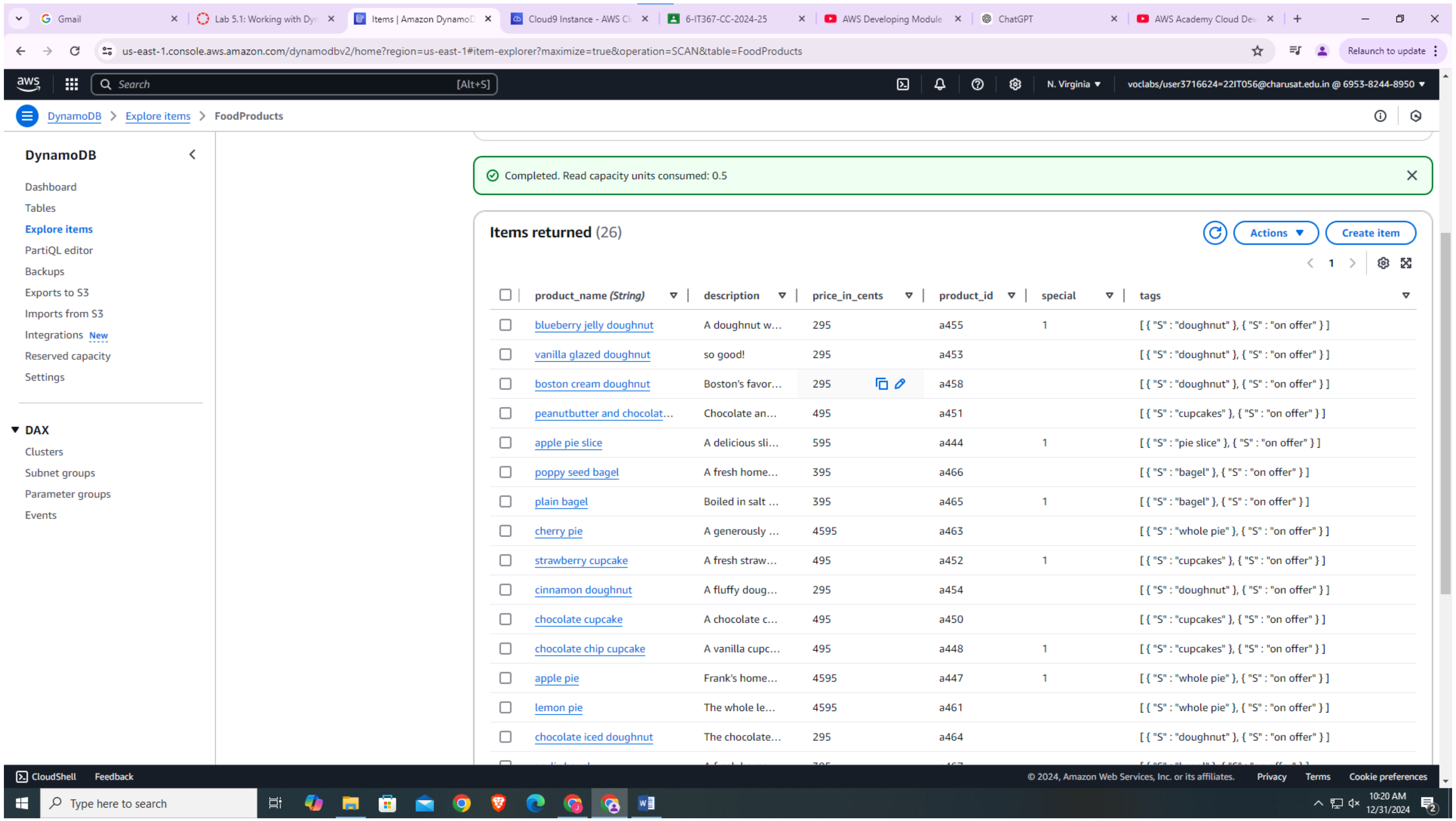
Figure 15:New record was added to the table.



```python
'''
    You must replace <FMI_1> with the table name
'''

import boto3, json


def batch_put(food_list):
    DDB = boto3.resource('dynamodb', region_name='us-east-1')
    table = DDB.Table('FoodProducts')
    with table.batch_writer() as batch:
        for food in food_list:
            product_name = food['product_name_str']
            product_id = food['product_id_str']
            price_in_cents = food['price_in_cents_int']
            description = food['description_str']
            tags = food['tag_str_arr']
            formatted_data  = {
                'product_name': product_name,
                'product_id': product_id,
```

```
    return self._make_api_call(operation_name, kwargs)
  File "/usr/local/lib/python3.8/site-packages/botocore/client.py", line 1023, in _make_api_call
    raise error_class(parsed_response, operation_name)
botocore.exceptions.ClientError: An error occurred (ValidationException) when calling the BatchWriteItem operation: Provided list of item keys contains duplicates
voclabs:~/environment/python_3 $ python batch_put.py
Adding special food item: apple pie slice 595
Adding food item: chocolate cake slice 595
Adding food item: chocolate cake 4095
Adding special food item: apple pie 4595
Adding special food item: chocolate chip cupcake 495
Adding food item: vanilla cupcake 495
Adding food item: chocolate cupcake 495
Adding food item: peanutbutter and chocolate cupcake 495
Adding special food item: strawberry cupcake 495
Adding food item: vanilla glazed doughnut 295
Adding food item: cinnamon doughnut 295
Adding special food item: blueberry jelly doughnut 295
Adding food item: chocolate doughnut 295
Adding food item: powdered sugar doughnut 295
Adding food item: raspberry jelly doughnut 295
Adding food item: boston cream doughnut 295
Adding food item: eclair 295
Adding food item: lemon pie slice 595
Adding food item: lemon pie 4595
Adding food item: cherry pie slice 595
Adding food item: cherry pie 4595
Adding food item: chocolate iced doughnut 295
Adding special food item: plain bagel 395
Adding food item: poppy seed bagel 395
Adding food item: garlic bagel 395
Adding food item: blueberry bagel 395
voclabs:~/environment/python_3 $
```

Figure 16: Modify and run batch_put.py file.

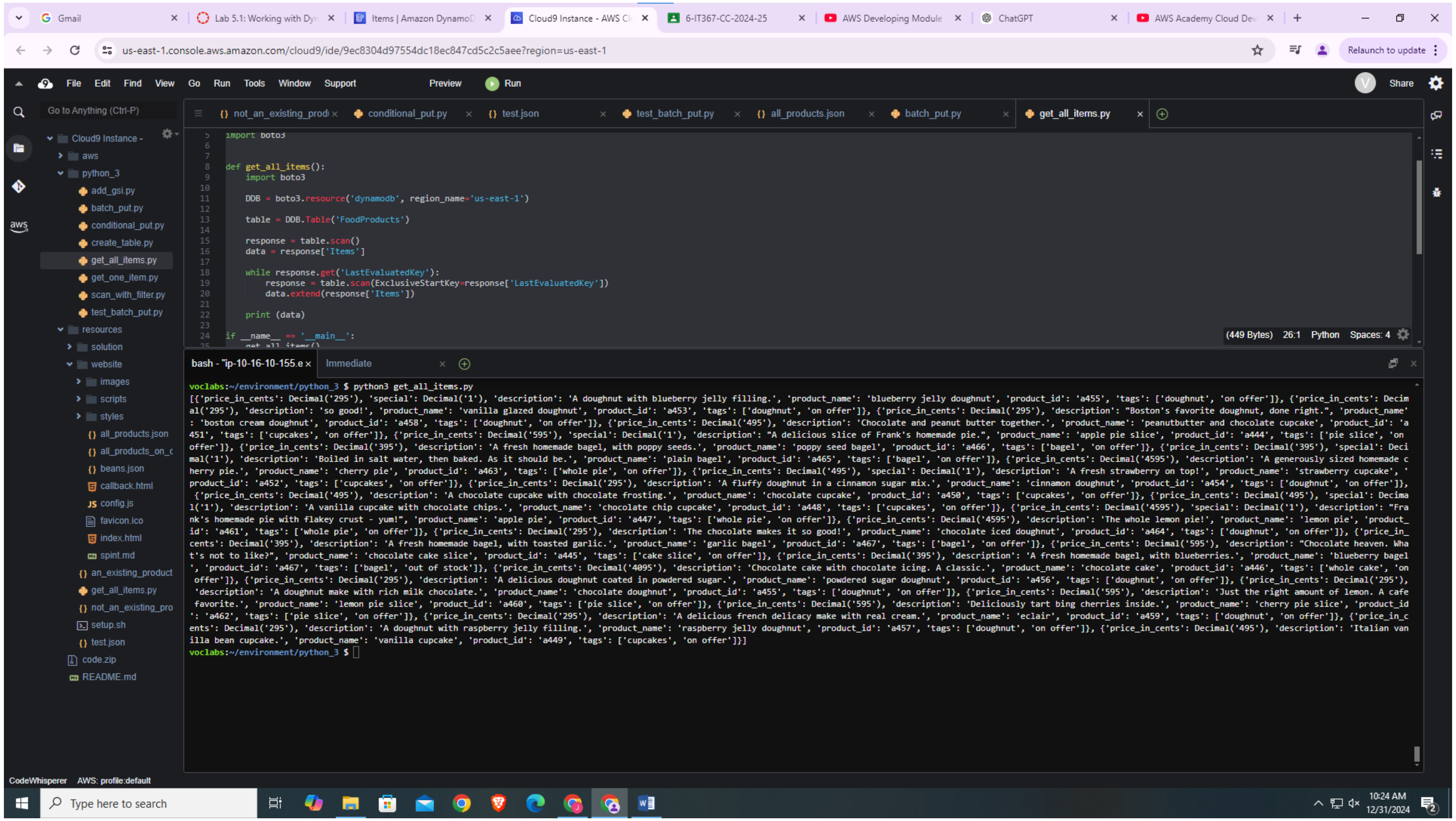


Completed. Read capacity units consumed: 0.5

Items returned (26)

product_name (String)	description	price_in_cents	product_id	special	tags
blueberry jelly doughnut	A doughnut w...	295	a455	1	[{ "S" : "doughnut" }, { "S" : "on offer" }]
vanilla glazed doughnut	so good!	295	a453		[{ "S" : "doughnut" }, { "S" : "on offer" }]
boston cream doughnut	Boston's favor...	295	a458		[{ "S" : "doughnut" }, { "S" : "on offer" }]
peanutbutter and chocolat...	Chocolate an...	495	a451		[{ "S" : "cupcakes" }, { "S" : "on offer" }]
apple pie slice	A delicious sli...	595	a444	1	[{ "S" : "pie slice" }, { "S" : "on offer" }]
poppy seed bagel	A fresh home...	395	a466		[{ "S" : "bagel" }, { "S" : "on offer" }]
plain bagel	Boiled in salt ...	395	a465	1	[{ "S" : "bagel" }, { "S" : "on offer" }]
cherry pie	A generously ...	4595	a463		[{ "S" : "whole pie" }, { "S" : "on offer" }]
strawberry cupcake	A fresh straw...	495	a452	1	[{ "S" : "cupcakes" }, { "S" : "on offer" }]
cinnamon doughnut	A fluffy doug...	295	a454		[{ "S" : "doughnut" }, { "S" : "on offer" }]
chocolate cupcake	A chocolate c...	495	a450		[{ "S" : "cupcakes" }, { "S" : "on offer" }]
chocolate chip cupcake	A vanilla cupc...	495	a448	1	[{ "S" : "cupcakes" }, { "S" : "on offer" }]
apple pie	Frank's home...	4595	a447	1	[{ "S" : "whole pie" }, { "S" : "on offer" }]
lemon pie	The whole le...	4595	a461		[{ "S" : "whole pie" }, { "S" : "on offer" }]
chocolate iced doughnut	The chocolate...	295	a464		[{ "S" : "doughnut" }, { "S" : "on offer" }]

Figure 17: Show data in table.

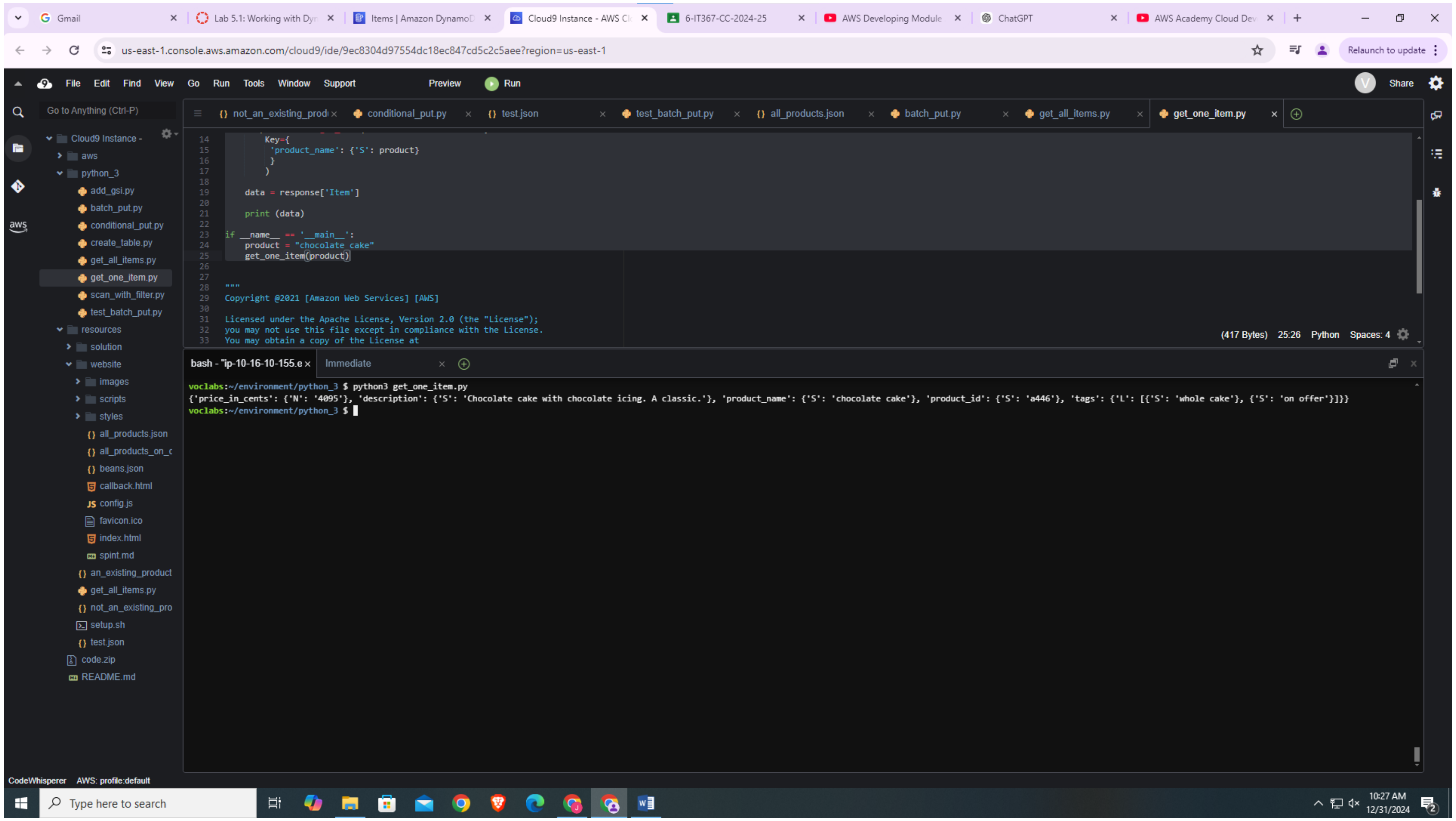


us-east-1.console.aws.amazon.com/cloud9/ide/9ec8304d97554dc18ec847cd5c2c5aee?region=us-east-1
not_an_existing_prod
conditional_put.py
test.json
test_batch_put.py
all_products.json
batch_put.py
get_all_items.py
(449 Bytes) 26:1 Python Spaces: 4
bash - "ip-10-16-10-155.e
Immediate

```python
import boto3


def get_all_items():
    import boto3

    DDB = boto3.resource('dynamodb', region_name='us-east-1')

    table = DDB.Table('FoodProducts')

    response = table.scan()
    data = response['Items']

    while response.get('LastEvaluatedKey'):
        response = table.scan(ExclusiveStartKey=response['LastEvaluatedKey'])
        data.extend(response['Items'])

    print (data)

if __name__ == '__main__':
    get_all_items()
```

```
voclabs:~/environment/python_3 $ python3 get_all_items.py
[{'price_in_cents': Decimal('295'), 'special': Decimal('1'), 'description': 'A doughnut with blueberry jelly filling.', 'product_name': 'blueberry jelly doughnut', 'product_id': 'a455', 'tags': ['doughnut', 'on offer']}, {'price_in_cents': Decim
al('295'), 'description': 'so good!', 'product_name': 'vanilla glazed doughnut', 'product_id': 'a453', 'tags': ['doughnut', 'on offer']}, {'price_in_cents': Decimal('295'), 'description': "Boston's favorite doughnut, done right.", 'product_name'
: 'boston cream doughnut', 'product_id': 'a458', 'tags': ['doughnut', 'on offer']}, {'price_in_cents': Decimal('495'), 'description': 'Chocolate and peanut butter together.', 'product_name': 'peanutbutter and chocolate cupcake', 'product_id': 'a
451', 'tags': ['cupcakes', 'on offer']}, {'price_in_cents': Decimal('595'), 'special': Decimal('1'), 'description': "A delicious slice of Frank's homemade pie.", 'product_name': 'apple pie slice', 'product_id': 'a444', 'tags': ['pie slice', 'on 
offer']}, {'price_in_cents': Decimal('395'), 'description': 'A fresh homemade bagel, with poppy seeds.', 'product_name': 'poppy seed bagel', 'product_id': 'a466', 'tags': ['bagel', 'on offer']}, {'price_in_cents': Decimal('395'), 'special': Deci
mal('1'), 'description': 'Boiled in salt water, then baked. As it should be.', 'product_name': 'plain bagel', 'product_id': 'a465', 'tags': ['bagel', 'on offer']}, {'price_in_cents': Decimal('4595'), 'description': 'A generously sized homemade c
herry pie.', 'product_name': 'cherry pie', 'product_id': 'a463', 'tags': ['whole pie', 'on offer']}, {'price_in_cents': Decimal('495'), 'special': Decimal('1'), 'description': 'A fresh strawberry on top!', 'product_name': 'strawberry cupcake', '
product_id': 'a452', 'tags': ['cupcakes', 'on offer']}, {'price_in_cents': Decimal('295'), 'description': 'A fluffy doughnut in a cinnamon sugar mix.', 'product_name': 'cinnamon doughnut', 'product_id': 'a454', 'tags': ['doughnut', 'on offer']},
 {'price_in_cents': Decimal('495'), 'description': 'A chocolate cupcake with chocolate frosting.', 'product_name': 'chocolate cupcake', 'product_id': 'a450', 'tags': ['cupcakes', 'on offer']}, {'price_in_cents': Decimal('495'), 'special': Decima
l('1'), 'description': 'A vanilla cupcake with chocolate chips.', 'product_name': 'chocolate chip cupcake', 'product_id': 'a448', 'tags': ['cupcakes', 'on offer']}, {'price_in_cents': Decimal('4595'), 'special': Decimal('1'), 'description': "Fra
nk's homemade pie with flakey crust - yum!", 'product_name': 'apple pie', 'product_id': 'a447', 'tags': ['whole pie', 'on offer']}, {'price_in_cents': Decimal('4595'), 'description': 'The whole lemon pie!', 'product_name': 'lemon pie', 'product_
id': 'a461', 'tags': ['whole pie', 'on offer']}, {'price_in_cents': Decimal('295'), 'description': 'The chocolate makes it so good!', 'product_name': 'chocolate iced doughnut', 'product_id': 'a464', 'tags': ['doughnut', 'on offer']}, {'price_in_
cents': Decimal('395'), 'description': 'A fresh homemade bagel, with toasted garlic.', 'product_name': 'garlic bagel', 'product_id': 'a467', 'tags': ['bagel', 'on offer']}, {'price_in_cents': Decimal('595'), 'description': "Chocolate heaven. Wha
t's not to like?", 'product_name': 'chocolate cake slice', 'product_id': 'a445', 'tags': ['cake slice', 'on offer']}, {'price_in_cents': Decimal('395'), 'description': 'A fresh homemade bagel, with blueberries.', 'product_name': 'blueberry bagel
', 'product_id': 'a467', 'tags': ['bagel', 'out of stock']}, {'price_in_cents': Decimal('4095'), 'description': 'Chocolate cake with chocolate icing. A classic.', 'product_name': 'chocolate cake', 'product_id': 'a446', 'tags': ['whole cake', 'on
 offer']}, {'price_in_cents': Decimal('295'), 'description': 'A delicious doughnut coated in powdered sugar.', 'product_name': 'powdered sugar doughnut', 'product_id': 'a456', 'tags': ['doughnut', 'on offer']}, {'price_in_cents': Decimal('295'),
 'description': 'A doughnut make with rich milk chocolate.', 'product_name': 'chocolate doughnut', 'product_id': 'a455', 'tags': ['doughnut', 'on offer']}, {'price_in_cents': Decimal('595'), 'description': 'Just the right amount of lemon. A cafe
 favorite.', 'product_name': 'lemon pie slice', 'product_id': 'a460', 'tags': ['pie slice', 'on offer']}, {'price_in_cents': Decimal('595'), 'description': 'Deliciously tart bing cherries inside.', 'product_name': 'cherry pie slice', 'product_id
': 'a462', 'tags': ['pie slice', 'on offer']}, {'price_in_cents': Decimal('295'), 'description': 'A delicious french delicacy make with real cream.', 'product_name': 'eclair', 'product_id': 'a459', 'tags': ['doughnut', 'on offer']}, {'price_in_c
ents': Decimal('295'), 'description': 'A doughnut with raspberry jelly filling.', 'product_name': 'raspberry jelly doughnut', 'product_id': 'a457', 'tags': ['doughnut', 'on offer']}, {'price_in_cents': Decimal('495'), 'description': 'Italian van
illa bean cupcake.', 'product_name': 'vanilla cupcake', 'product_id': 'a449', 'tags': ['cupcakes', 'on offer']}]
voclabs:~/environment/python_3 $
```

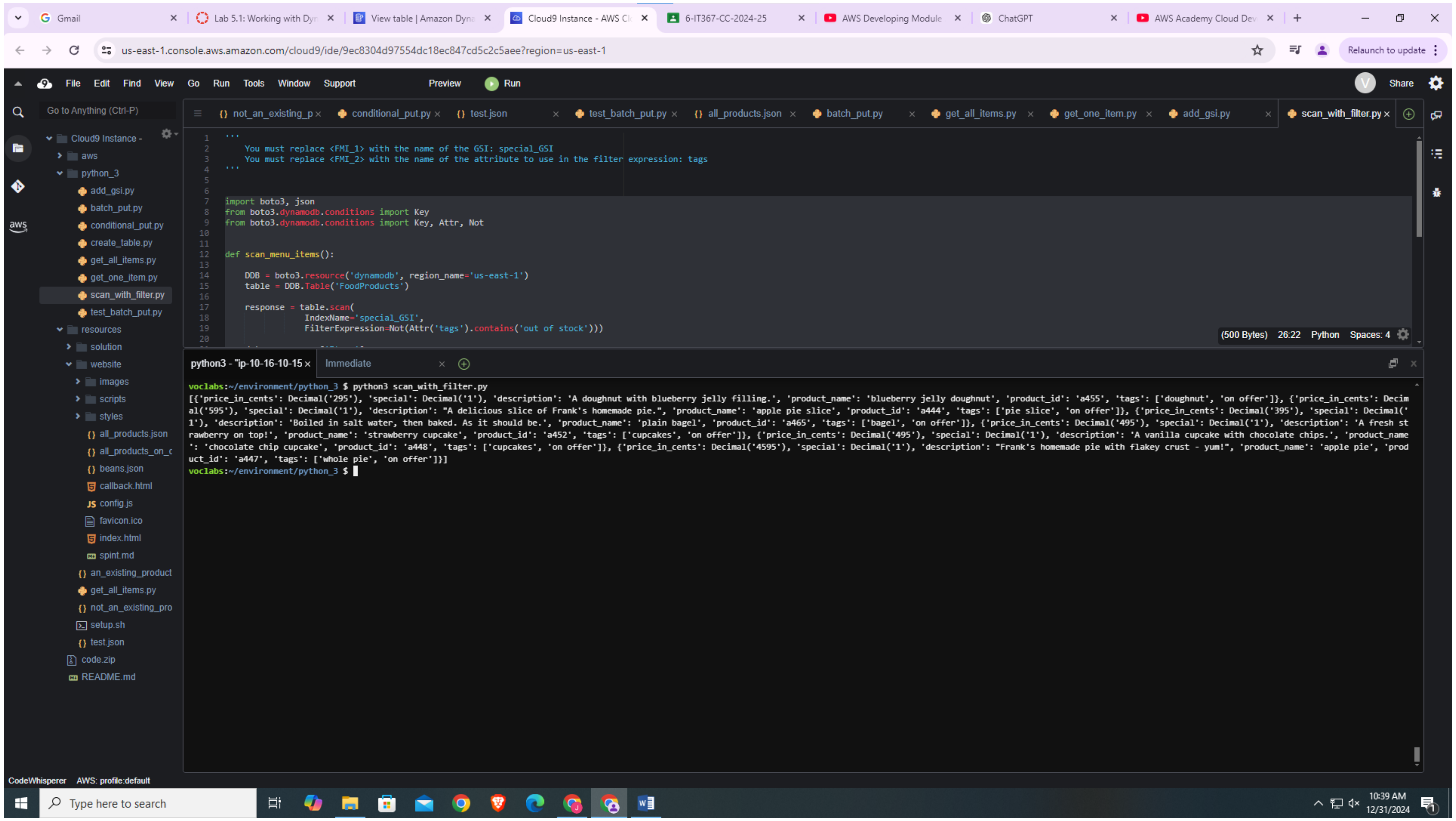
Figure 18: Modify and run get_all_data.py file.



```python
            Key={
             'product_name': {'S': product}
             }
            )

    data = response['Item']

    print (data)

if __name__ == '__main__':
    product = "chocolate cake"
    get_one_item(product)


"""
Copyright @2021 [Amazon Web Services] [AWS]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
```

```
voclabs:~/environment/python_3 $ python3 get_one_item.py
{'price_in_cents': {'N': '4095'}, 'description': {'S': 'Chocolate cake with chocolate icing. A classic.'}, 'product_name': {'S': 'chocolate cake'}, 'product_id': {'S': 'a446'}, 'tags': {'L': [{'S': 'whole cake'}, {'S': 'on offer'}]}}
voclabs:~/environment/python_3 $
```

Figure 19:Modify and run get_one_item.py file.



```
'''
    You must replace <FMI_1> with the name of the GSI: special_GSI
    You must replace <FMI_2> with the name of the attribute to use in the filter expression: tags
'''


import boto3, json
from boto3.dynamodb.conditions import Key
from boto3.dynamodb.conditions import Key, Attr, Not


def scan_menu_items():

    DDB = boto3.resource('dynamodb', region_name='us-east-1')
    table = DDB.Table('FoodProducts')

    response = table.scan(
                IndexName='special_GSI',
                FilterExpression=Not(Attr('tags').contains('out of stock')))

```

```
voclabs:~/environment/python_3 $ python3 scan_with_filter.py
[{'price_in_cents': Decimal('295'), 'special': Decimal('1'), 'description': 'A doughnut with blueberry jelly filling.', 'product_name': 'blueberry jelly doughnut', 'product_id': 'a455', 'tags': ['doughnut', 'on offer']}, {'price_in_cents': Decim
al('595'), 'special': Decimal('1'), 'description': "A delicious slice of Frank's homemade pie.", 'product_name': 'apple pie slice', 'product_id': 'a444', 'tags': ['pie slice', 'on offer']}, {'price_in_cents': Decimal('395'), 'special': Decimal('
1'), 'description': 'Boiled in salt water, then baked. As it should be.', 'product_name': 'plain bagel', 'product_id': 'a465', 'tags': ['bagel', 'on offer']}, {'price_in_cents': Decimal('495'), 'special': Decimal('1'), 'description': 'A fresh st
rawberry on top!', 'product_name': 'strawberry cupcake', 'product_id': 'a452', 'tags': ['cupcakes', 'on offer']}, {'price_in_cents': Decimal('495'), 'special': Decimal('1'), 'description': 'A vanilla cupcake with chocolate chips.', 'product_name
': 'chocolate chip cupcake', 'product_id': 'a448', 'tags': ['cupcakes', 'on offer']}, {'price_in_cents': Decimal('4595'), 'special': Decimal('1'), 'description': "Frank's homemade pie with flakey crust - yum!", 'product_name': 'apple pie', 'prod
uct_id': 'a447', 'tags': ['whole pie', 'on offer']}]
voclabs:~/environment/python_3 $
```

Figure 20: Modify and run scan_with_filter.py file.

LATEST APPLICATIONS:

1. Real-Time Data Processing
2. Serverless Architectures
3. Gaming Leaderboards
4. IoT Device Management
5. AI/ML Model Metadata Storage
6. E-Commerce and Personalization
7. Mobile App Backends

LEARNING OUTCOME:

By completing this practical, learners gain the ability to create and manage DynamoDB tables for storing and retrieving data, add and update items using AWS CLI and Boto3, and implement efficient querying techniques with Global Secondary Indexes (GSIs). They develop hands-on experience in integrating DynamoDB with dynamic applications, enabling real-time data updates. This exercise enhances understanding of NoSQL databases, teaches best practices for scalable and efficient data management, and demonstrates real-world applications such as e-commerce, mobile backends, and personalized user experiences.

REFERENCE:

1. https://awsacademy.instructure.com/courses/104050