AWS DynamoDB Lab

2021-01-29 | 2022-02-15

| Tech , AWS , AWS Certificate , Labs , AWS DAS , AWS , DB , DB , NoSQL

15k | 14 mins.

AWS DynamoDB

AWS DynamoDB

DynamoDB

Introduction to AWS DynamoDB

> https://play.whizlabs.com/site/task_details?lab_type=1&task_id=13&quest_id=35

Lab Details

1. This lab walks you through Amazon DynamoDB features. We will create a table in Amazon DynamoDB to store information and then query that information from the DynamoDB table.

Introduction

What is AWS DynamoDB?

- Definition
    - DynamoDB is a fast and flexible NoSQL database designed for applications that need consistent, single-digit millisecond latency at any scale. It is a fully managed database and it supports both document and key value data models.

- It has a very flexible data model. This means that you don't need to define your database schema upfront. It also has reliable performance.
- DynamoDB is a good fit for mobile gaming, ad-tech, IoT and many other applications.

DynamoDB Tables

DynamoDB tables consist of

- **Items** (Think of a row of data in a table).
- **Attributes** (Think of a column of data in a table).
- Supports **key-value** and **document data structures**.
- `Key` = the name of the data. `Value` = the data itself.
- `Document` can be written in **JSON**, **HTML** or **XML**.

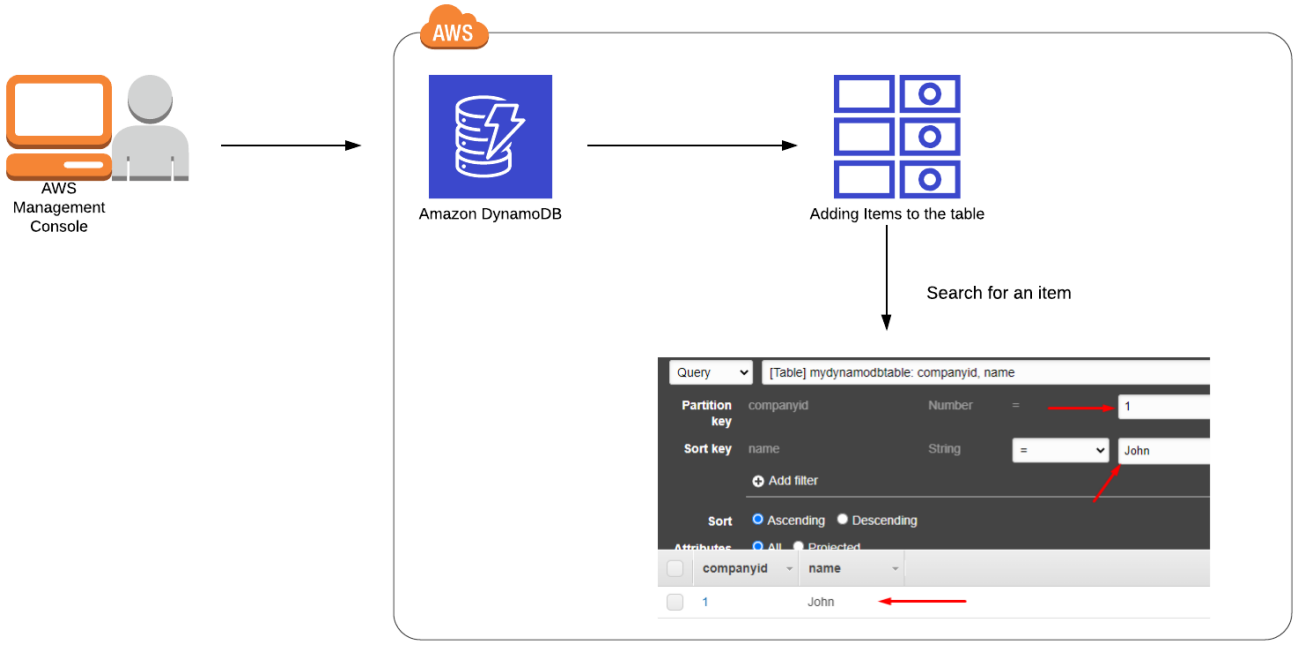
DynamoDB- Primary Keys

- DynamoDB stores and retrieves data based on a **Primary key**.
- DynamoDB also uses **Partition keys** to determine the physical location data is stored.
- **If you are using a partition key as your Primary key, then no items will have the same Partition key.**
- **All items with the same partition key are stored together and then sorted according to the sort key value.**
- **Composite Keys (Partition Key + Sort Key) can be used in combination.**
- **Two items may have the same partition key, but must have a different sort key.**

Lab Tasks

1. Log into AWS Management Console.
2. Create a DynamoDB table.
3. Insert data into that DynamoDB table.
4. Search for an item in the DynamoDB table.

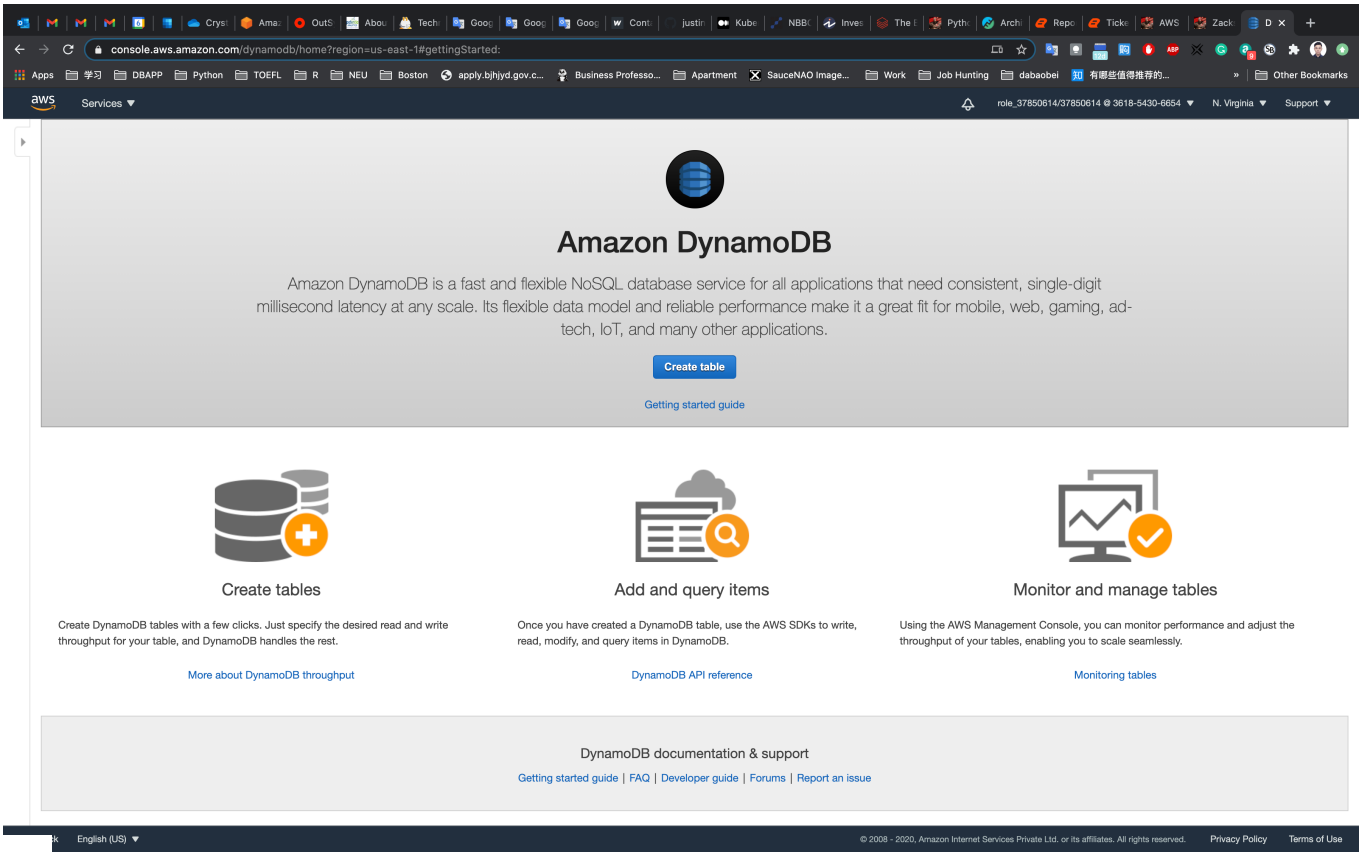
Architecture Diagram



DynamoDB Configuration

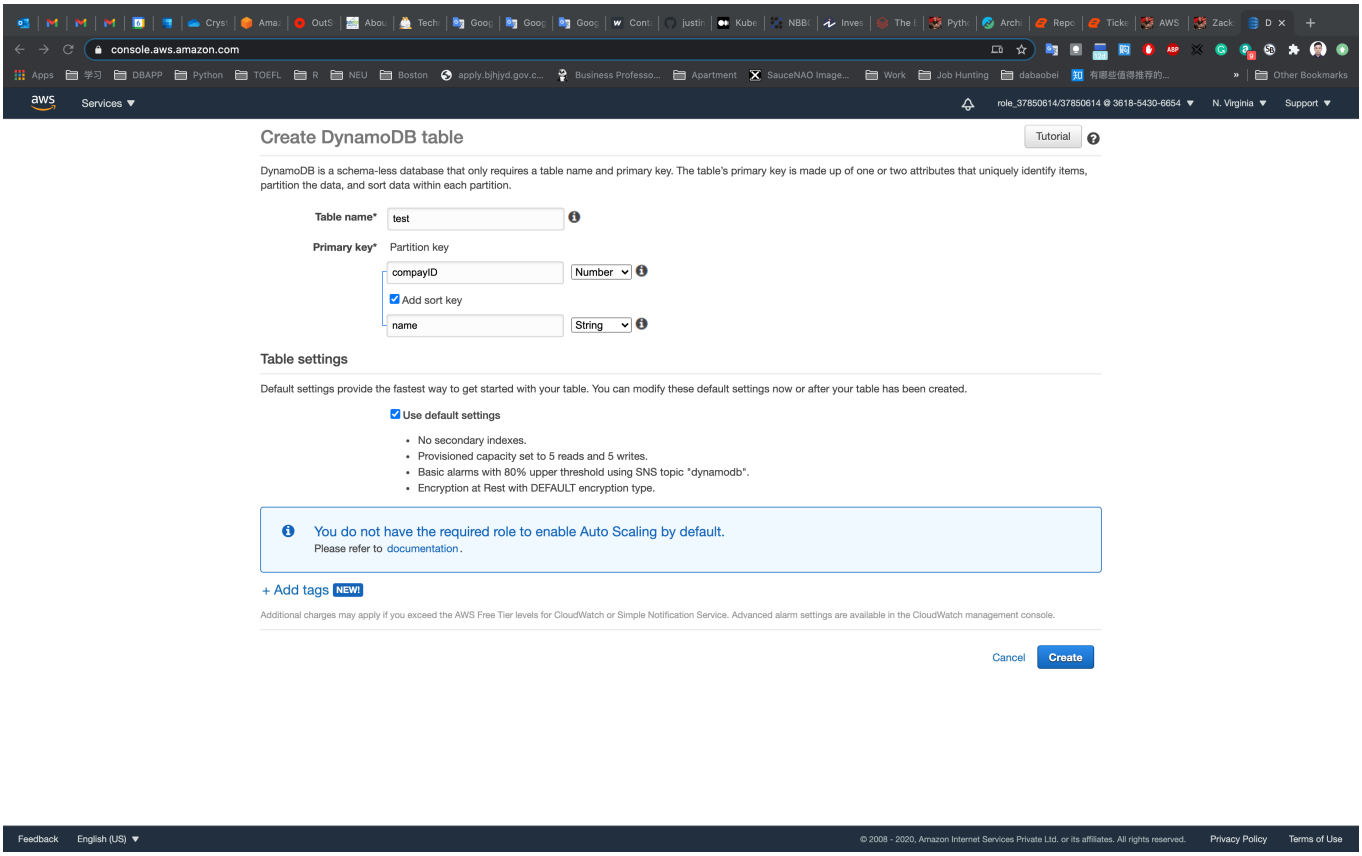
Services -> DynamoDB

Click on `Create table`.

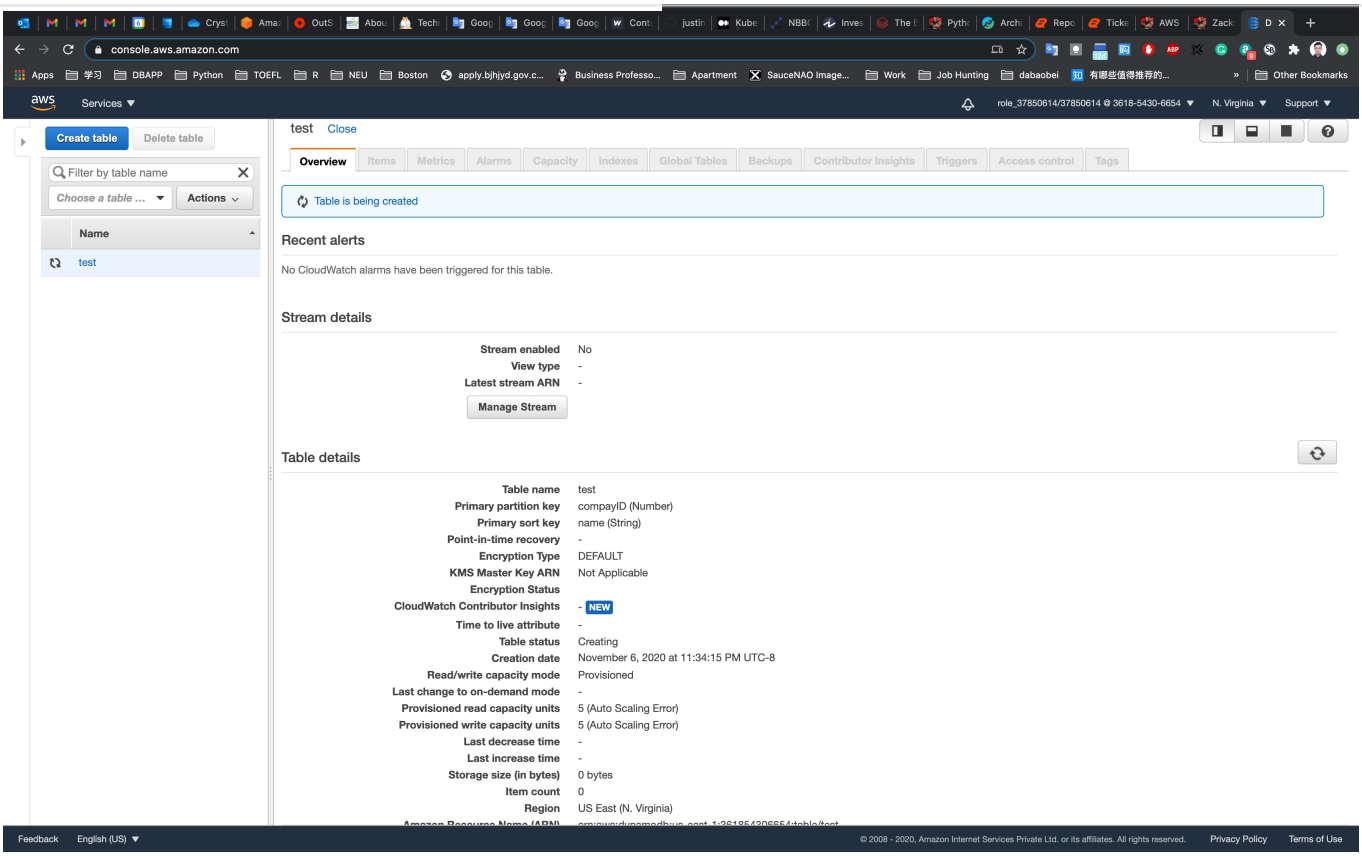


- Table Name: `test`
- Primary key: `companyID` and select `Number`
- Add sort key: Enter **name** in the respective field and select `String`.
- The combination of a Primary Key and a Sort Key uniquely identifies each item in a DynamoDB table.

Click on `Create`.



Your table will be created within 2-3 minutes.

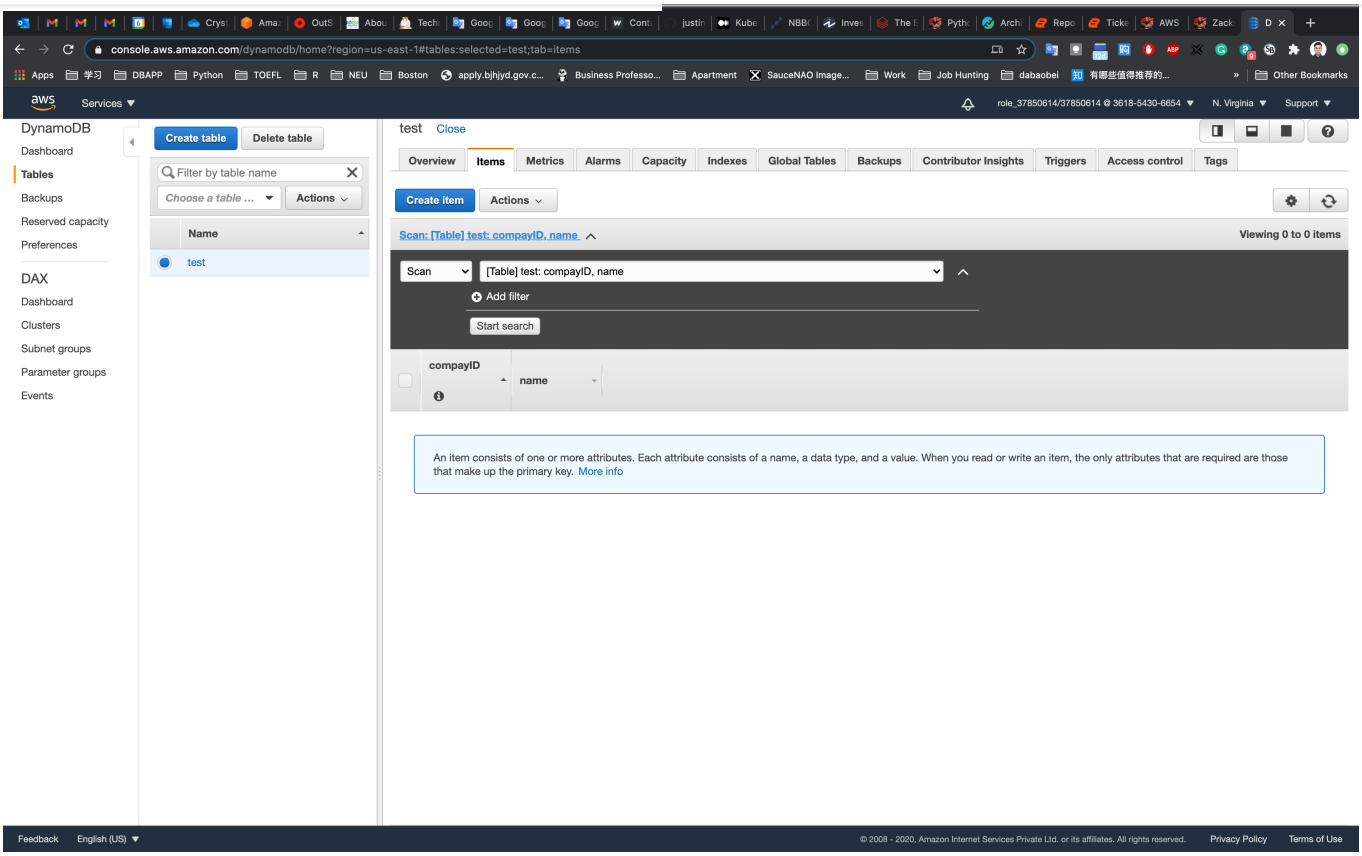


Query DynamoDB

Insert Items

Next, we are inserting data in the table we created.

Click on the `Items` tab.

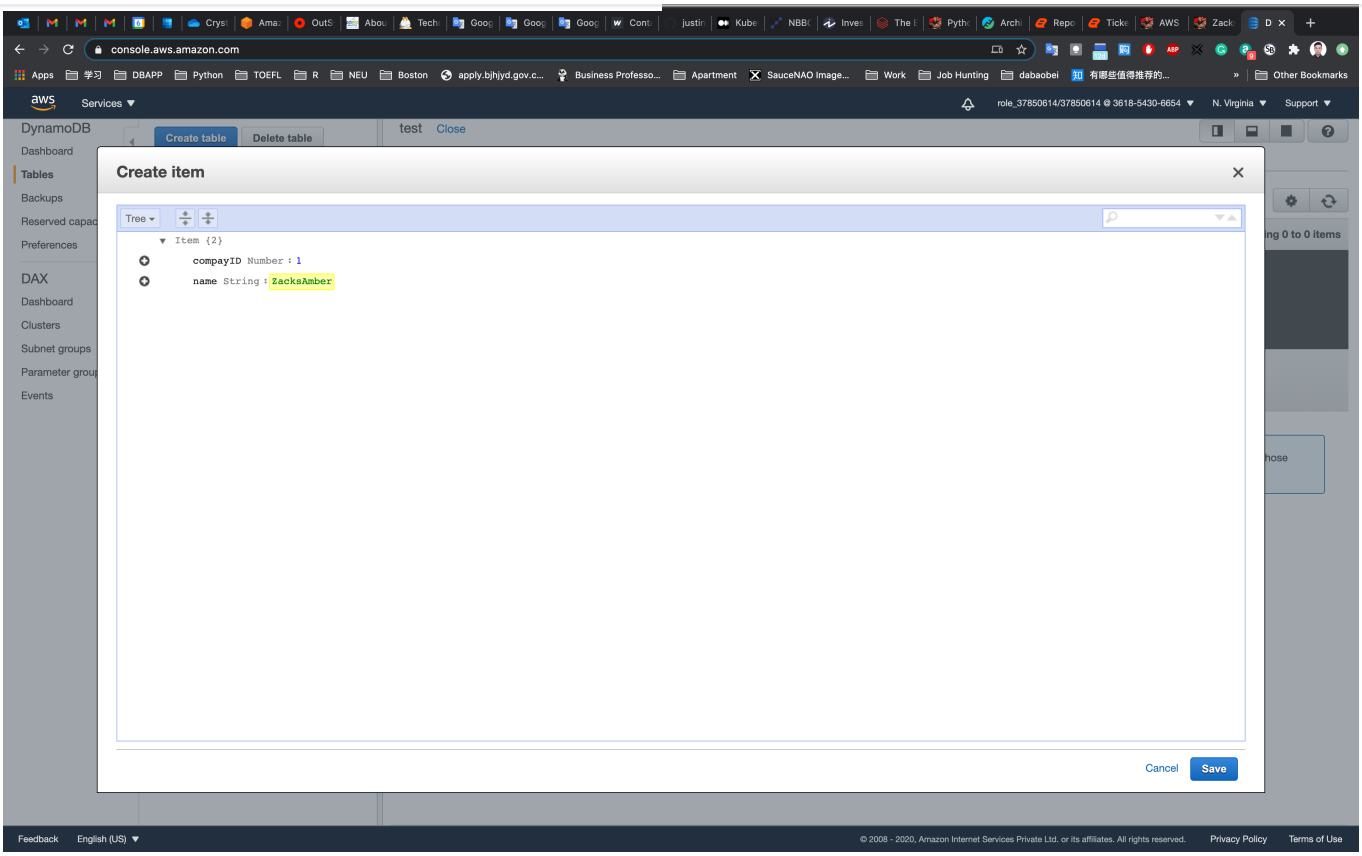


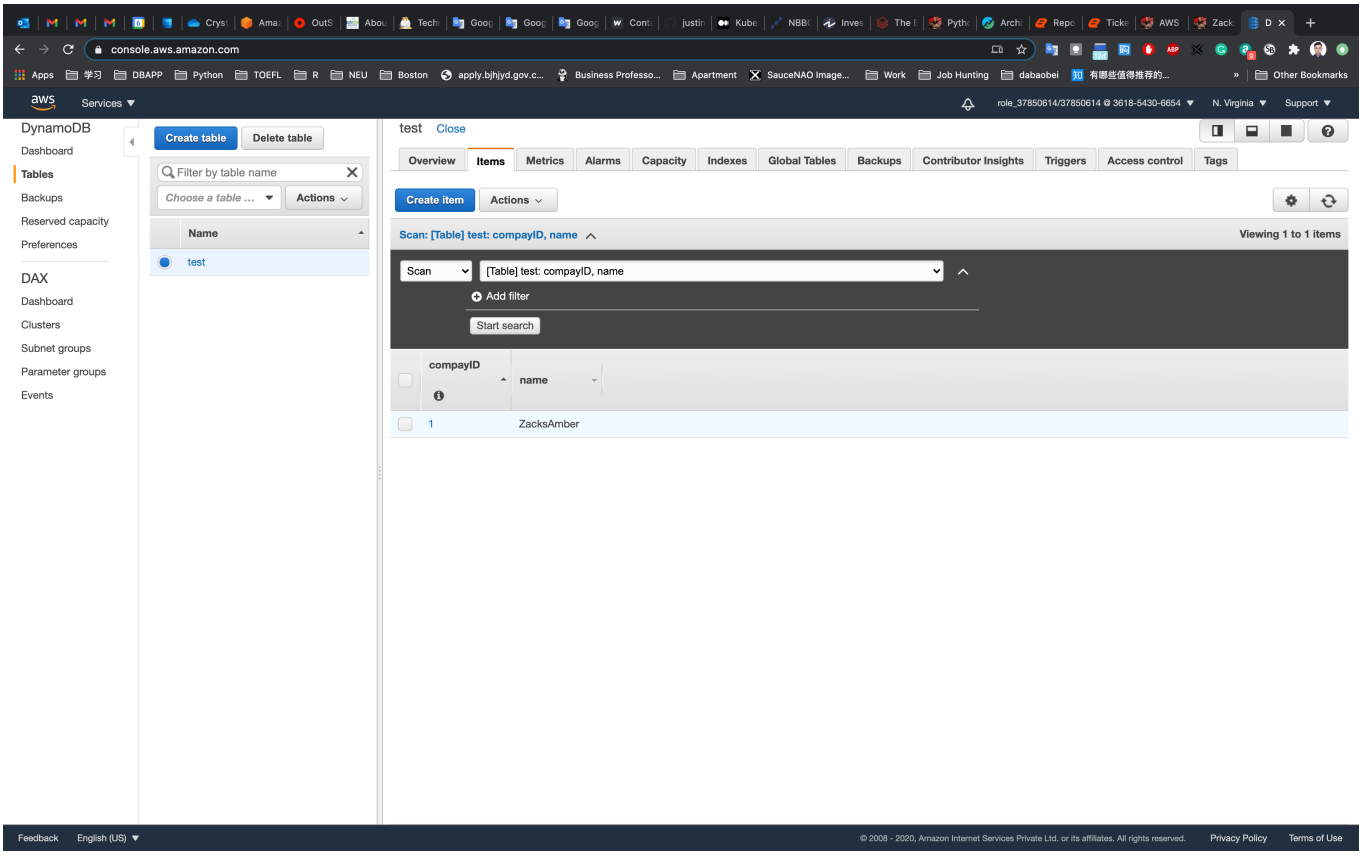
Click on `Create item`.

Add new primary key and sort key values.

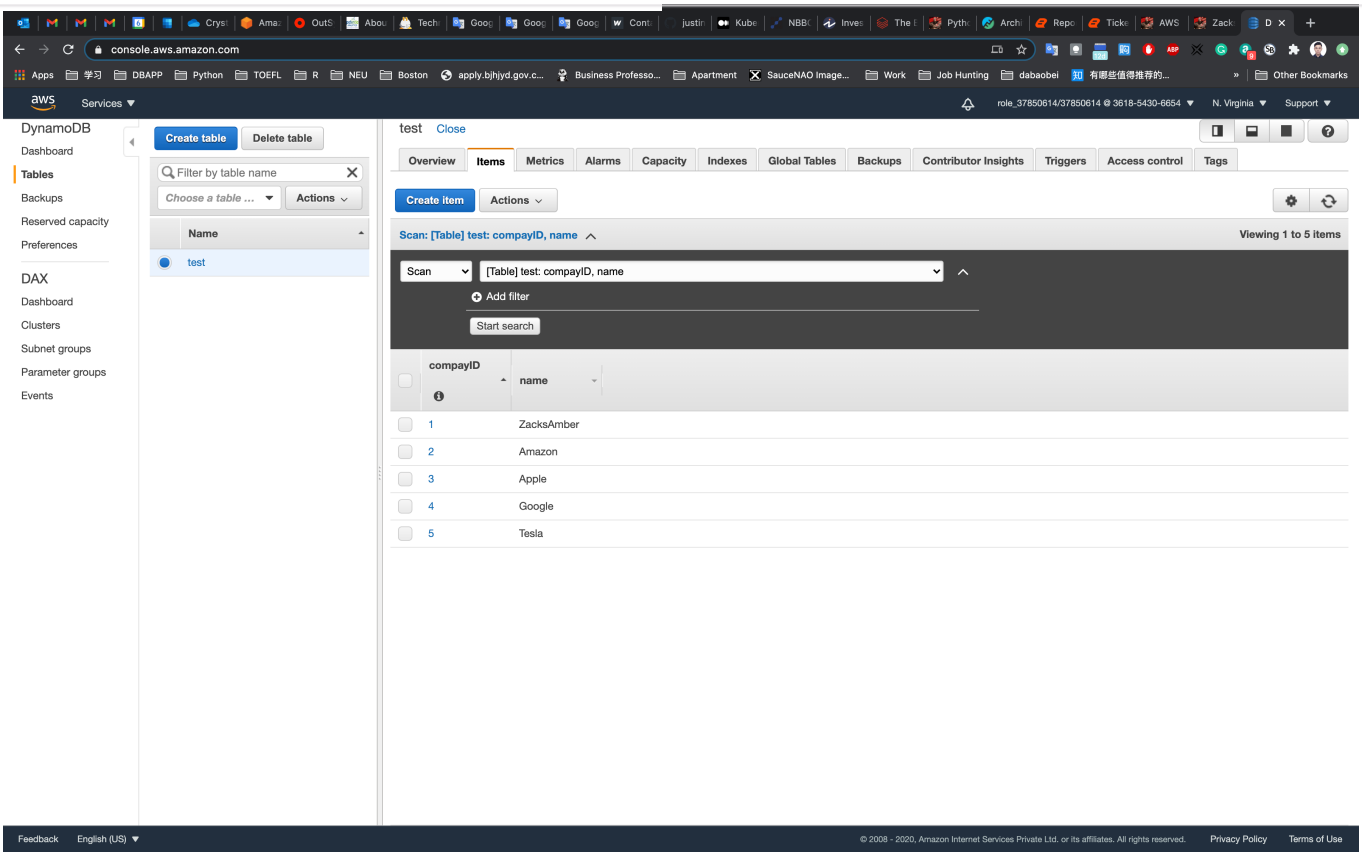
- companyid: `1`
- name: `ZacksAmber`

Click on `Save`.





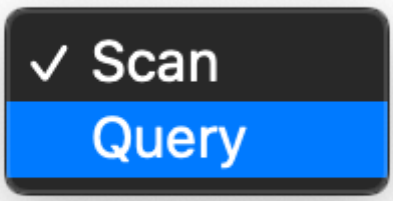
For testing purposes, add 4-5 items as shown in the above step.



Search for Items

We will query the items in our table by using scan.

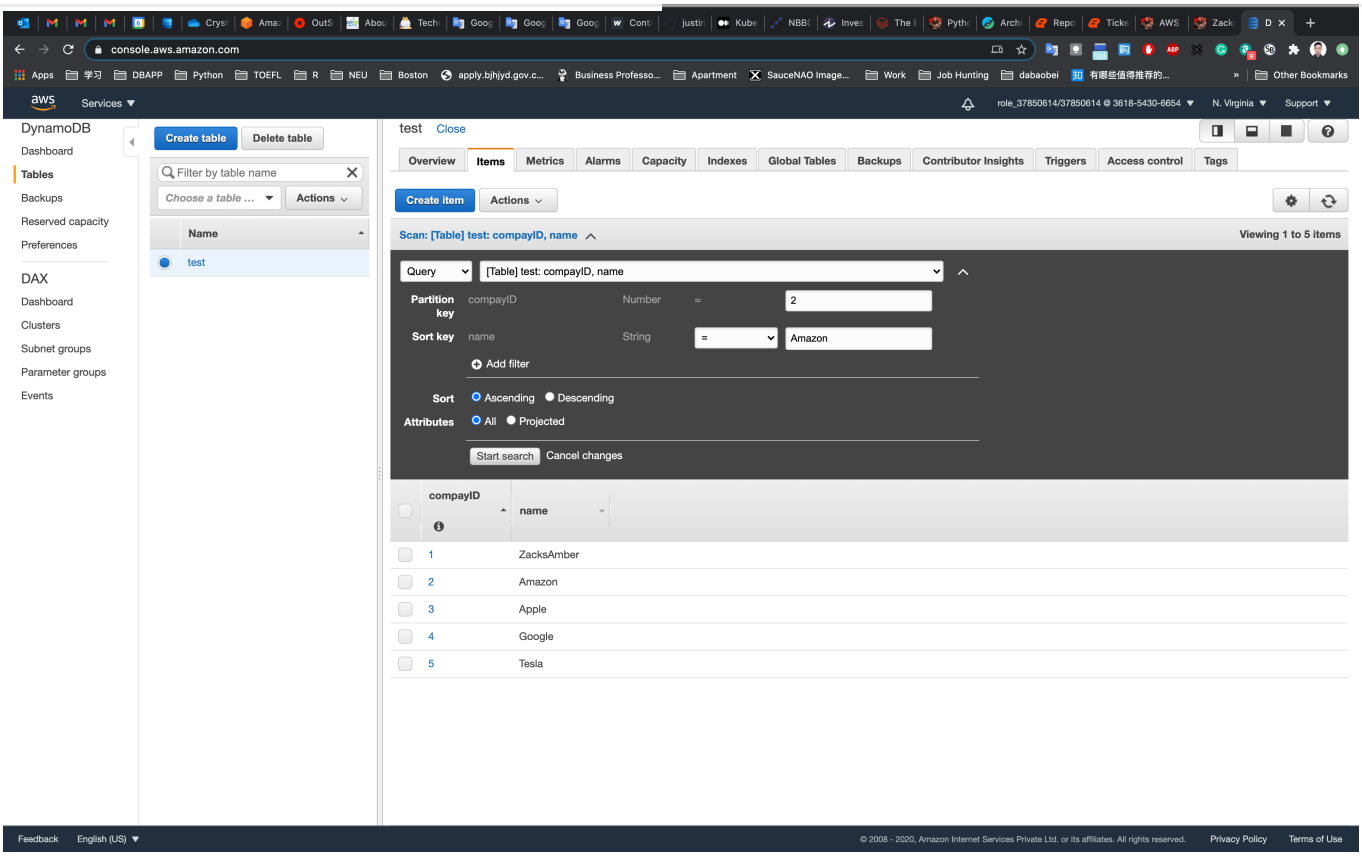
Click the drop-down list showing the `Scan` in `Items` Tab (located below the `Create item button`) and change it to `Query`.



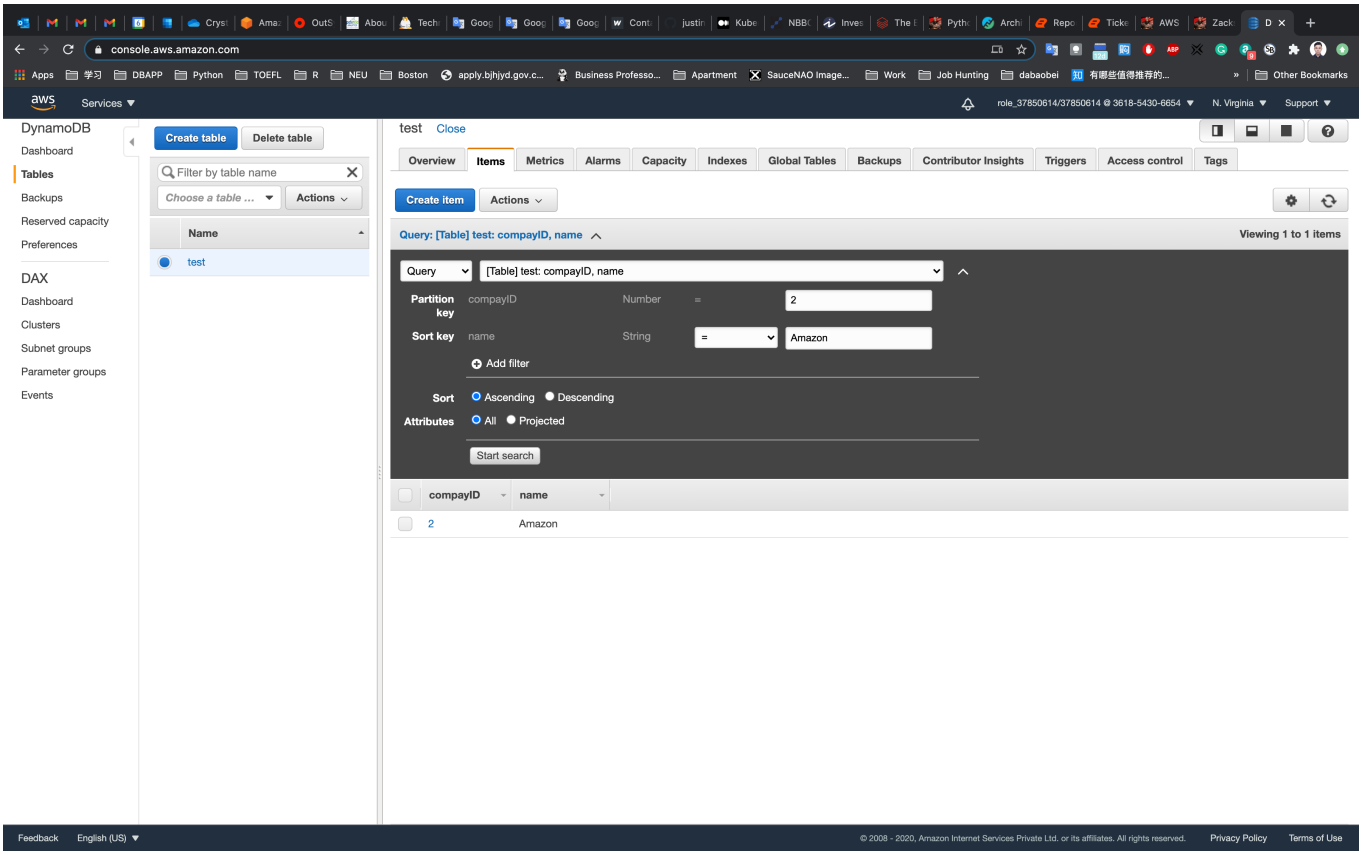
In the query window, enter the partition key and sort key which you want to search.

- Partition Key: `2`
- Sort Key: `Amazon`

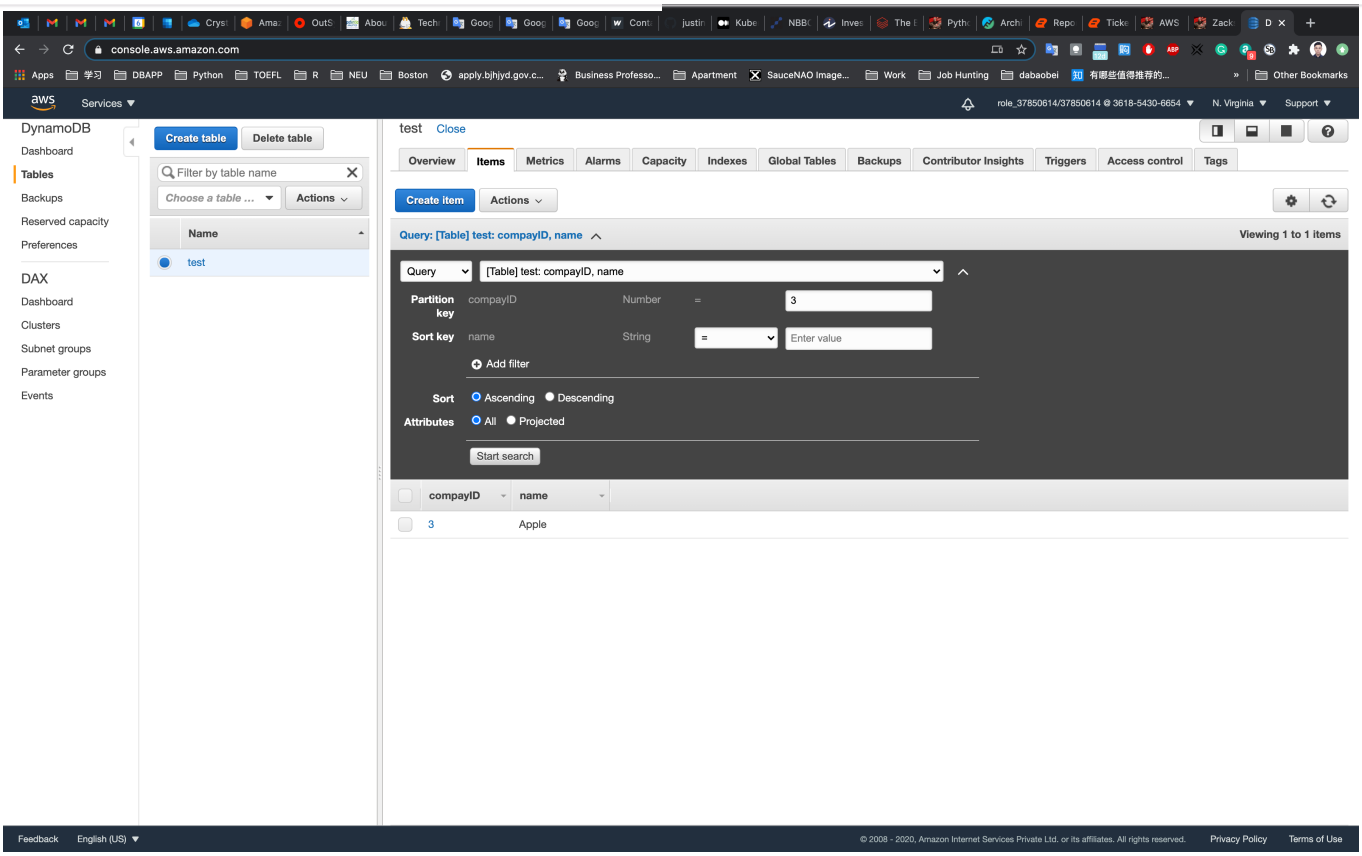
Click on `Start search`



You will be able to see a result table with your filtered records. A sample screenshot is given below:



ɔnly search for partition key



Completion and Conclusion

1. You have successfully created an Amazon DynamoDB table.
2. You have inserted multiple items into the table using a partition key and sort key.
3. You have successfully searched for items in the table using the partition and sort keys.

DynamoDB & Global Secondary Index

> https://play.whizlabs.com/site/task_details?lab_type=1&task_id=40&quest_id=35

Lab Details

1. This lab walks you through the steps to create a DynamoDB database and use Global Secondary Indexes in a case study.

Introduction

What is AWS DynamoDB?

- Definition
    - DynamoDB is a fast and flexible NoSQL database designed for applications that need consistent, single-digit millisecond latency at any scale. It is a fully managed database and it supports both document and key value data models.
    - It has a very flexible data model. This means that you don't need to define your database schema upfront. It also has reliable performance.
    - DynamoDB is a good fit for mobile gaming, ad-tech, IoT and many other applications.

DynamoDB Tables

DynamoDB tables consist of

- **Items** (Think of a row of data in a table).
- **Attributes** (Think of a column of data in a table).
- Supports **key-value** and **document data structures**.
- `Key` = the name of the data. `Value` = the data itself.
- `Document` can be written in **JSON**, **HTML** or **XML**.

DynamoDB - Primary Keys

- DynamoDB stores and retrieves data based on a **Primary key**.
- DynamoDB also uses **Partition keys** to determine the physical location data is stored.
- **If you are using a partition key as your Primary key, then no items will have the same Partition key.**
- **All items with the same partition key are stored together and then sorted according to the sort key value.**
- **Composite Keys (Partition Key + Sort Key) can be used in combination.**
- **Two items may have the same partition key, but must have a different sort key.**

What is an Index in DynamoDB

- In SQL databases, an index is a data structure which allows you to perform queries on specific columns in a table.
- You select the column that is required to include in the index and run the searches on the index instead of on the entire dataset.

In DynamoDB, two types of indexes are supported to help speed-up your queries.

- **Local secondary Index**.
- **Global Secondary Index**.

Local Secondary Index

- **Can only be created when you are creating the table.**
- **Cannot be removed, added or modified later.**
- It has the same partition key as the original table.
- Has a different Sort key.
- Gives you a different view of your data, organized according to an alternate sort key.
- Any queries based on Sort key are much faster using the index than the main table.

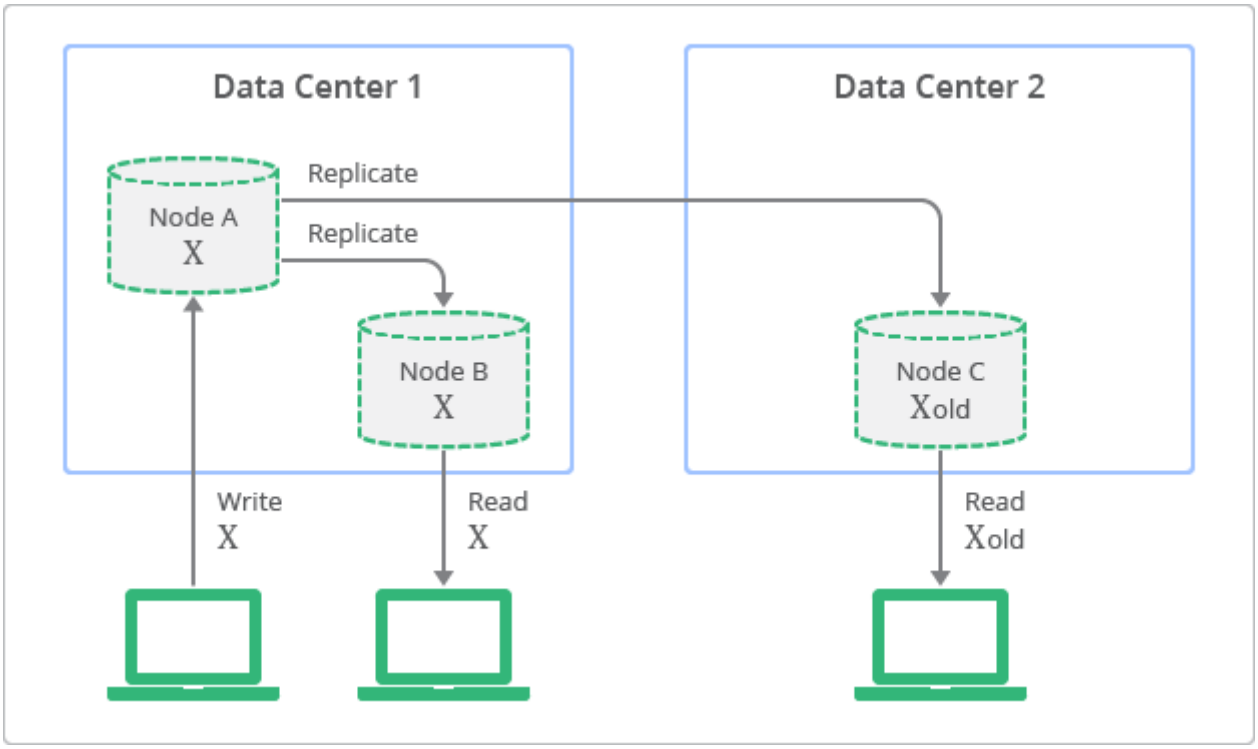
Global Secondary Index

- You can create a GSI on creation or you can add it later.
- Different partition key as well as different sort key.
- It gives a completely different view of the data.
- Speeds up the queries relating to this alternative Partition or Sort Key.

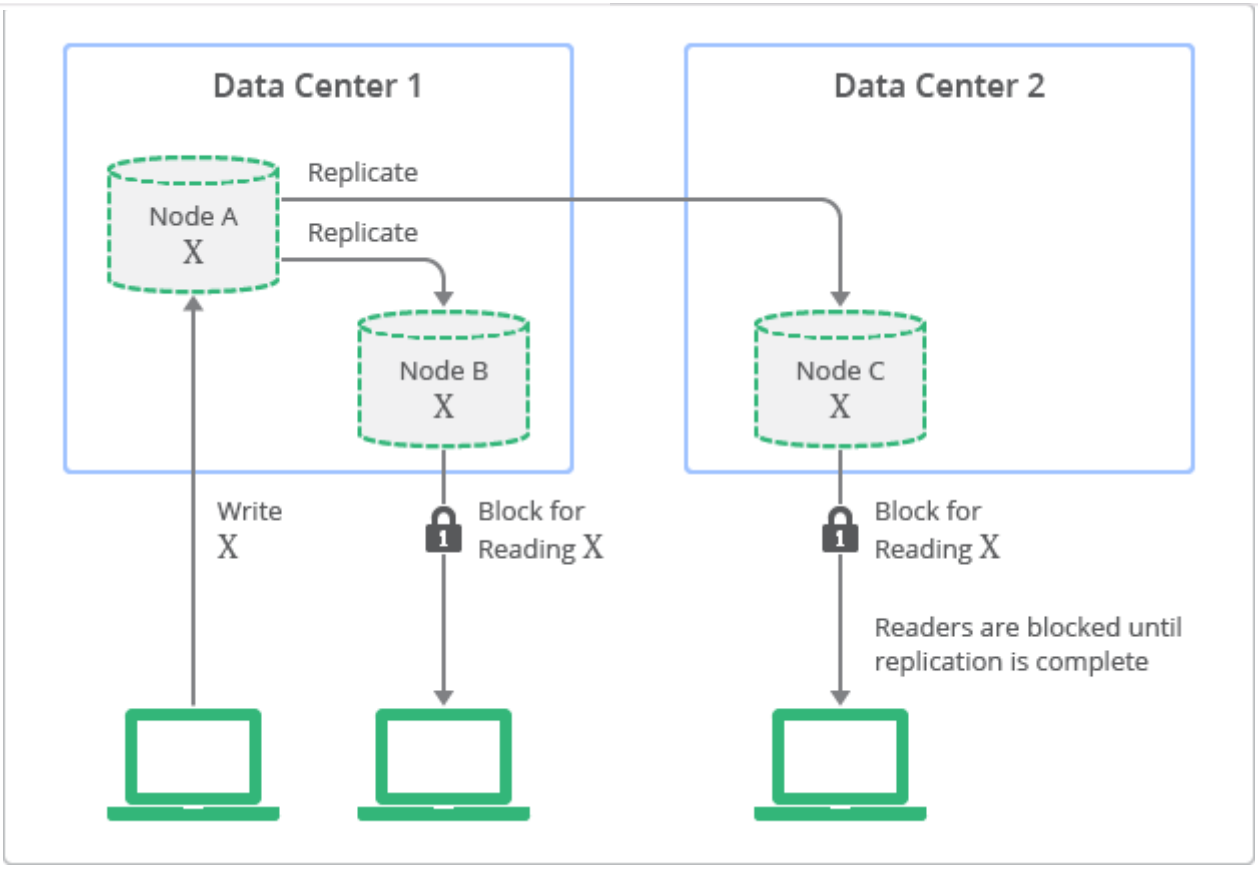
No.	LSI	GSI
1	Key = hash key and a range key	Key = hash or hash-and-range
2	Hash same attribute as that of the table. Rage key can be any scalar table attribute	The index hash key and range key (if present) can be any scalar table attributes.
3	For each hash key, the total size of all indexed items must be 10GB or less.	No size restrictions for global secondary indexes.

No.	LSI	GSI
4	Query over a single partition, as specified by the hash key value in the query.	Query over the entire table, across all partitions.
5	Eventual consistency or strong consistency.	Eventual consistency only.
6	Read and write capacity units consumed from the table.	Every global secondary index has its own provisioned read and write capacity units.
7	Query will automatically fetch non-projected attributes from the table.	Query can only request projected attributes. It will not fetch attributes from the table.

Eventual Consistency vs Strong Consistency



Eventual Consistency



Strong Consistency

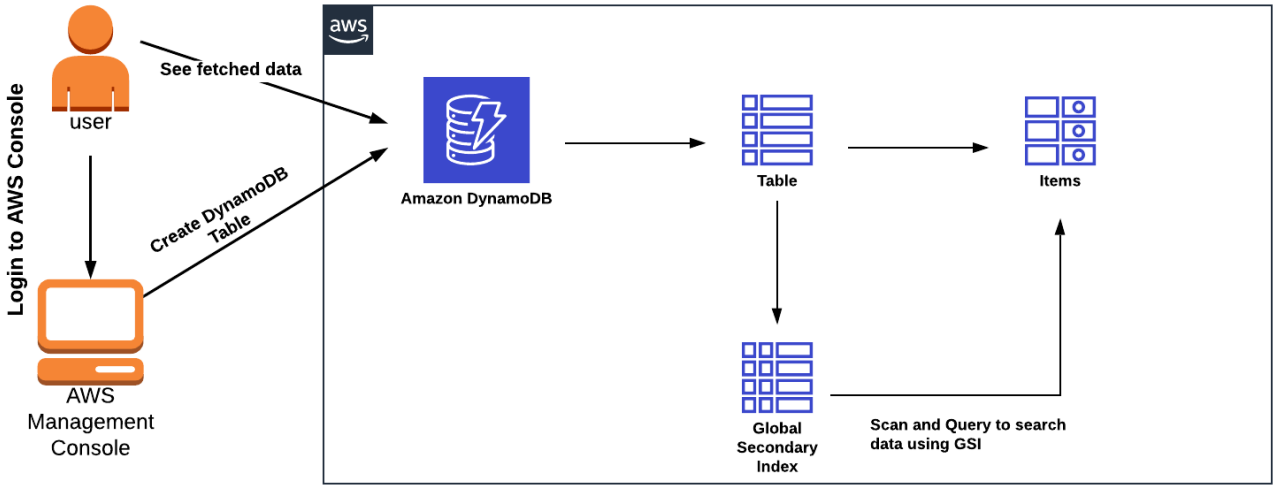
Lab Details

Case Study: Creating a Global Secondary Index

Let's get our hands dirty by creating a table and adding a GSI on the table.

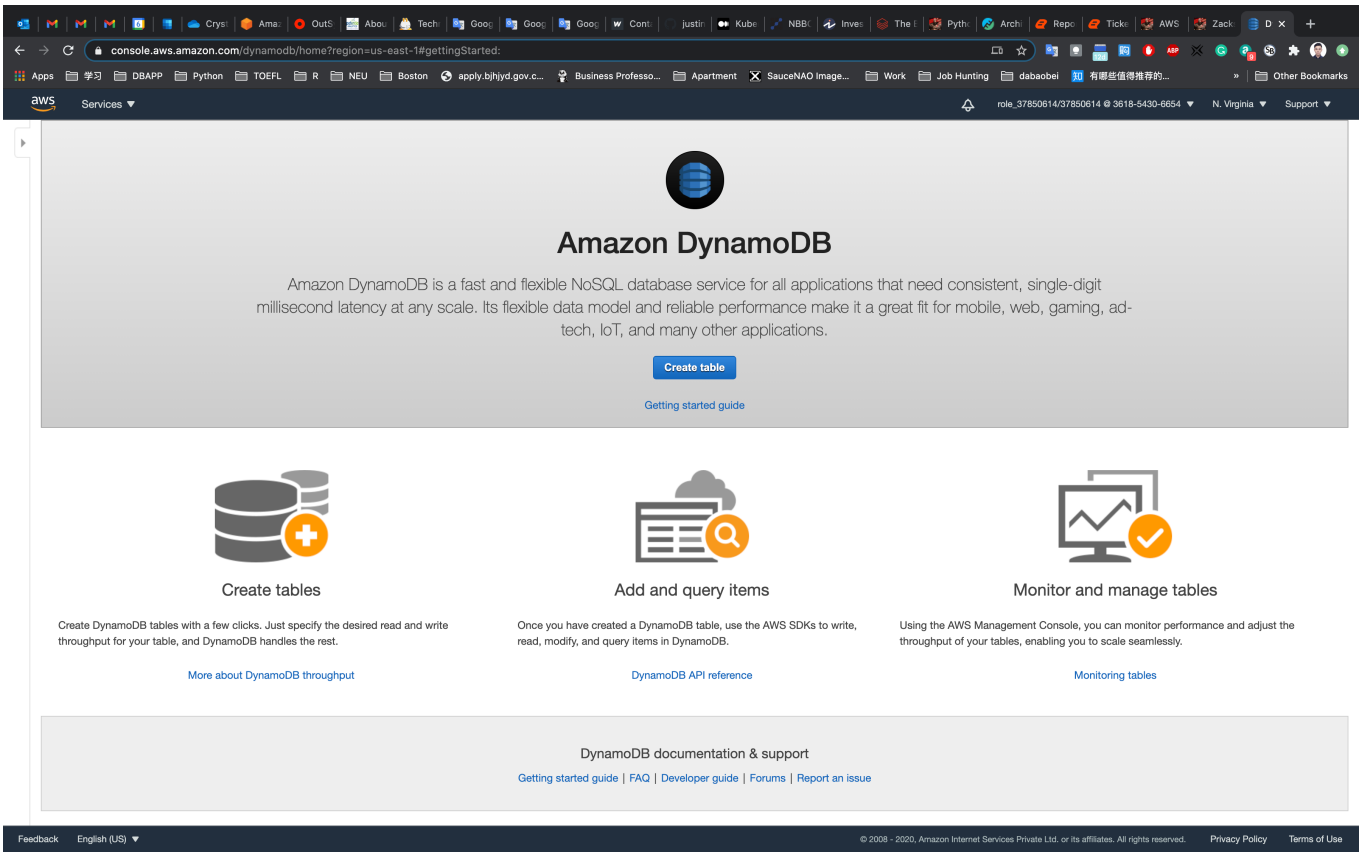
In this example, imagine we want to keep track of orders that were returned by our users. We'll store the date of the return in a `ReturnDate` attribute. We'll also add a global secondary index with a composite key schema using `ReturnDate` as the HASH key and `UserAmount` as the `RANGE` key.

Architecture Diagram



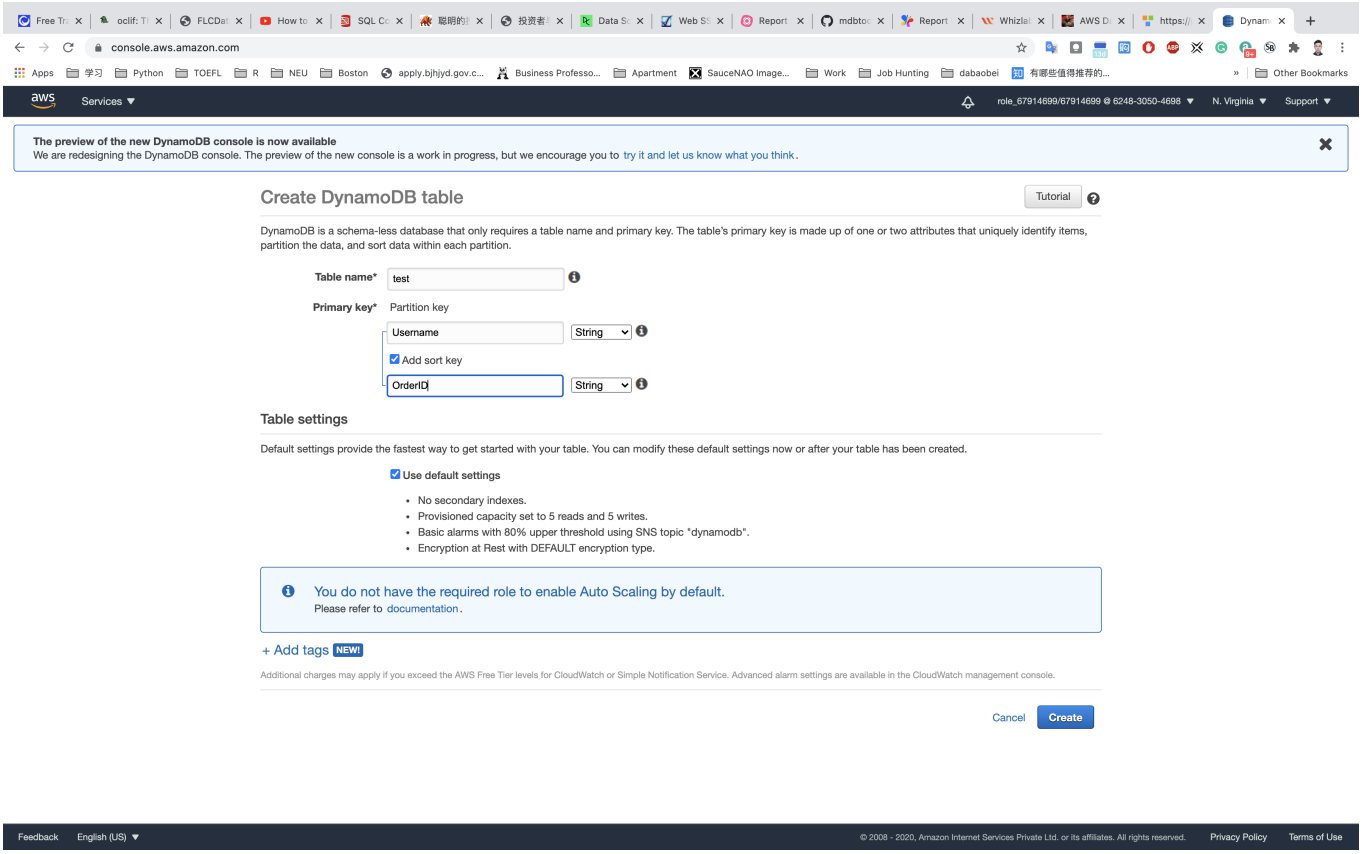
DynamoDB Configuration

Services -> DynamoDB

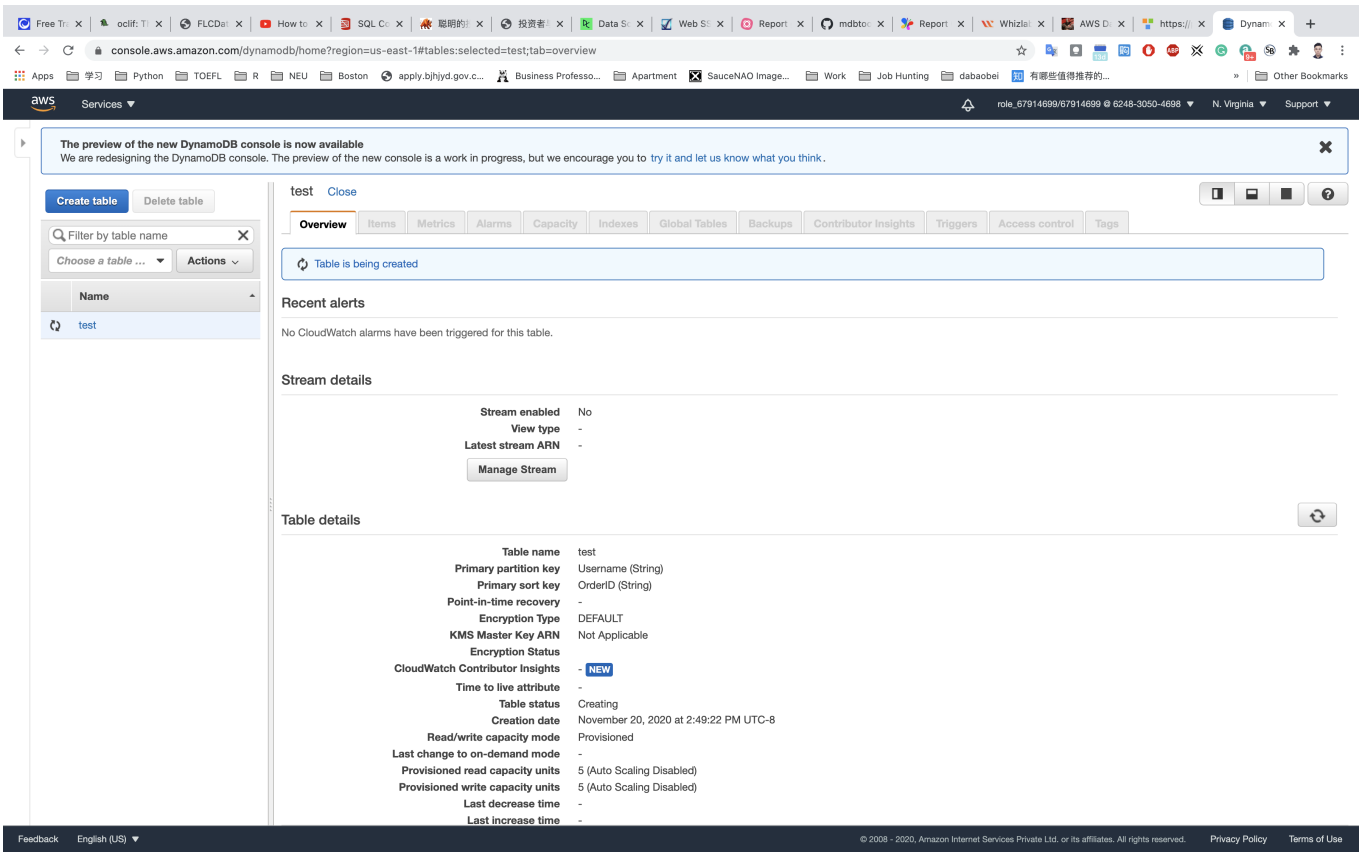


- Table Name: `test`
- Primary key(Partition key): `Username` and select `String`
- Add sort key: Enter **OrderID** in the respective field and select `String` .
- The combination of a Primary Key and a Sort Key uniquely identifies each item in a DynamoDB table.
- Keep all other settings as default.

Click on `Create`.



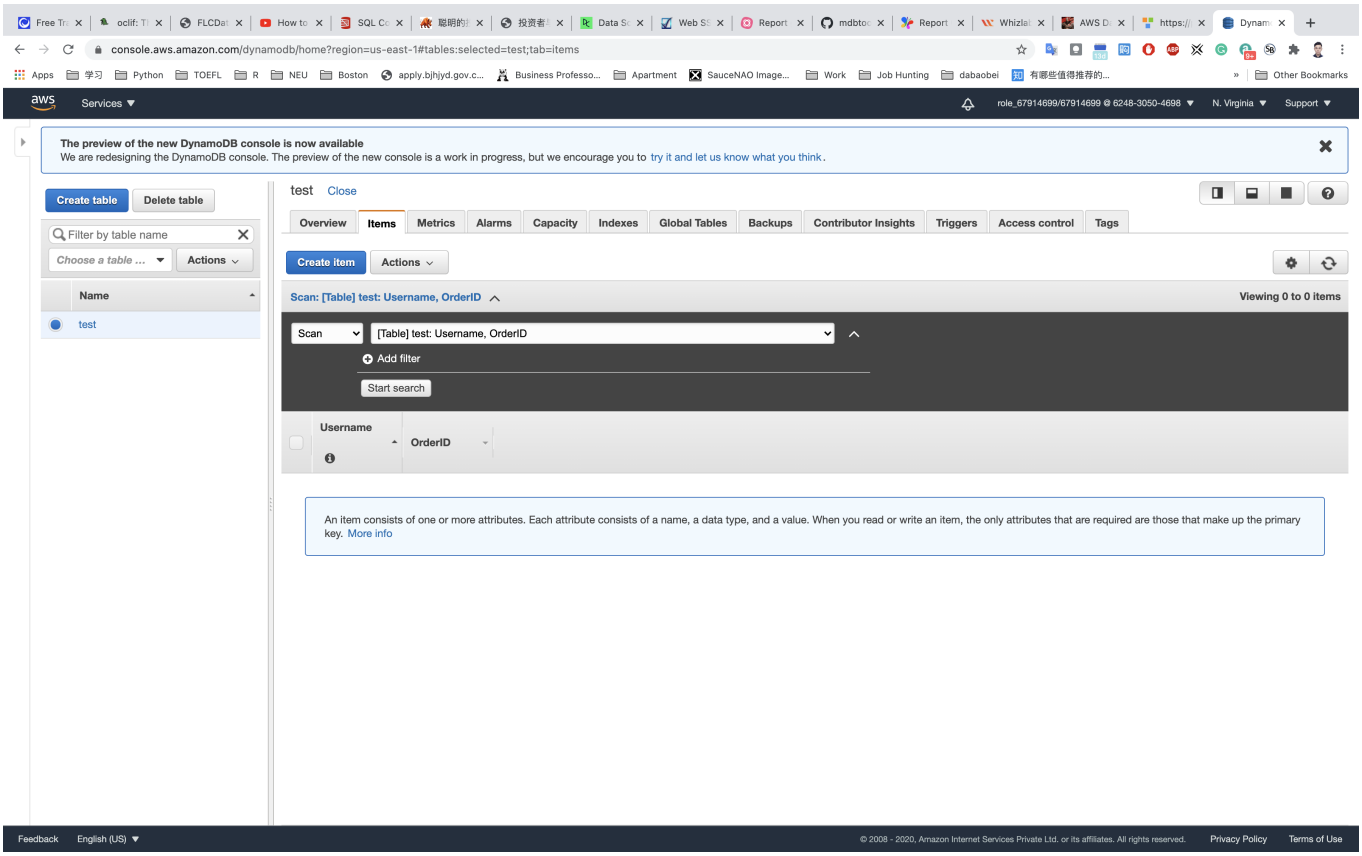
Your table will be created within 2-3 minutes.



Scan & Query DynamoDB

Create Item

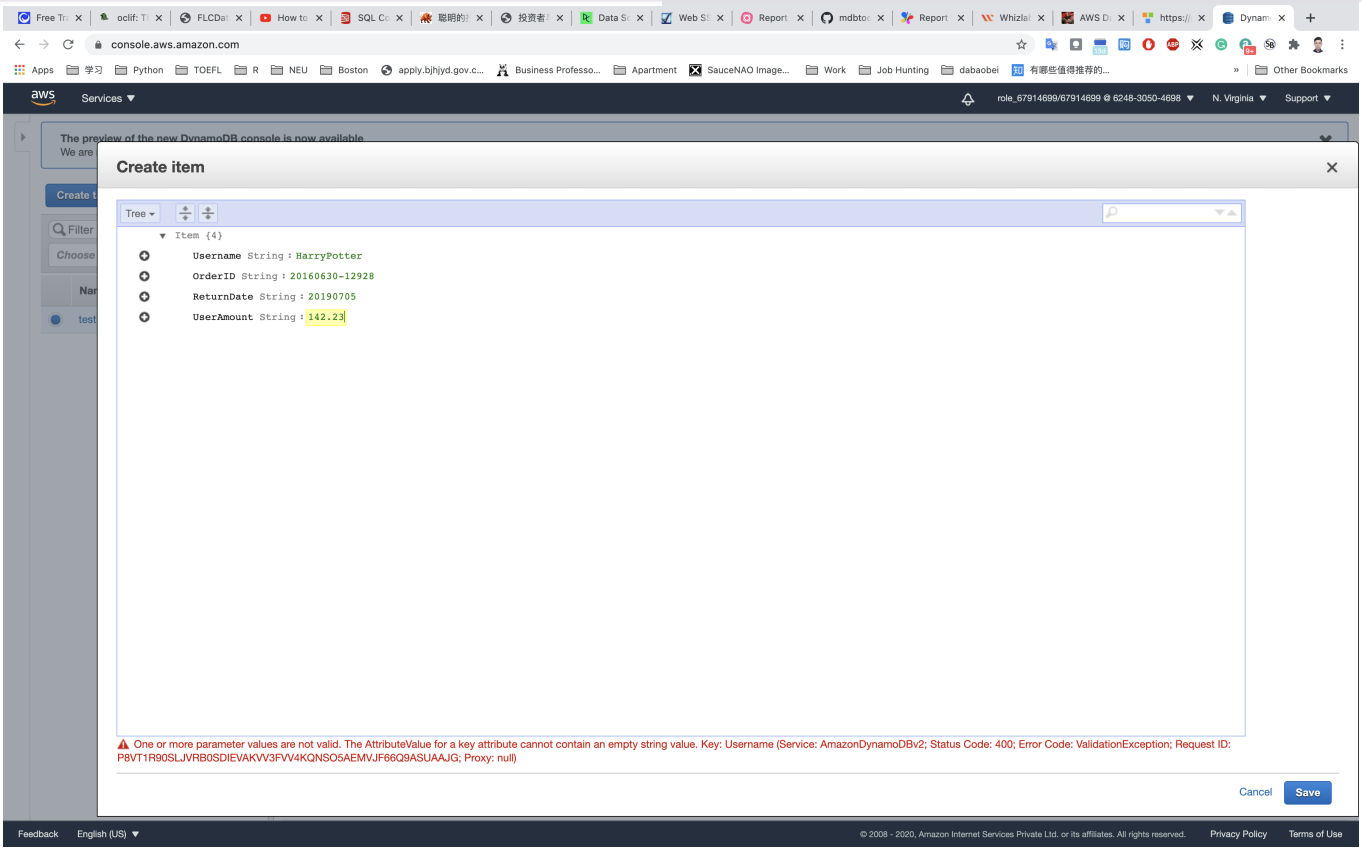
Select the `Items` tab next to the `overview` tab and click on `Create Item`.



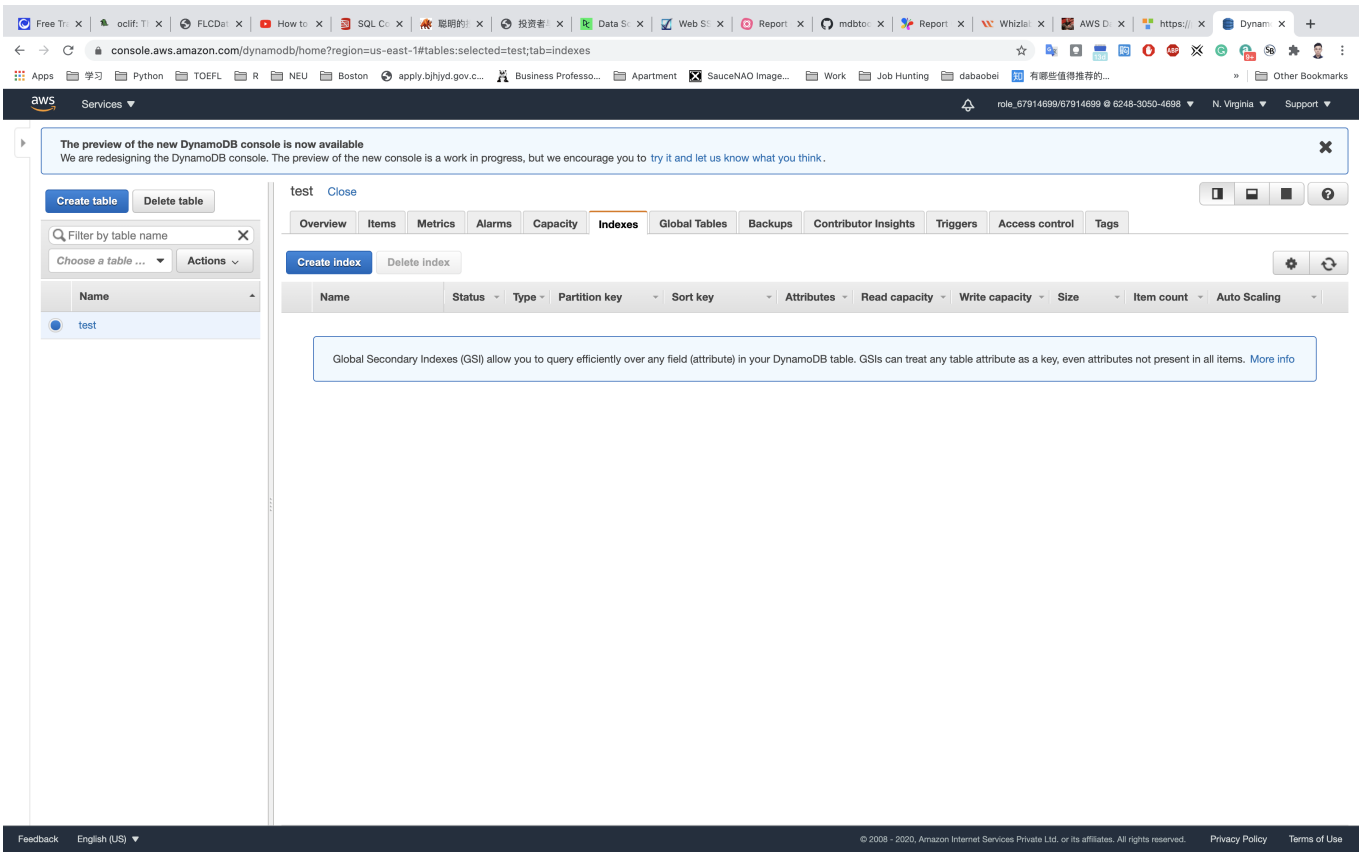
Once you select the create item, you'll see `Username` and `OrderID`, but we need 2 more attributes in our table. Click on `+` on `OrderID` then select `Append`. Choose String in the dropdown list.

Add two attributes: `ReturnDate` and `UserAmount`. Enter the string value in the field and click on `Save`.

- `UserName` : HarryPotter
- `OrderID` : 20160630-12928
- `ReturnDate` : 20190705
- `UserAmount` : 142.23

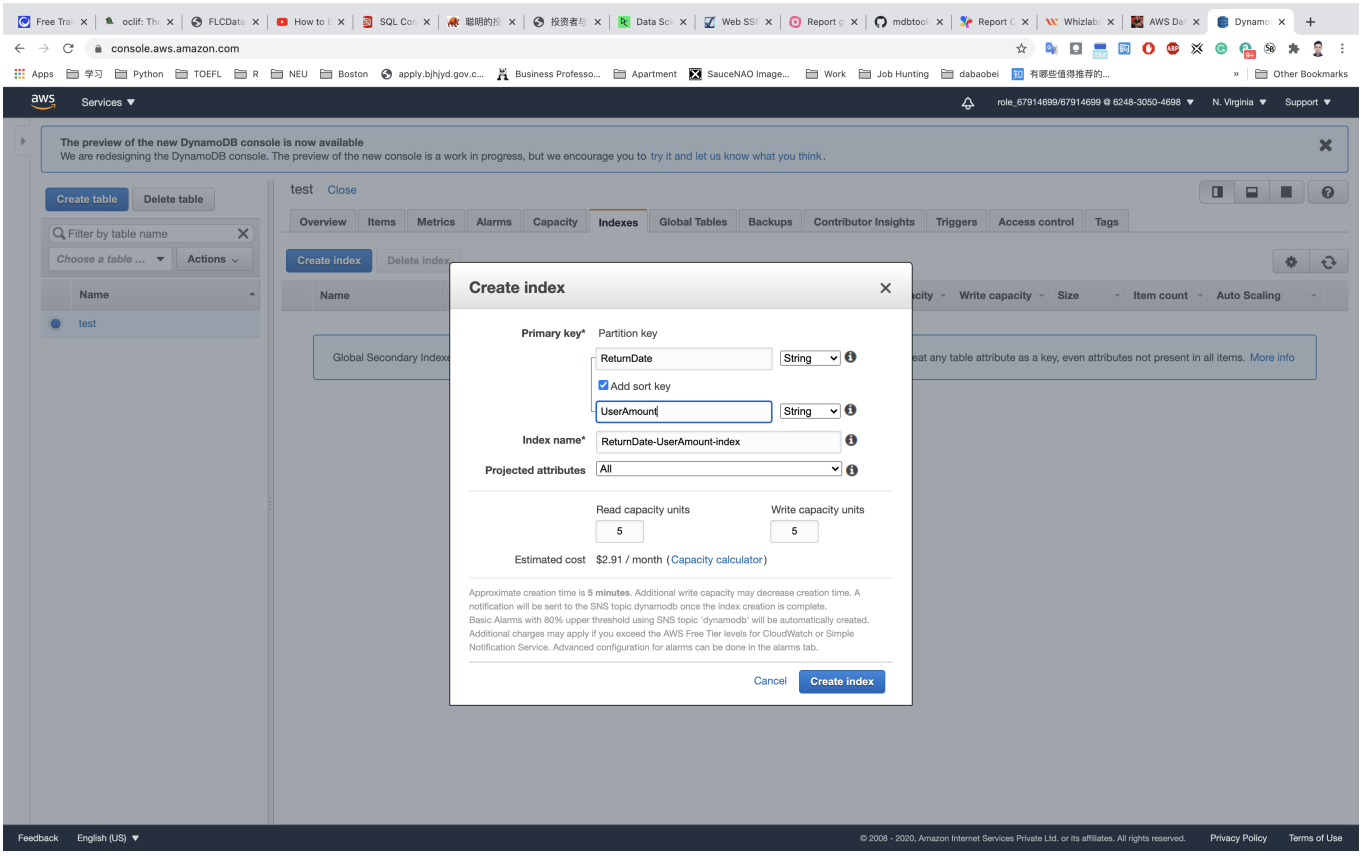


Navigate to the `Indexes` tab section (next to the capacity tab on the header) and click on `Create index`.



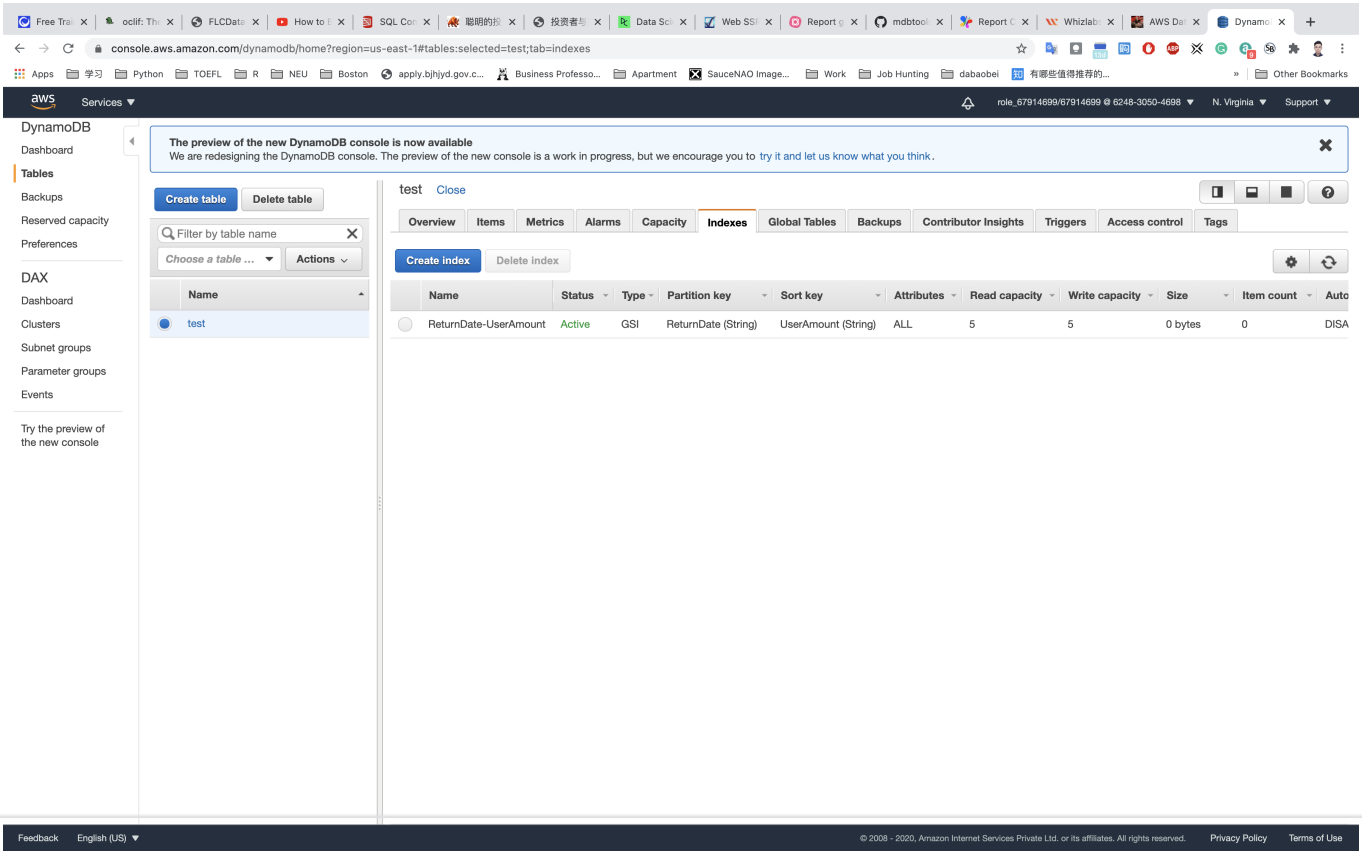
d the parameters `ReturnDate` and `UserAmount`. Click on `Create index` and wait until the index state changes to **Active**. The nomenclature for Index name is a combination of the

columns we selected, followed by Index. You can modify the index name as per the requirement.



Once the GSI is active, you can check the `Indexes` tab to confirm. Check the index type.

- Note: It will take 5-10 minutes to be active.



Move to the `Items` tab and click on `Create item`. Insert data into the table and click on `Save`.

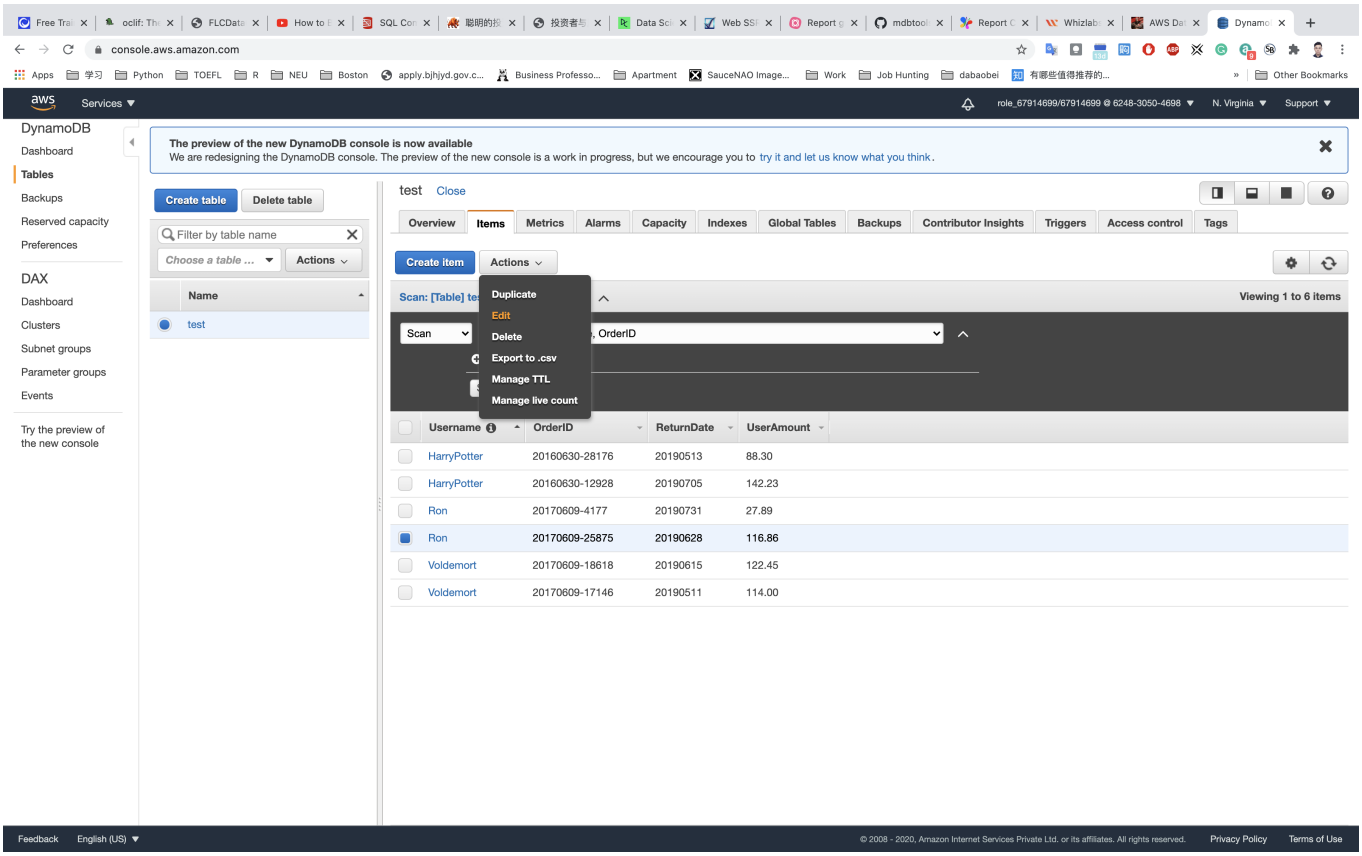
- Note: Refresh the console a few times if the newly-added attributes are not displayed in the field.

Add remaining values to the table.

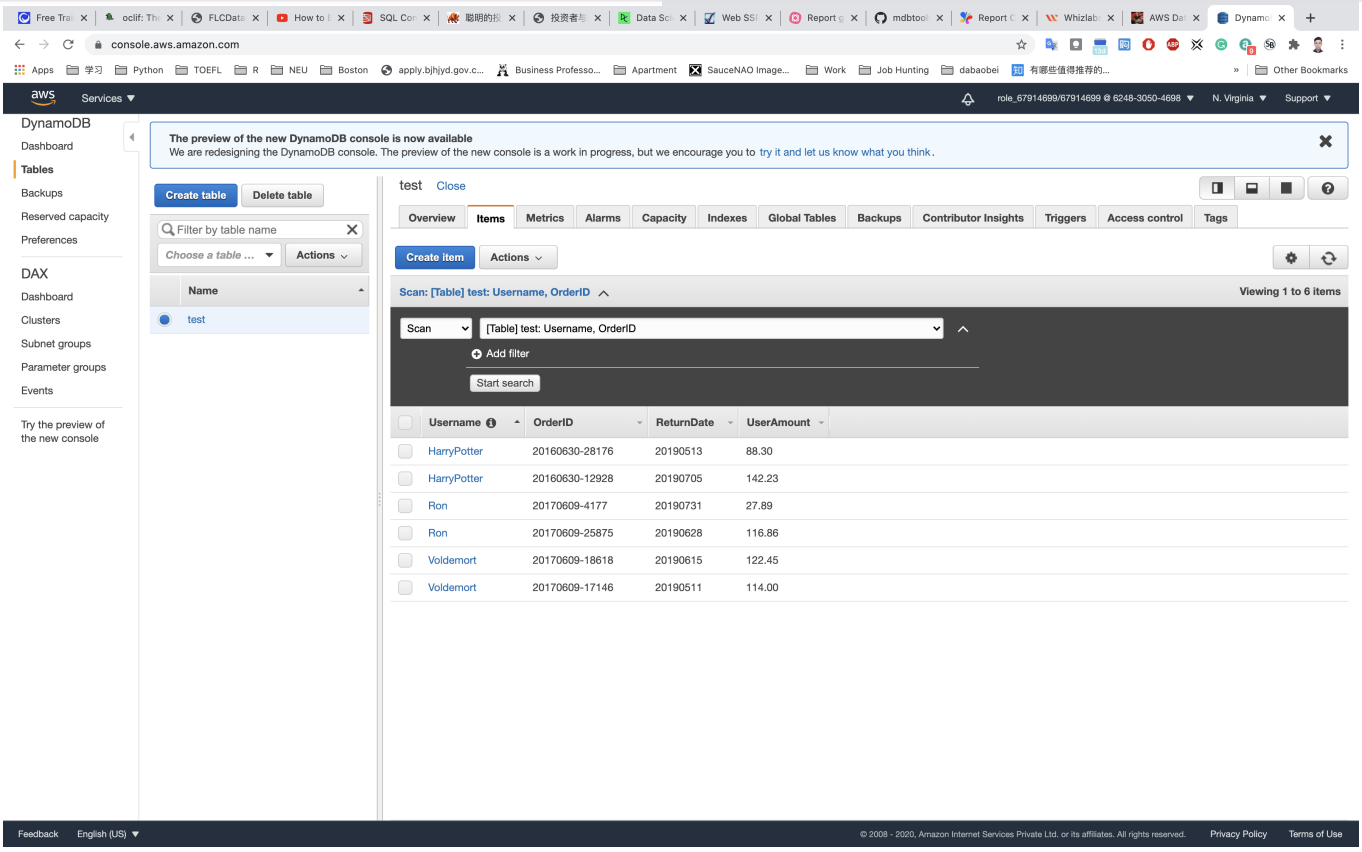
- `UserName` : HarryPotter
- `OrderID` : 20160630-28176
- `ReturnDate` : 20190513
- `UserAmount` : 88.30
- `UserName` : Ron
- `OrderID` : 20170609-25875
- `ReturnDate` : 20190628
- `UserAmount` : 116.86
- `UserName` : Ron
- `OrderID` : 20170609-4177
- `ReturnDate` : 20190731
- `UserAmount` : 27.89
- `UserName` : Voldemort
- `OrderID` : 20170609-17146
- `ReturnDate` : 20190511

- `UserAmount` : 114.00
- `UserName` : Voldemort
- `OrderID` : 20170609-18618
- `ReturnDate` : 20190615
- `UserAmount` : 122.45

In case you added a wrong value, you can edit with the `edit` option on the column.



Once you have added all the data in the table, please review it.

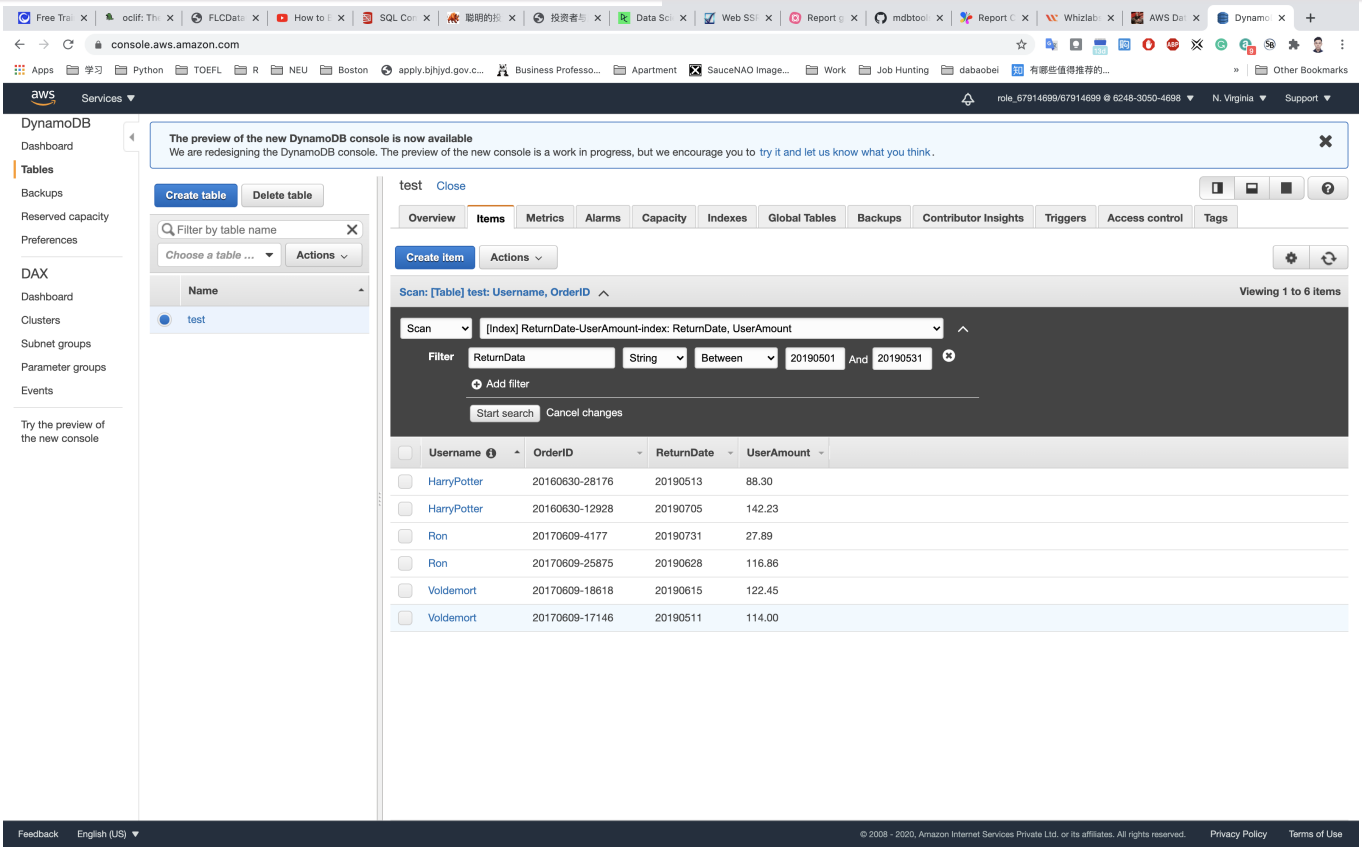


Use Global Secondary Index to Fetch Data

1. Now with the help of **GSI** we will try to fetch the data from the table, avoiding a full scan. This will lead to better performance and saving resources. We'll be fetching data and adding filter conditions on the return date.

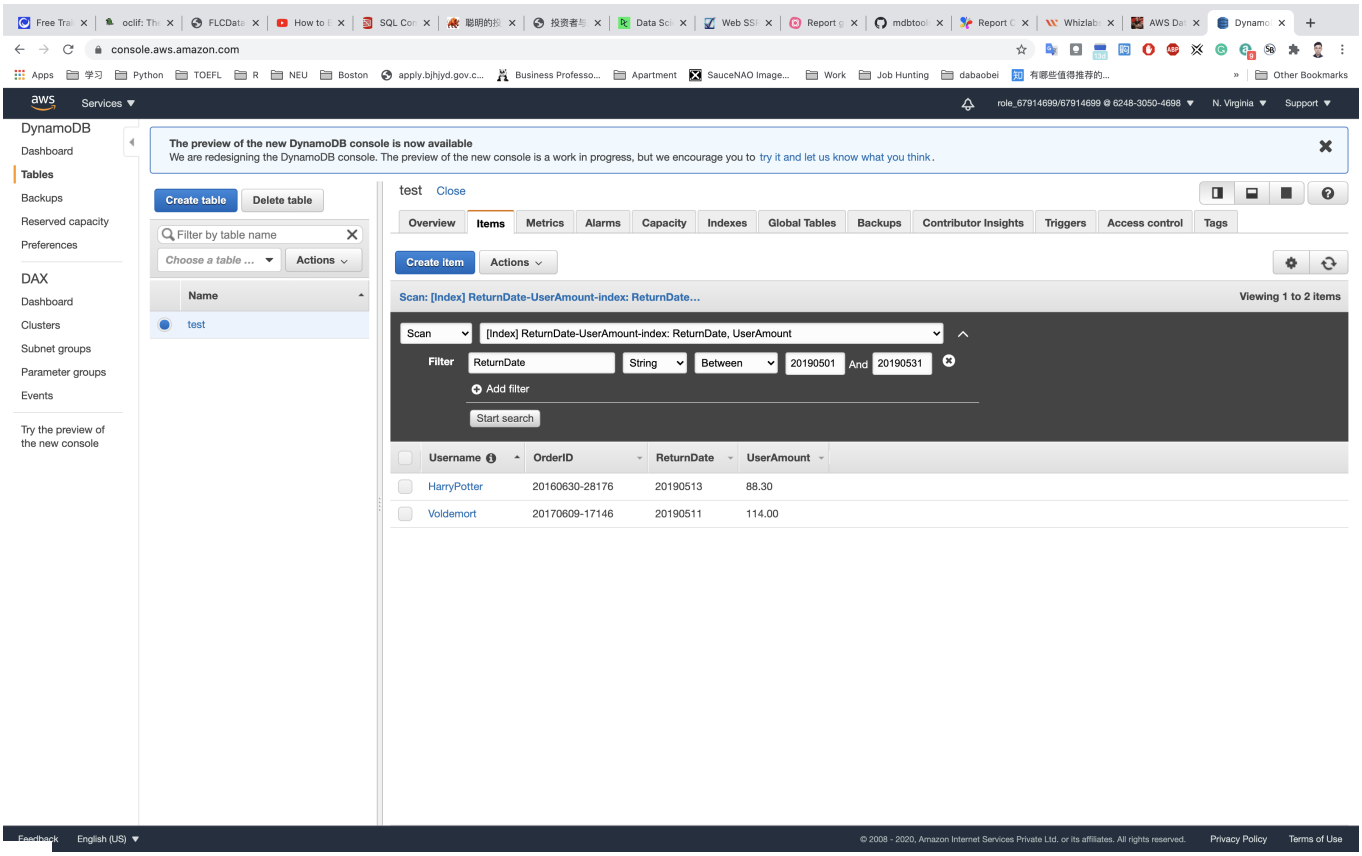
2. Lets try with the **Scan** option to search for data. We need a **ReturnDate** (PartitionKey) and we need to check which users returned an item on that date. To do so, we can use the sort key to qualify the amount as per the requirement.

    - Select the "**Scan**" option (which is in the left upper corner).
    - In this example, we are trying to fetch the users who returned their orders in the month of May.
    - Select the "**Index**" option which we created and add a filter condition on "**ReturnDate**". Select data type "**String**" because our attributes are a "String" data type, and then select the clause "**Between**"- `20190501` and `20190531`

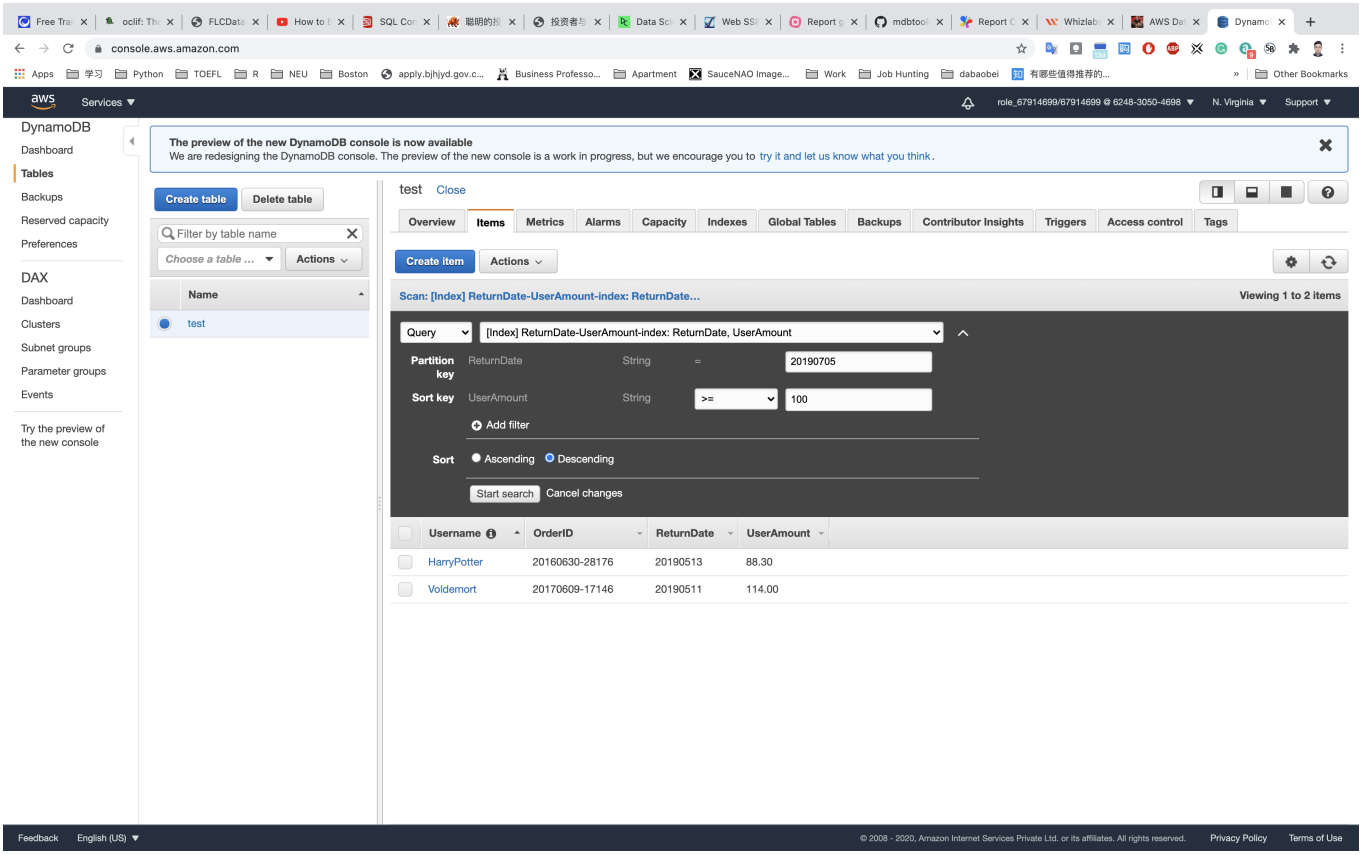


Note: If the Global secondary index is not showing in the Scan list, Please wait another 5 to 10 minutes and reload the entire page. This is an AWS side delay.

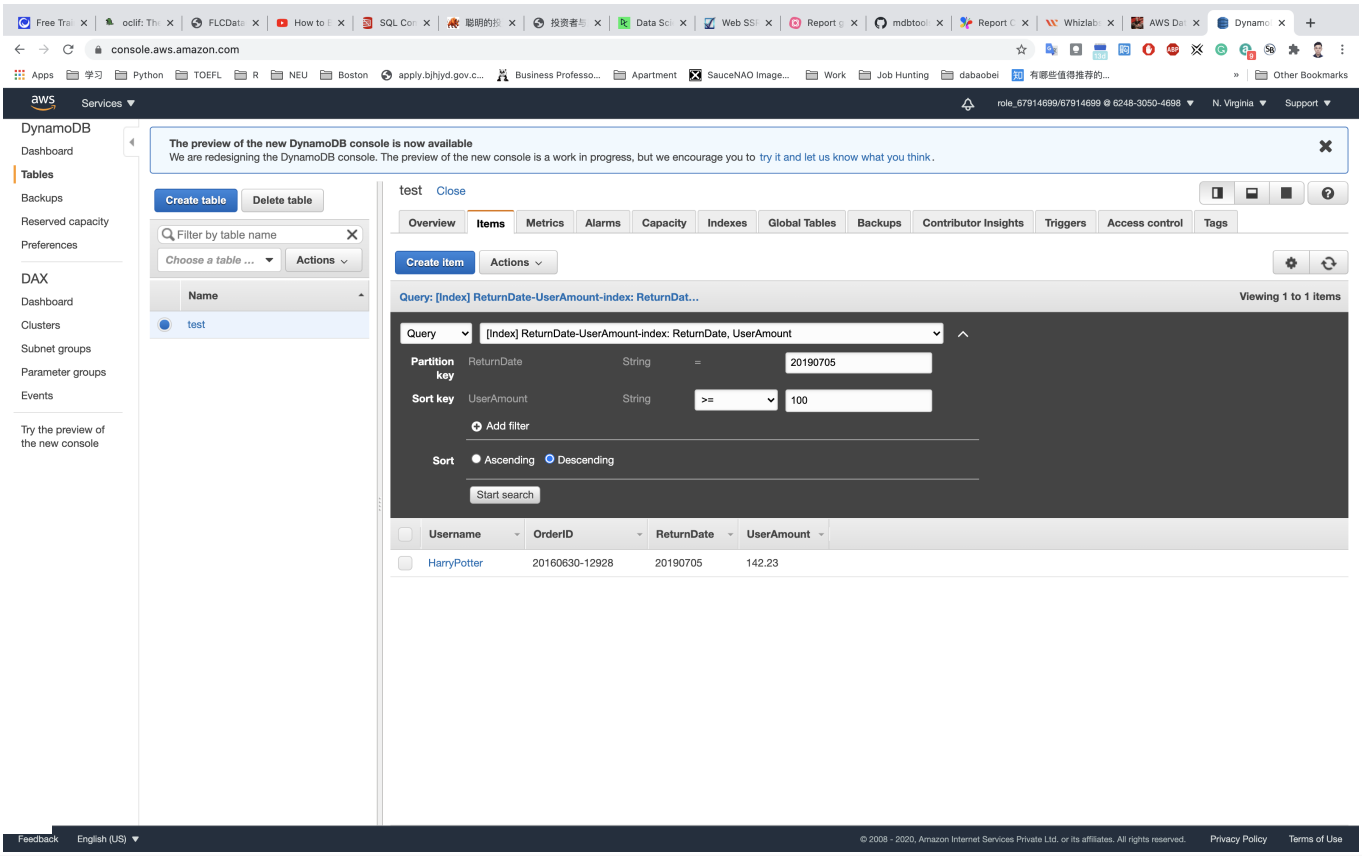
Click on `Start search`



3. Lets try with the **Query** option to search for some data. We need a **ReturnDate** (PartitionKey) and we need to check which users returned an item on that day. To do so, we'll use the sort key to qualify the amount, as per the requirement.



Click on `Start search`



This global secondary index allows us to quickly find all the returns entered on various dates, which would normally require full table scans.

Completion and Conclusion

1. You have created a DynamoDB table with Global Secondary Indexes.
2. You have successfully fetched data using Global Secondary Indexes.

One more thing

Best Practices for Querying and Scanning Data

Performance Considerations for Scans

In general, `Scan` operations are less efficient than other operations in DynamoDB. A `Scan` operation always scans the entire table or secondary index. It then filters out values to provide the result you want, essentially adding the extra step of removing data from the result set.

If possible, you should avoid using a `Scan` operation on a large table or index with a filter that removes many results. Also, as a table or index grows, the `Scan` operation slows. The `Scan` operation examines every item for the requested values and can use up the provisioned throughput for a large table or index in a single operation. For faster response times, design your tables and indexes so that your applications can use `Query` instead of `Scan` . (For tables, you can also consider using the `GetItem` and `BatchGetItem` APIs.)

Alternatively, design your application to use `Scan` operations in a way that minimizes the impact on your request rate.

DynamoDB Global Table & Load Data from S3

> DynamoDB

Lab Details

- DynamoDB Tables
- DynamoDB Two Read Models
- Cost vs. Performance - Two Capacity Modes
- DynamoDB Global Tables

Task Details

- Create a DynamoDB Table with a Local Secondary Index
- Make a DynamoDB replica in another region, which is high availability
- Load data from S3 with COPY command

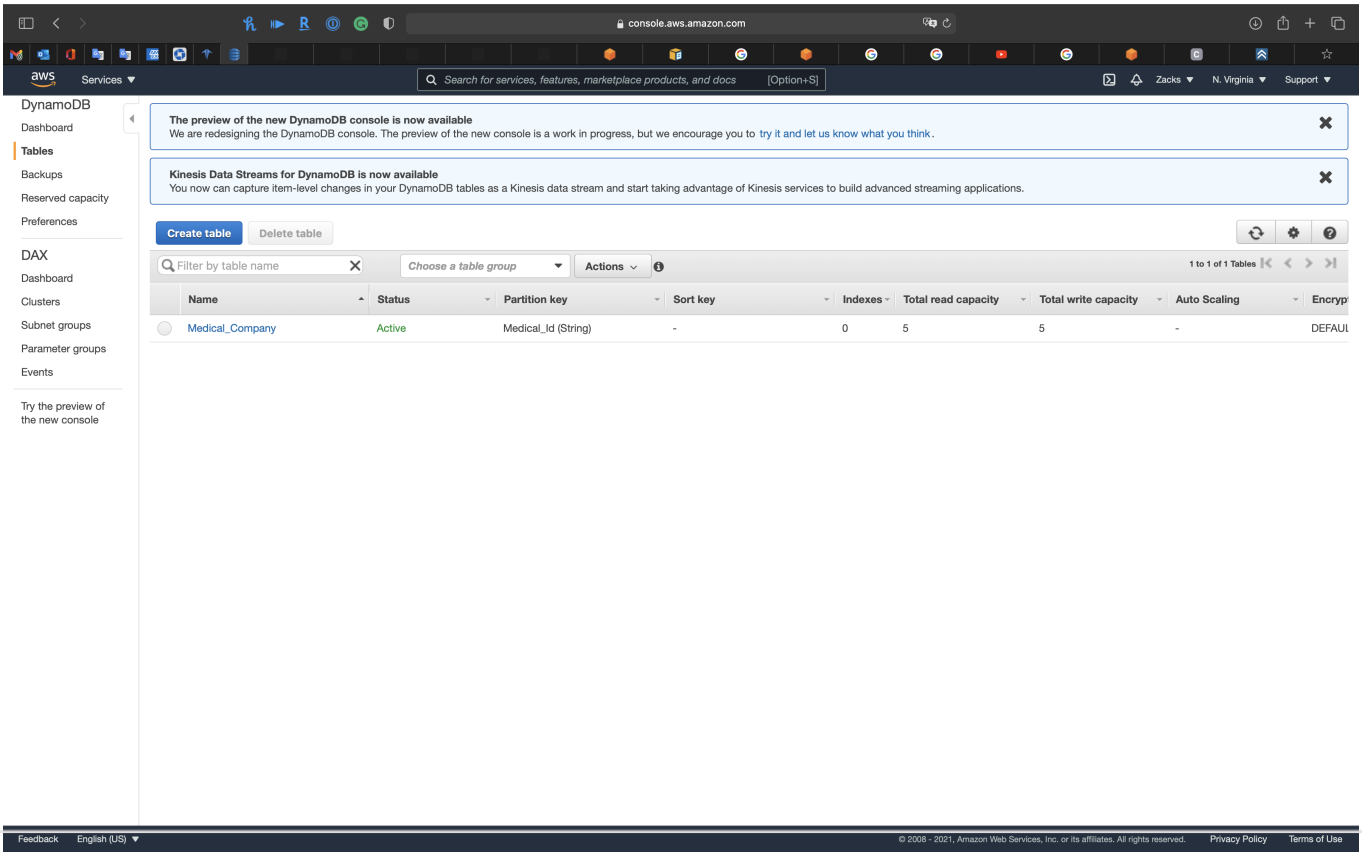
DynamoDB Configuration

Services -> DynamoDB

Create Table

Click on `Tables` navigation

Click on `Create table`



Create DynamoDB table

- Table name: `dynamodb_lab`
- Primary key
  - Partition key: `String`
    - `customer_id`
  - Check `Add sort key` : `String`
    - `order_timestamp`

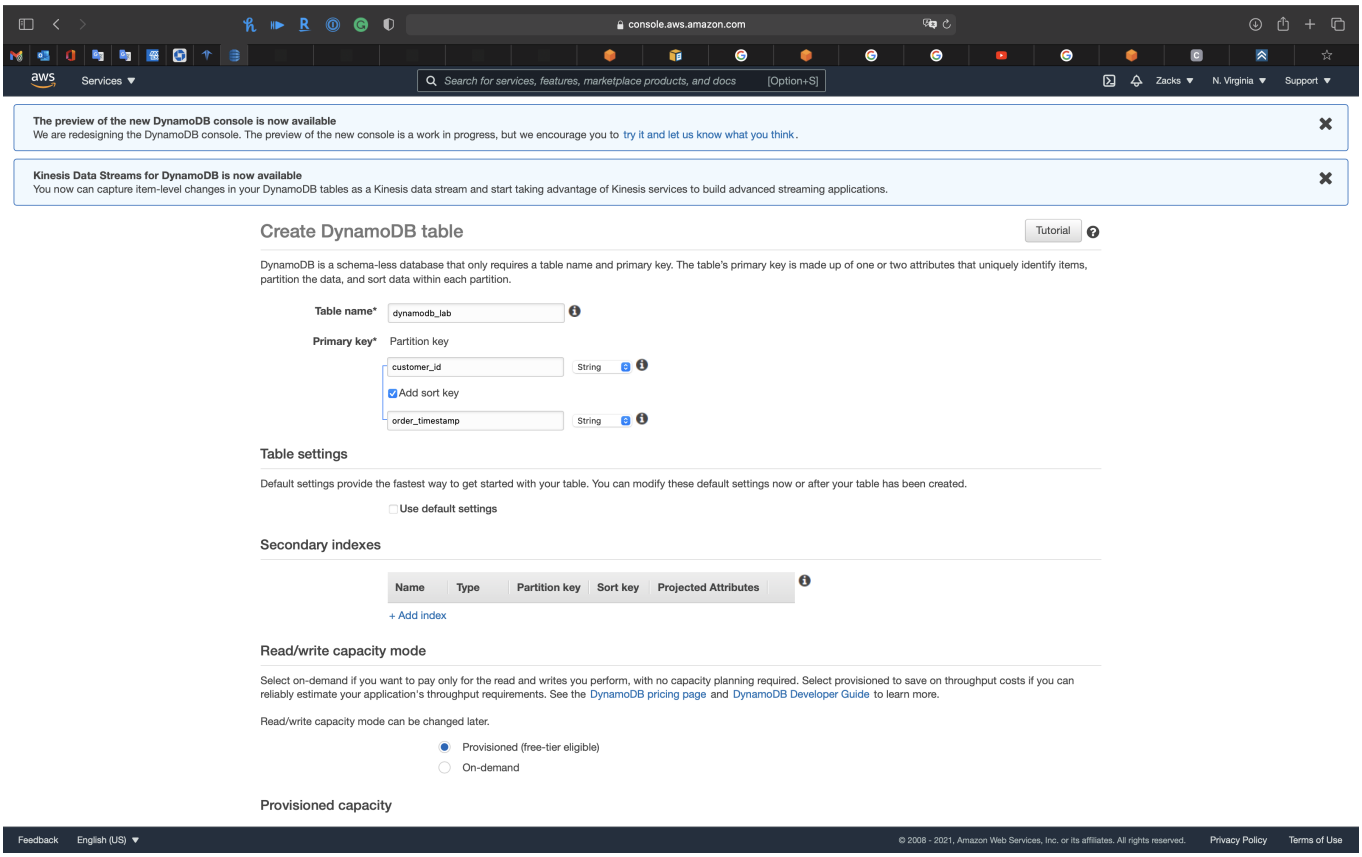


Table settings

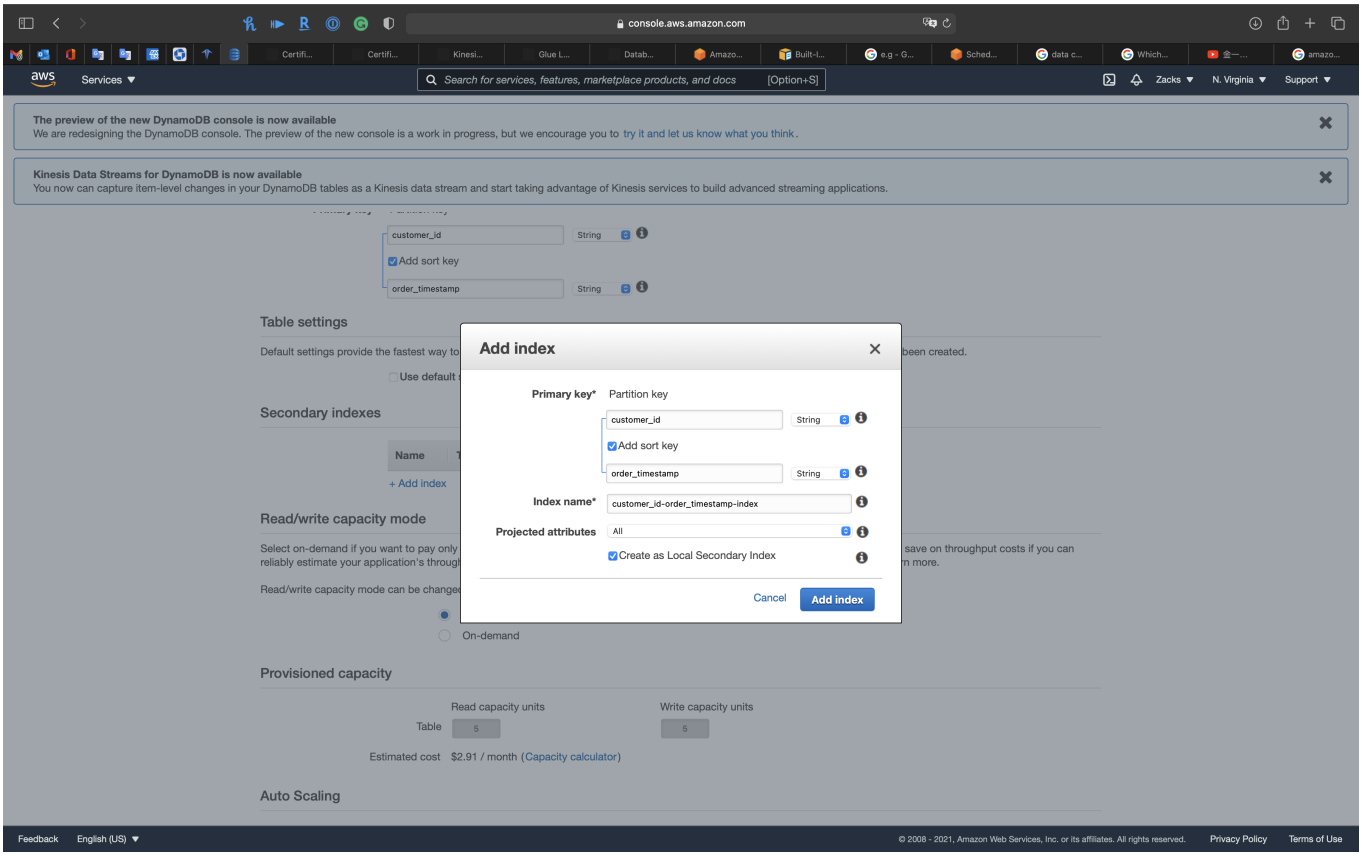
Uncheck `Use default settings`

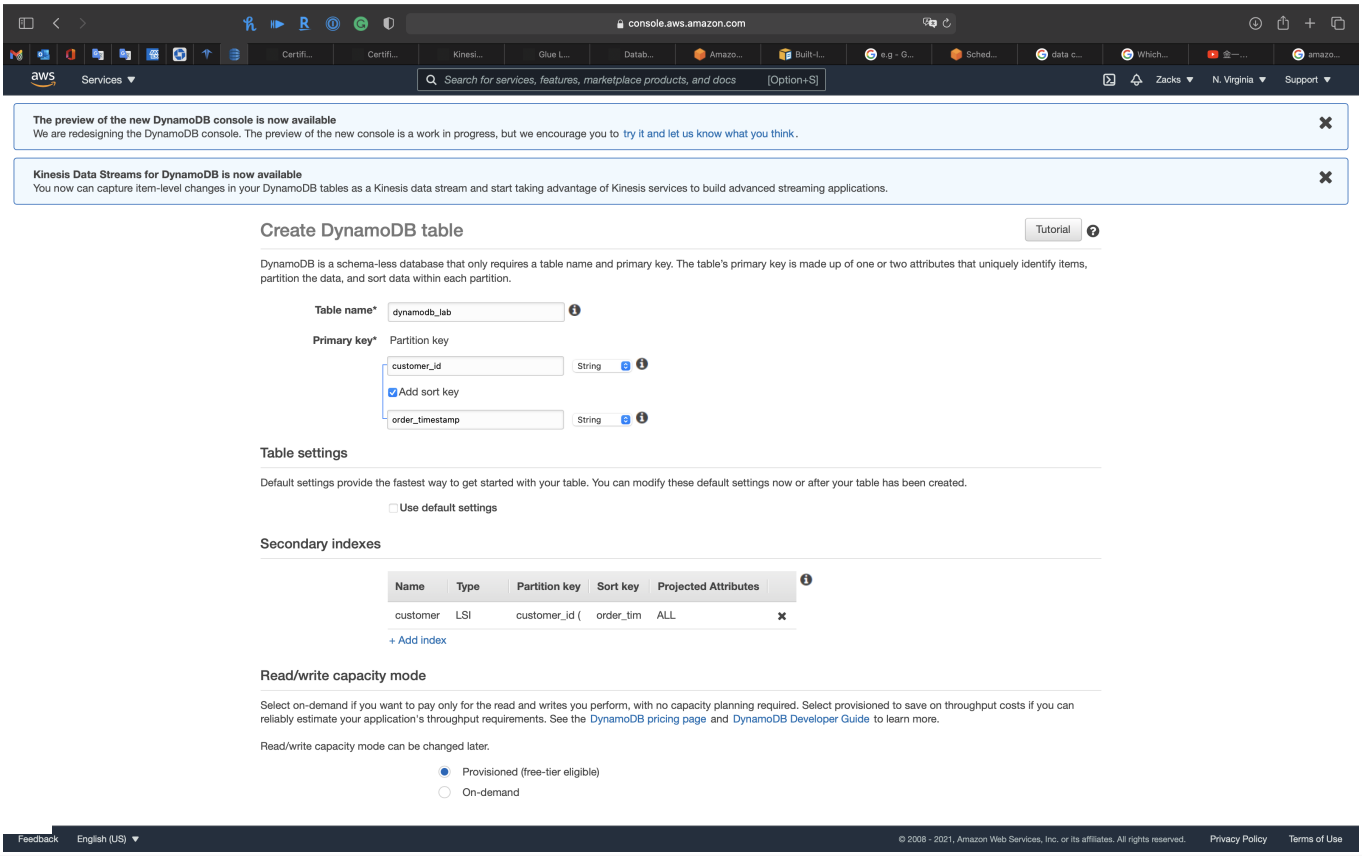
Secondary indexes

Click on `+ Add index`

- Partition key: `string`
  - `customer_id`

- Add sort key: `string`
    - `order_timestamp`
- Projected attributes: `All`
- Check `Create as Local Secondary Index`





> Projected attributes: Projected attributes are attributes stored in the index, and can be returned by queries and scans performed on the index. Local secondary index queries can also return attributes that are not projected by fetching them from the table. Global secondary index queries can only return projected attributes. Note, that projected attributes incur write and storage costs. For more information and performance tuning tips, see Projections.

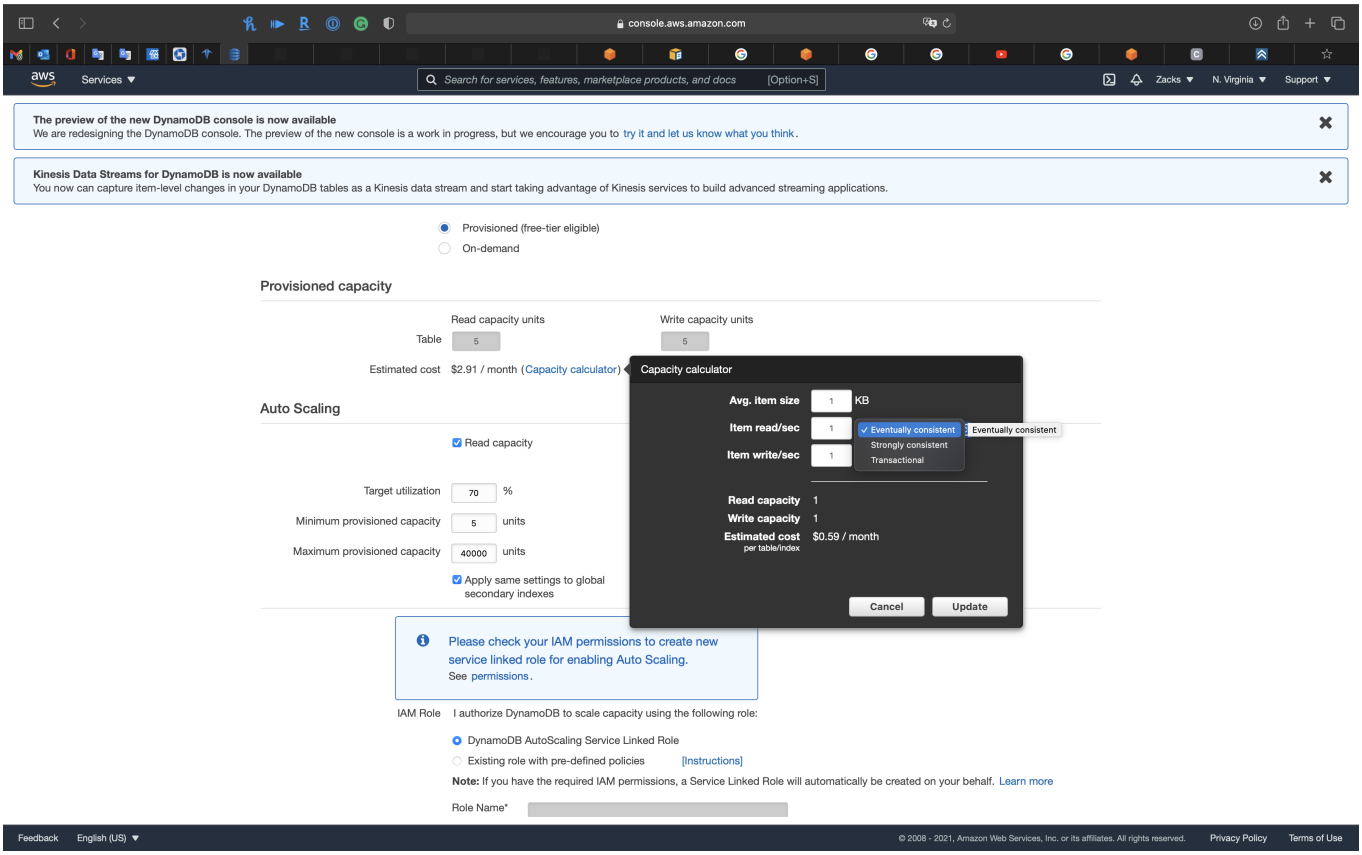
Read/write capacity mode

Check `Provisioned (free-tier eligible)`

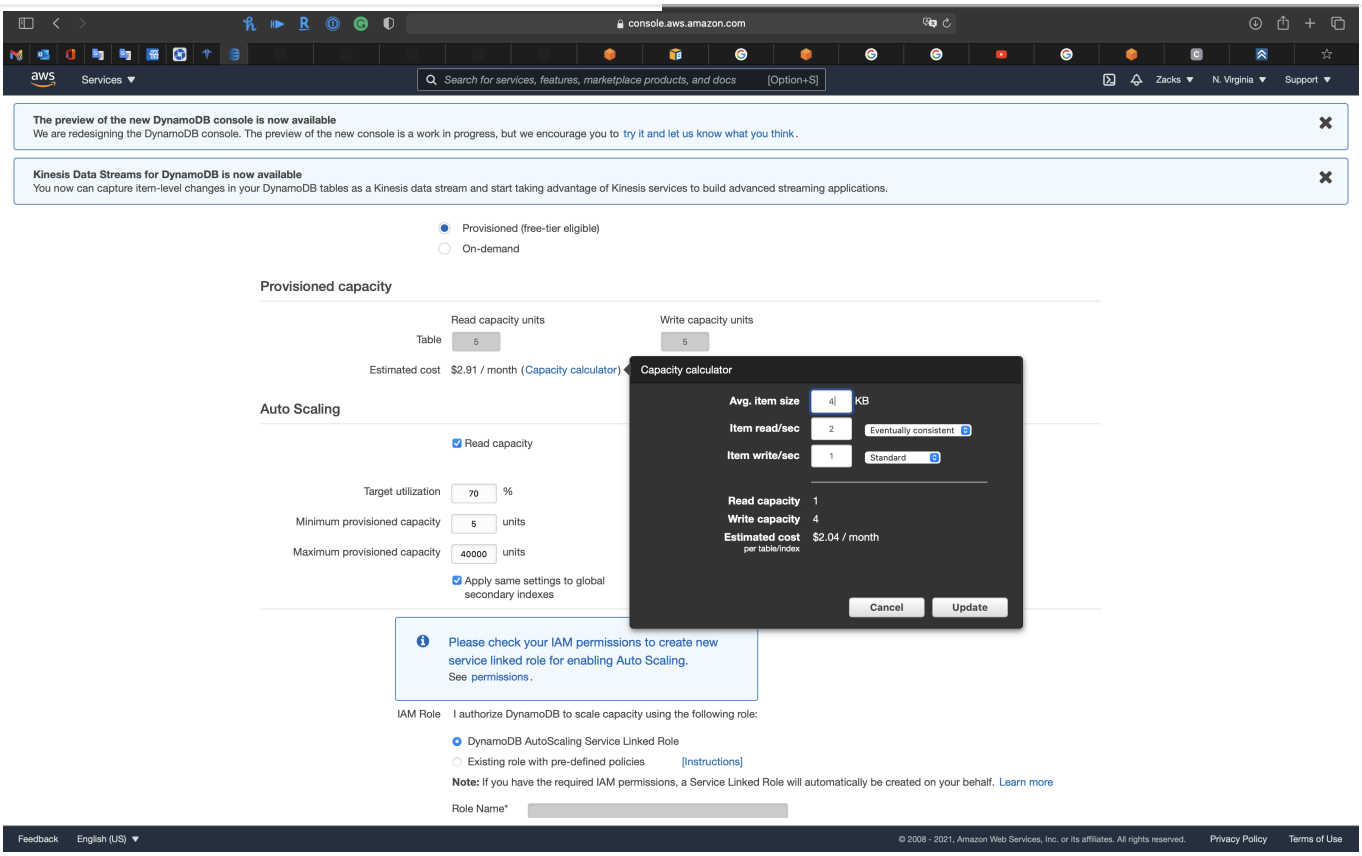
Provisioned capacity

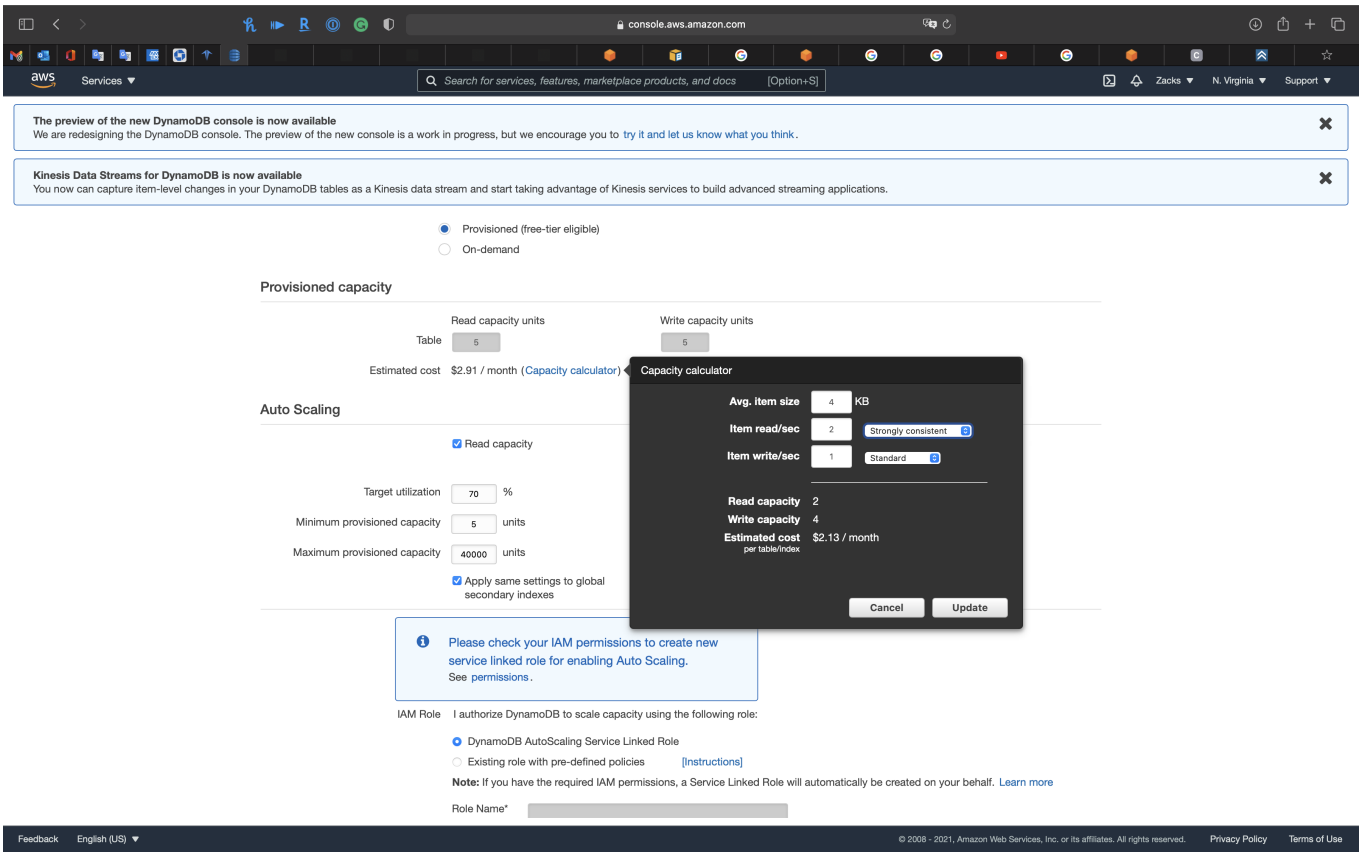
For modifying **Read capacity units** or **Write capacity units**, uncheck **Auto Scaling** first.

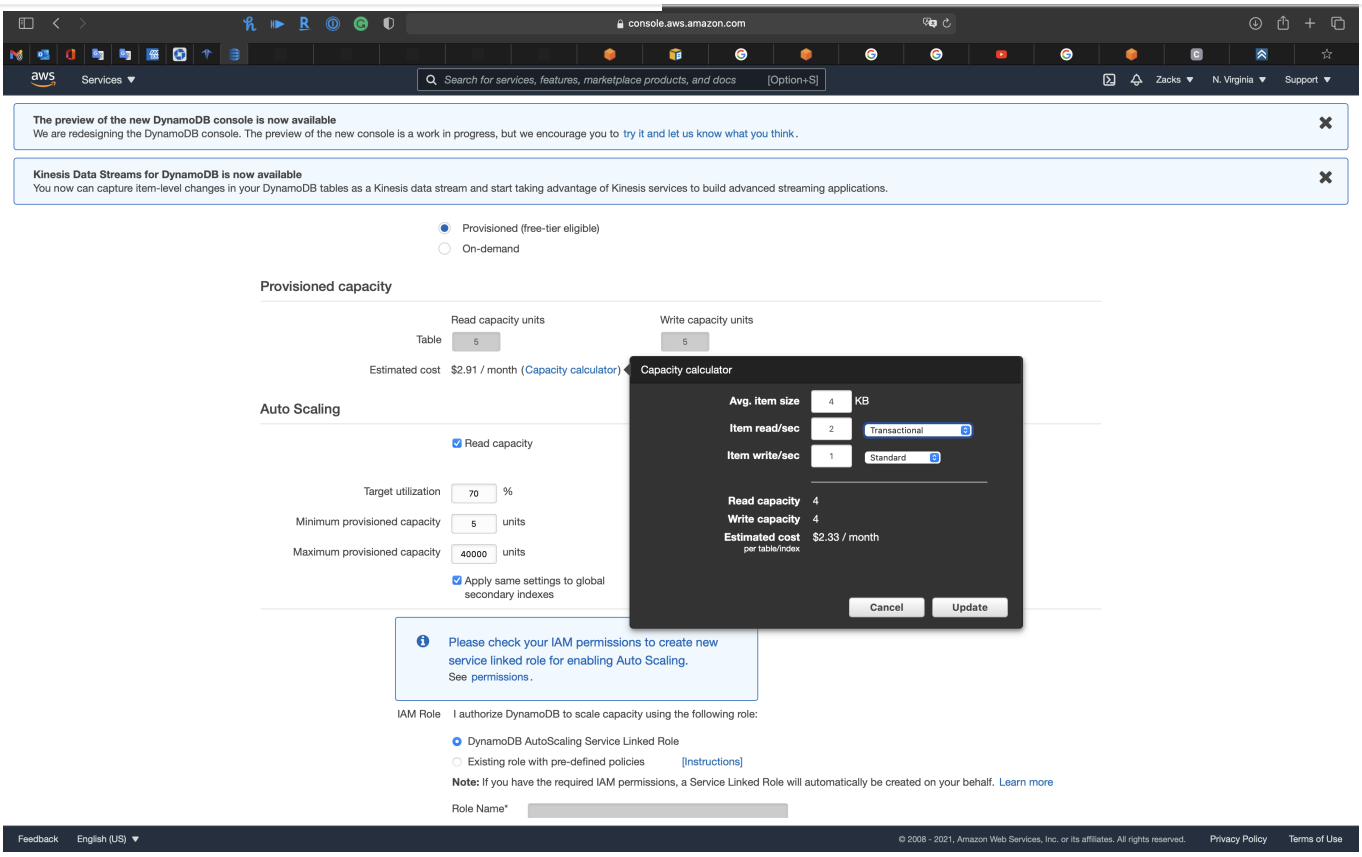
You can also click on `Capacity calculator`



See how the estimated cost change


console.aws.amazon.com
Services
Search for services, features, marketplace products, and docs [Option+S]
Zacks
N. Virginia
Support
The preview of the new DynamoDB console is now available
We are redesigning the DynamoDB console. The preview of the new console is a work in progress, but we encourage you to try it and let us know what you think.
Kinesis Data Streams for DynamoDB is now available
You now can capture item-level changes in your DynamoDB tables as a Kinesis data stream and start taking advantage of Kinesis services to build advanced streaming applications.
Provisioned (free-tier eligible)
On-demand
Provisioned capacity
Read capacity units
Write capacity units
Table
Estimated cost $2.91 / month (Capacity calculator)
Capacity calculator
Avg. item size 4 KB
Item read/sec 2 Eventually consistent
Item write/sec 1 Standard
Read capacity 1
Write capacity 4
Estimated cost per table/index $2.04 / month
Cancel
Update
Auto Scaling
Read capacity
Target utilization 70 %
Minimum provisioned capacity 5 units
Maximum provisioned capacity 40000 units
Apply same settings to global secondary indexes
Please check your IAM permissions to create new service linked role for enabling Auto Scaling.
See permissions.
IAM Role
I authorize DynamoDB to scale capacity using the following role:
DynamoDB AutoScaling Service Linked Role
Existing role with pre-defined policies [Instructions]
Note: If you have the required IAM permissions, a Service Linked Role will automatically be created on your behalf. Learn more
Role Name*
Feedback
English (US)
© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

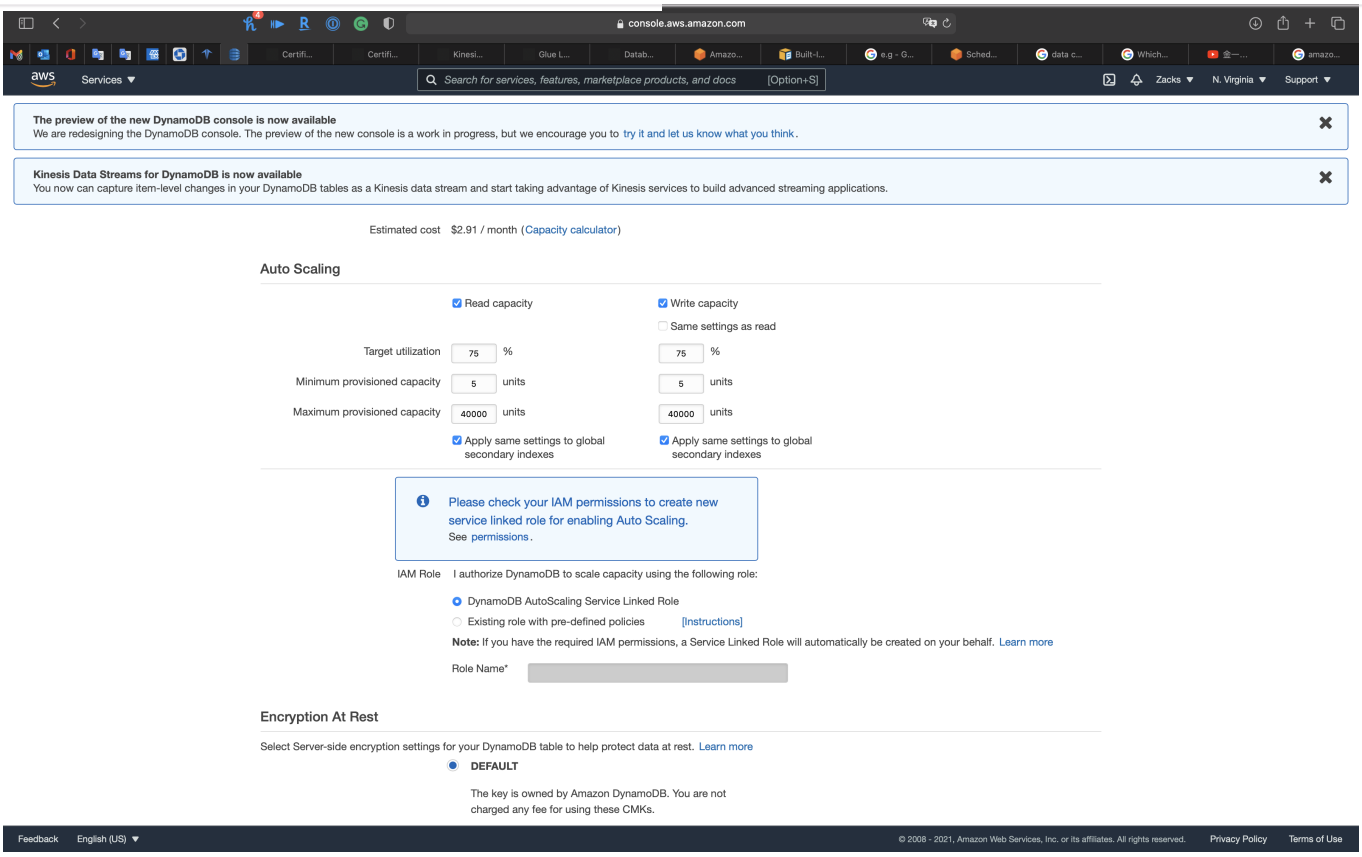

console.aws.amazon.com
Services
Search for services, features, marketplace products, and docs [Option+S]
Zacks
N. Virginia
Support
The preview of the new DynamoDB console is now available
We are redesigning the DynamoDB console. The preview of the new console is a work in progress, but we encourage you to try it and let us know what you think.
Kinesis Data Streams for DynamoDB is now available
You now can capture item-level changes in your DynamoDB tables as a Kinesis data stream and start taking advantage of Kinesis services to build advanced streaming applications.
Provisioned (free-tier eligible)
On-demand
Provisioned capacity
Read capacity units
Write capacity units
Table
Estimated cost $2.91 / month (Capacity calculator)
Capacity calculator
Avg. item size 4 KB
Item read/sec 2 Strongly consistent
Item write/sec 1 Standard
Read capacity 2
Write capacity 4
Estimated cost per table/index $2.13 / month
Cancel
Update
Auto Scaling
Read capacity
Target utilization 70 %
Minimum provisioned capacity 5 units
Maximum provisioned capacity 40000 units
Apply same settings to global secondary indexes
Please check your IAM permissions to create new service linked role for enabling Auto Scaling.
See permissions.
IAM Role
I authorize DynamoDB to scale capacity using the following role:
DynamoDB AutoScaling Service Linked Role
Existing role with pre-defined policies [Instructions]
Note: If you have the required IAM permissions, a Service Linked Role will automatically be created on your behalf. Learn more
Role Name*
Feedback
English (US)
© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use



- Provisioned capacity
  - Specify read capacity units (RCUs) and write capacity units (WCUs) per second for your application
  - For Avg. item size up to 4 KB, every 4 KB of Avg. item will be counted as a coefficient
    - 1 RCU = 1 read/sec strongly consistent
    - 1 RCU = 2 read/rec eventually consistent
    - 2 RCU = 1 read/rec transactional
  - For Avg. item size up to 1 KB, every 1 KB of Avg. item will be counted as a coefficient
    - 1 WCU = 1 write/sec
  - Can use auto scaling to automatically calibrate your table's capacity
    - Helps manage cost

Auto Scaling

Set as the screenshot.

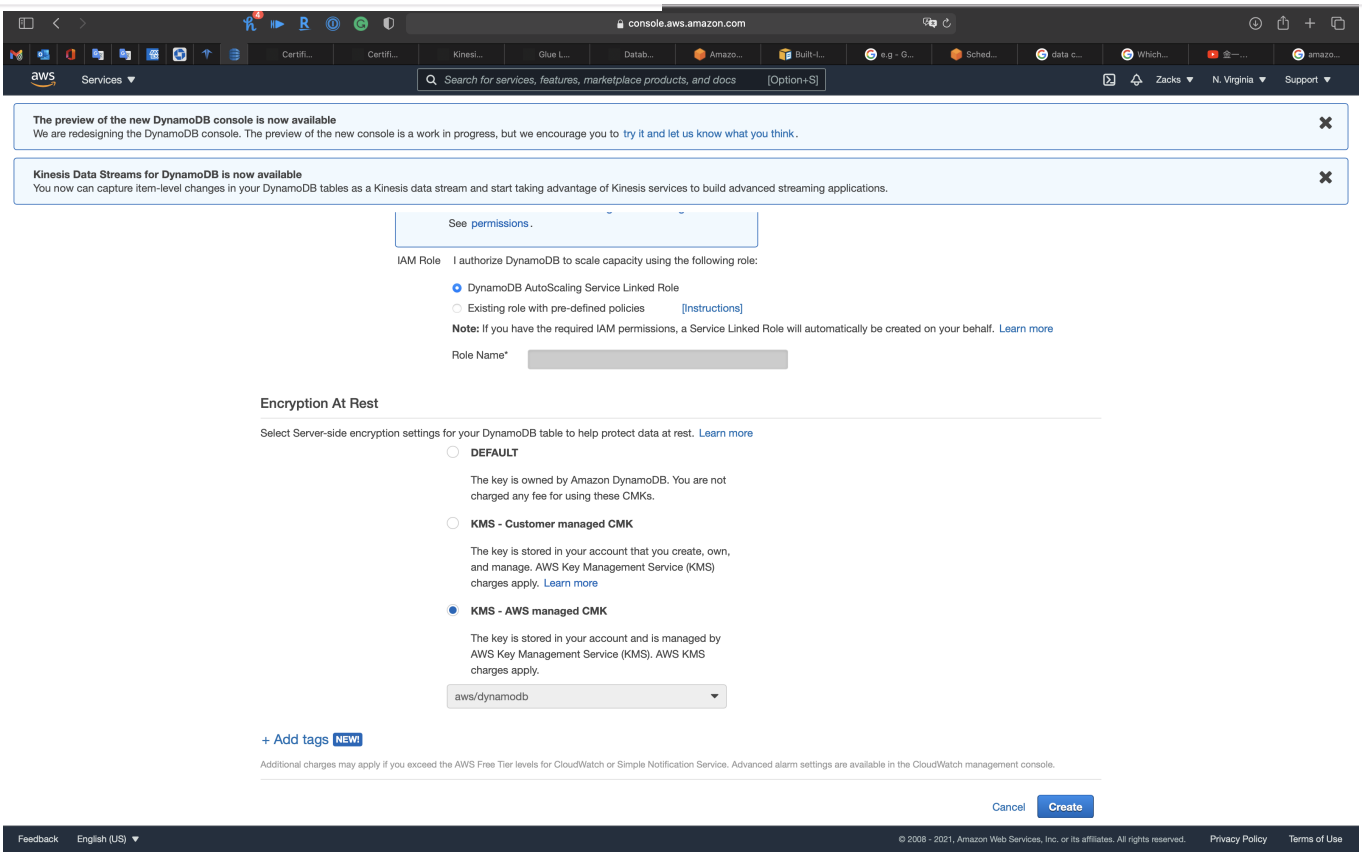


Encryption At Rest

Encryption service is free for DynamoDB.

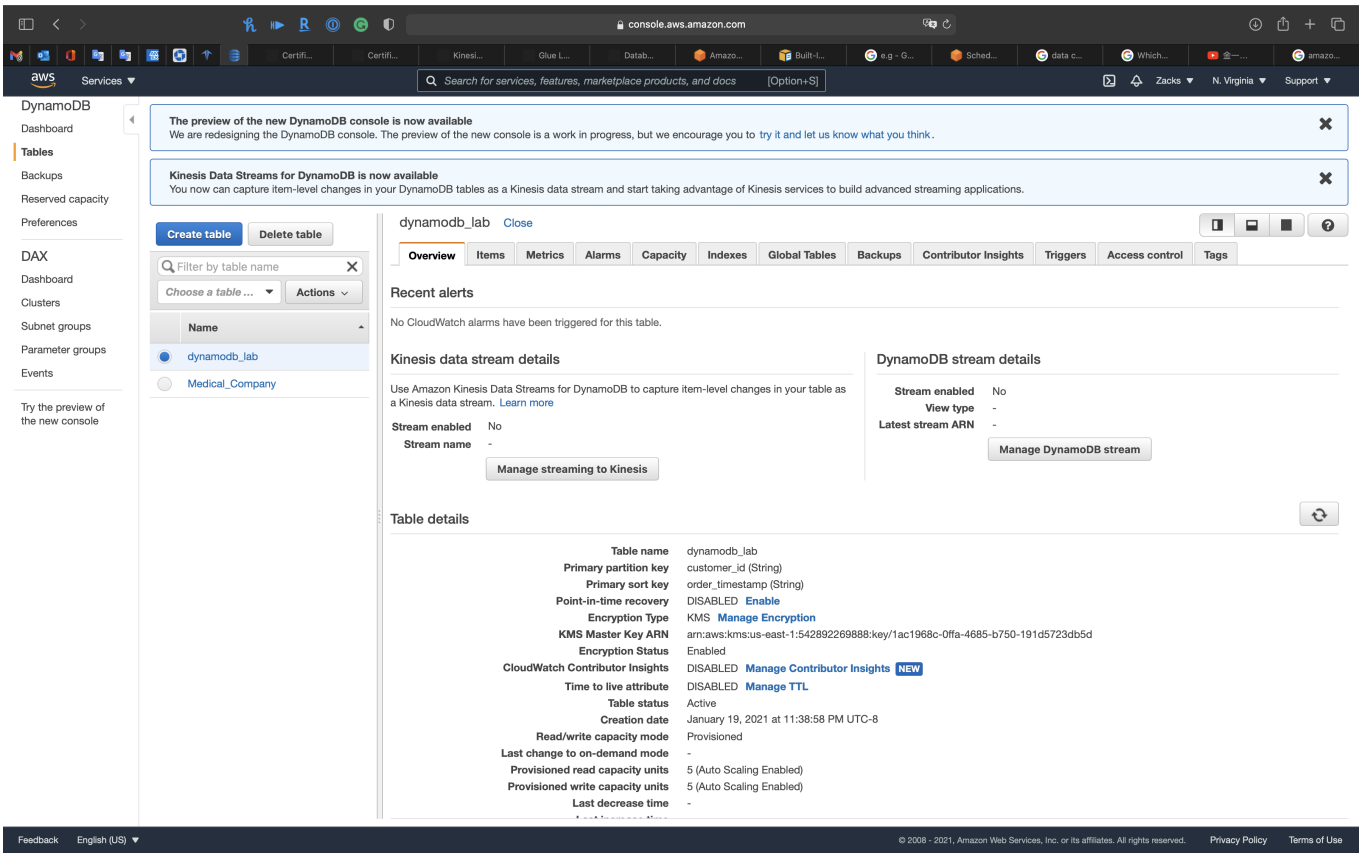
`KMS - AWS managed CMK` is a good choice. But for this lab let we select `Default`.

Click on `Create`



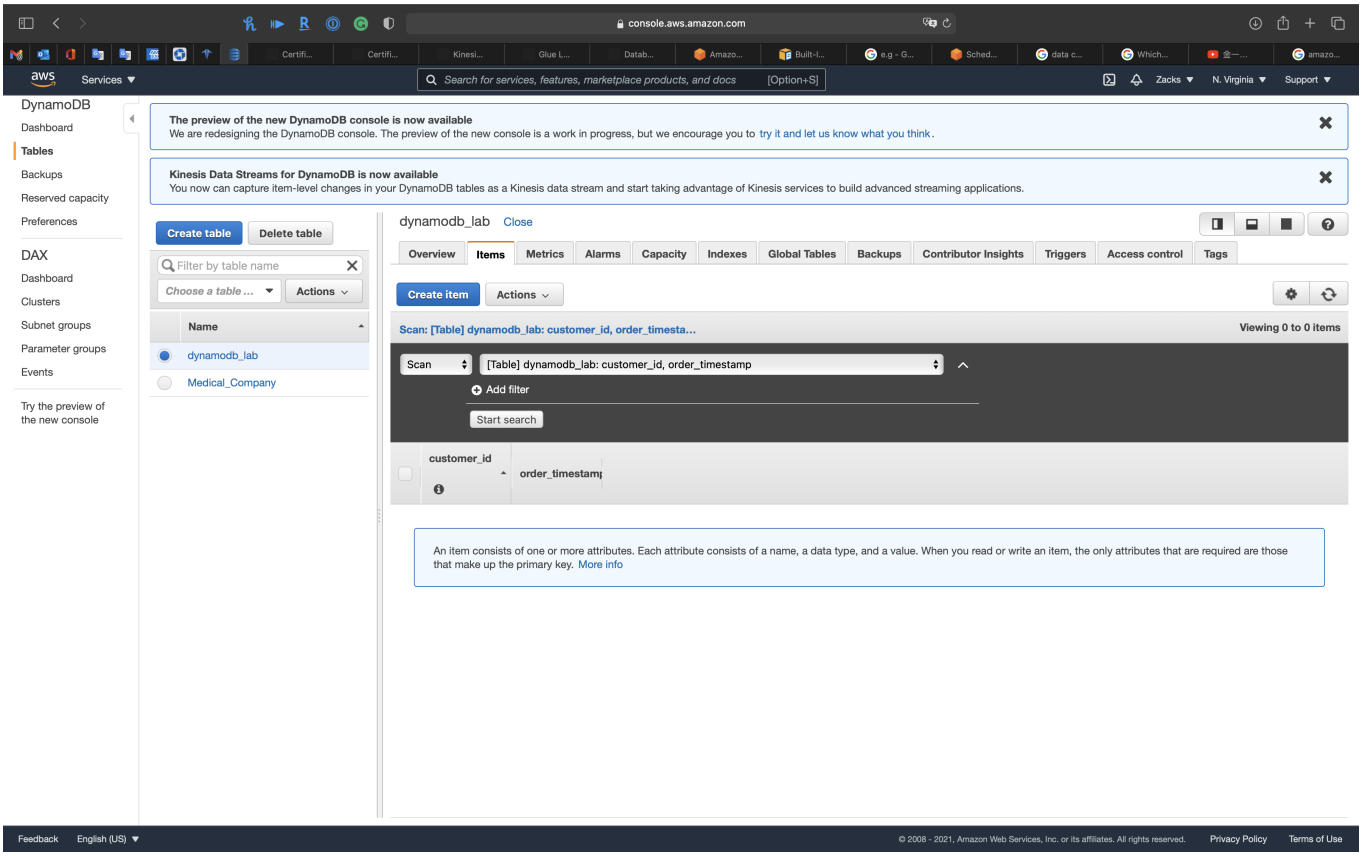
Global Tables Configuration

Click on `Items` tab.

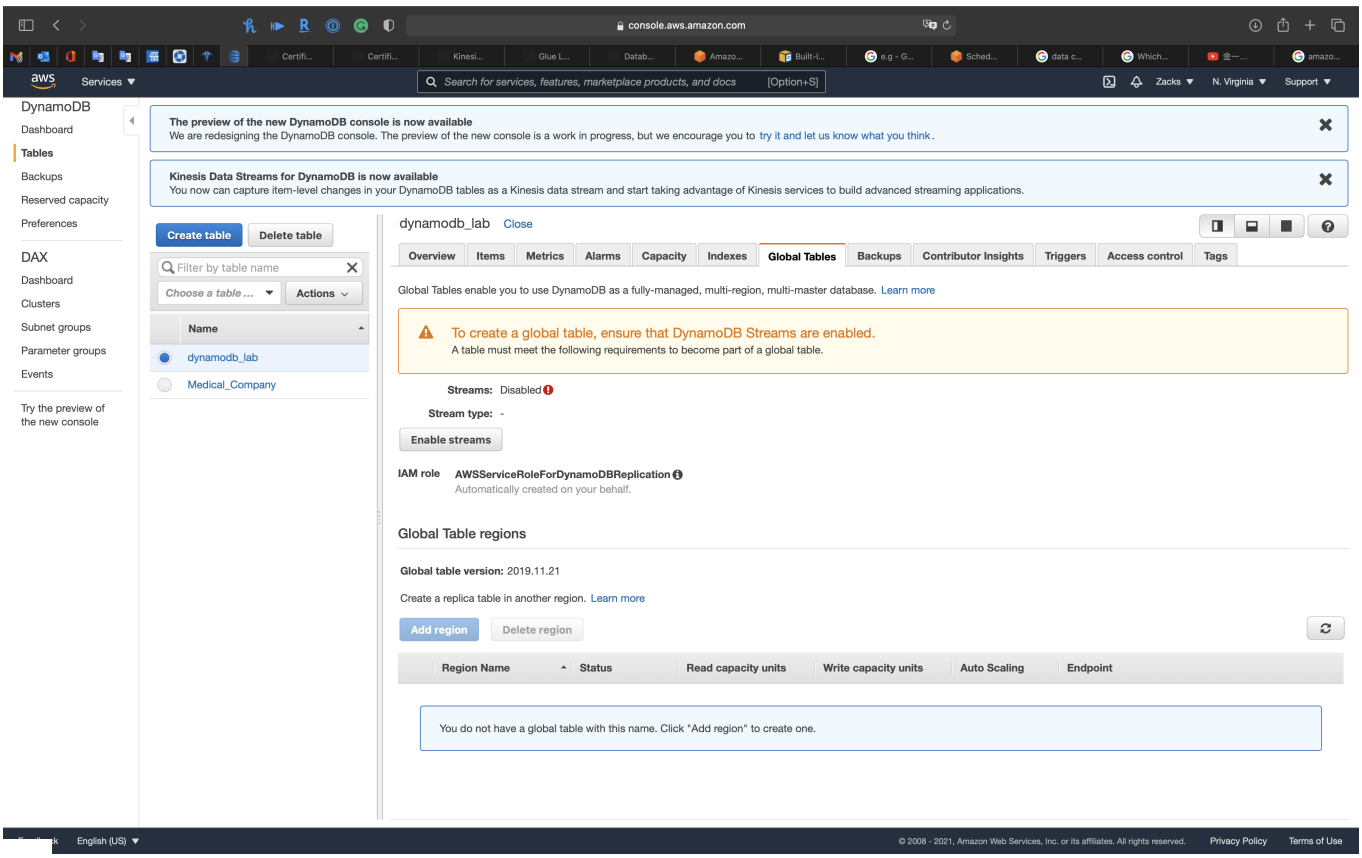


Now we don't have any item.

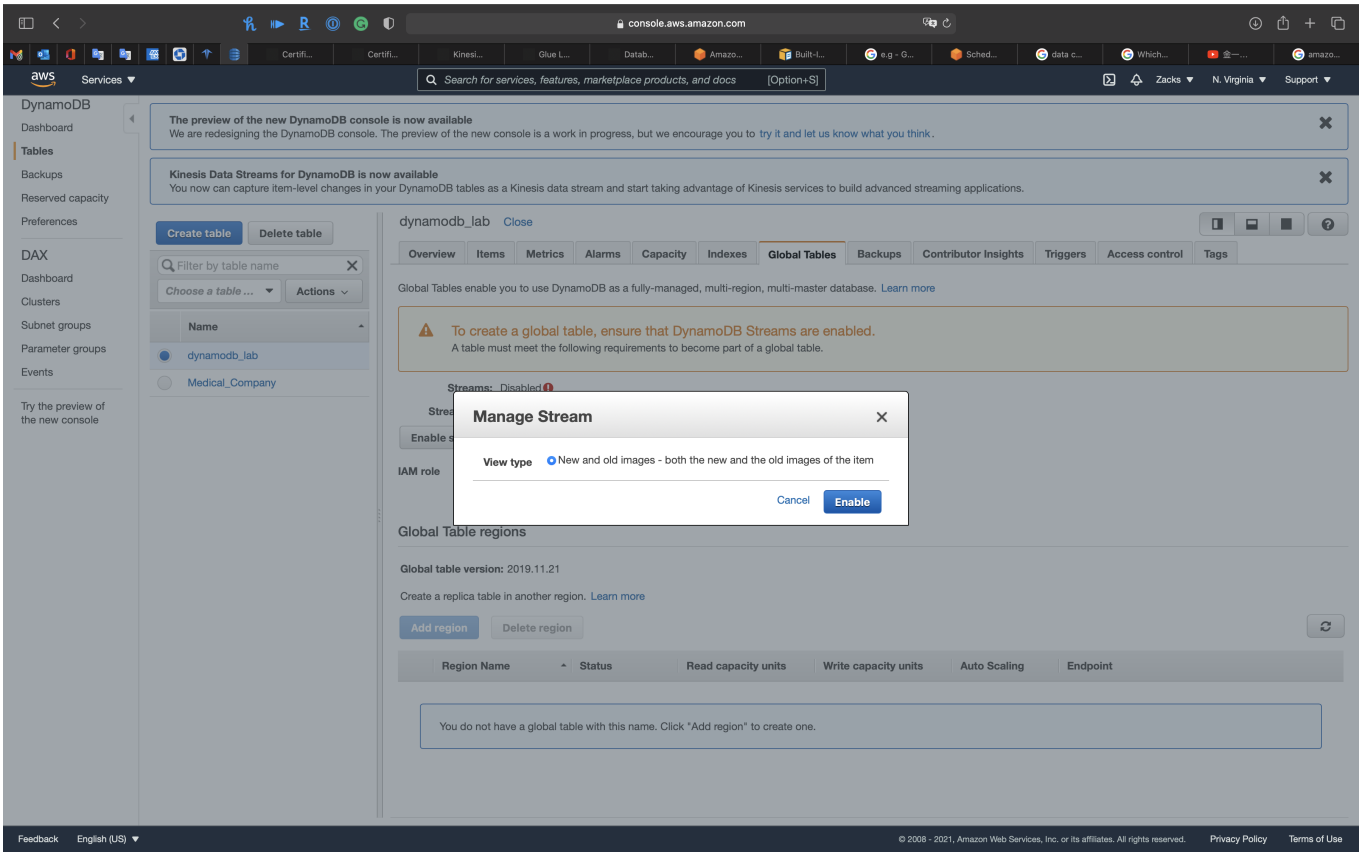
Click on Global Tables



Click on `Enable streams`



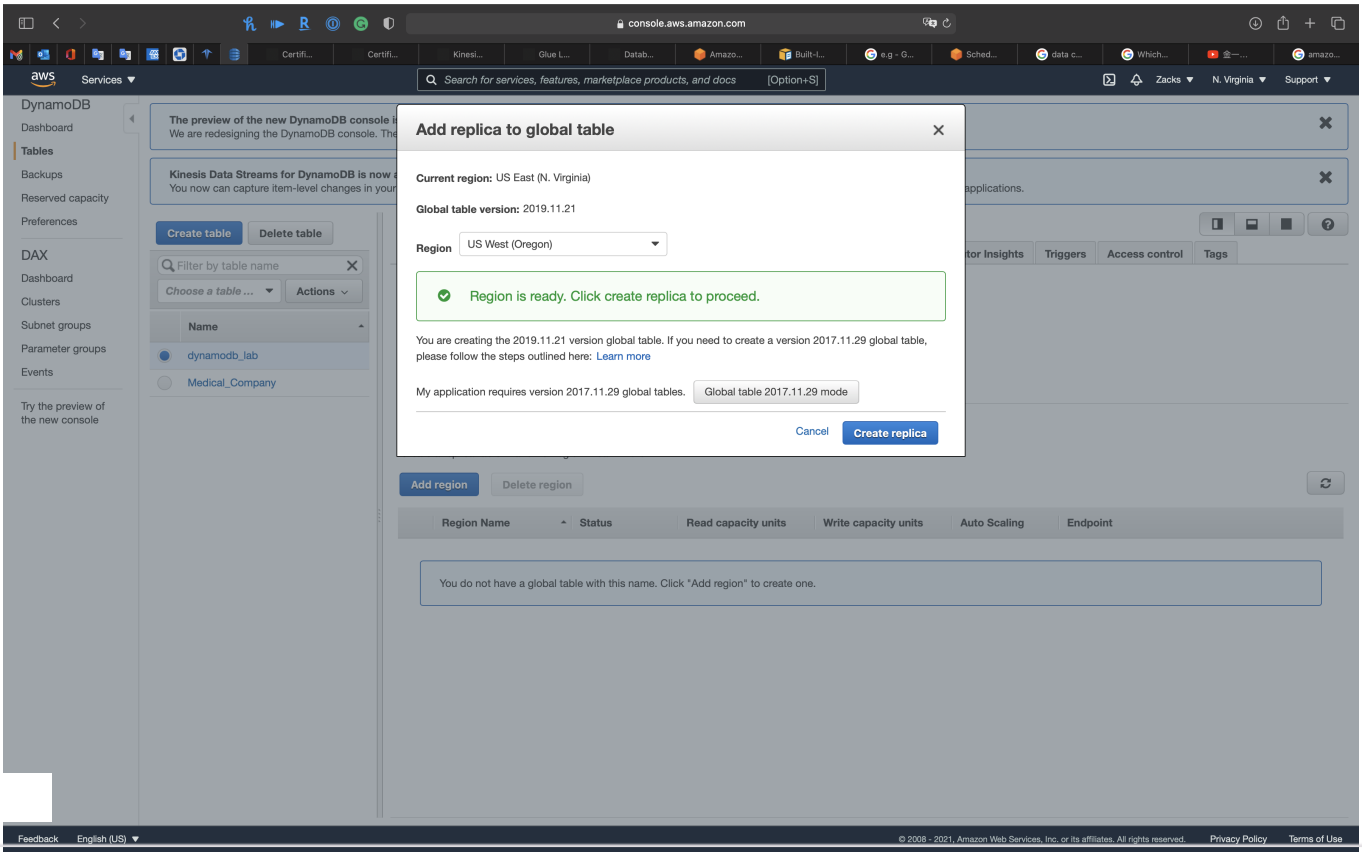
Click on `Enable`



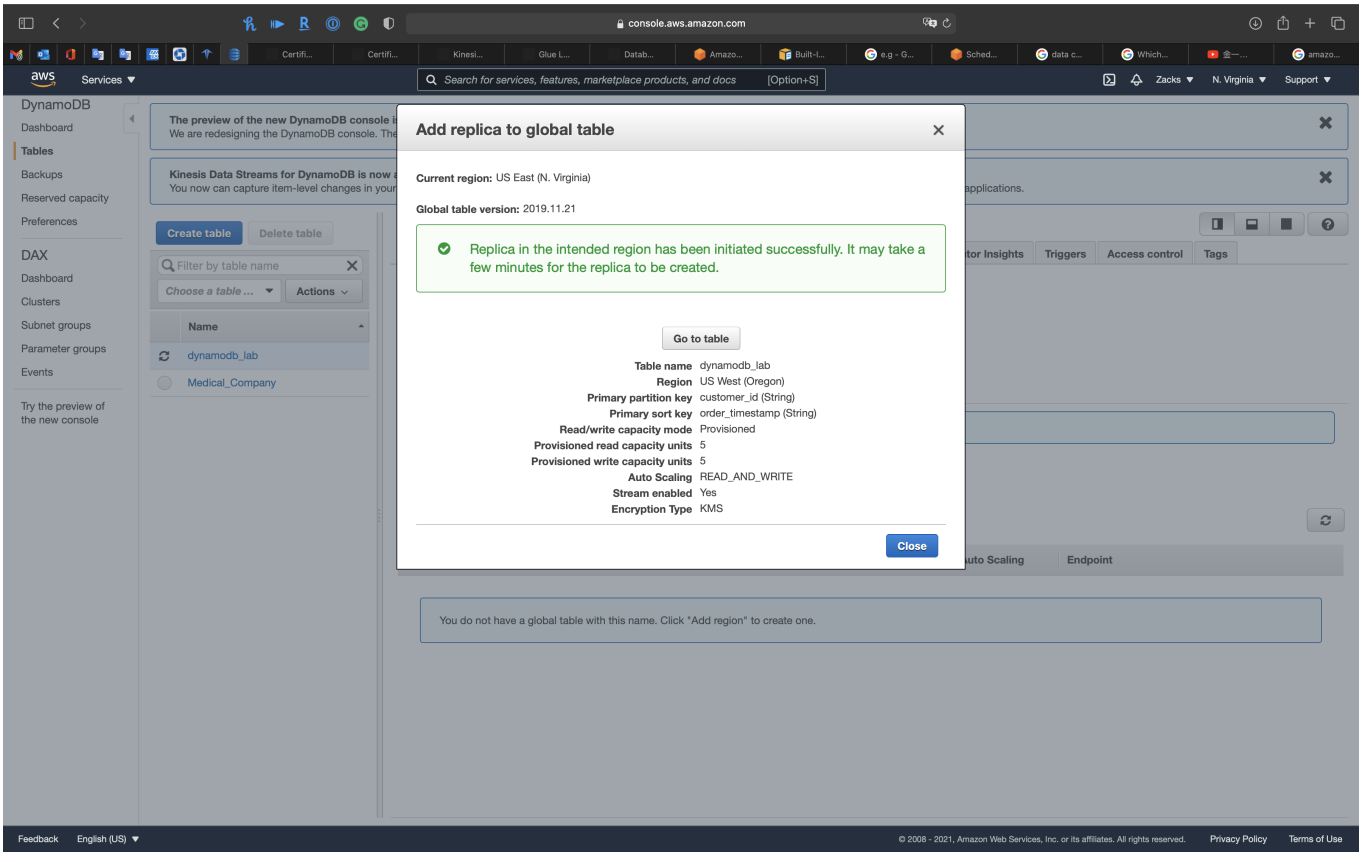
Click on `Add region`

Select a region that is different from your **Current region**

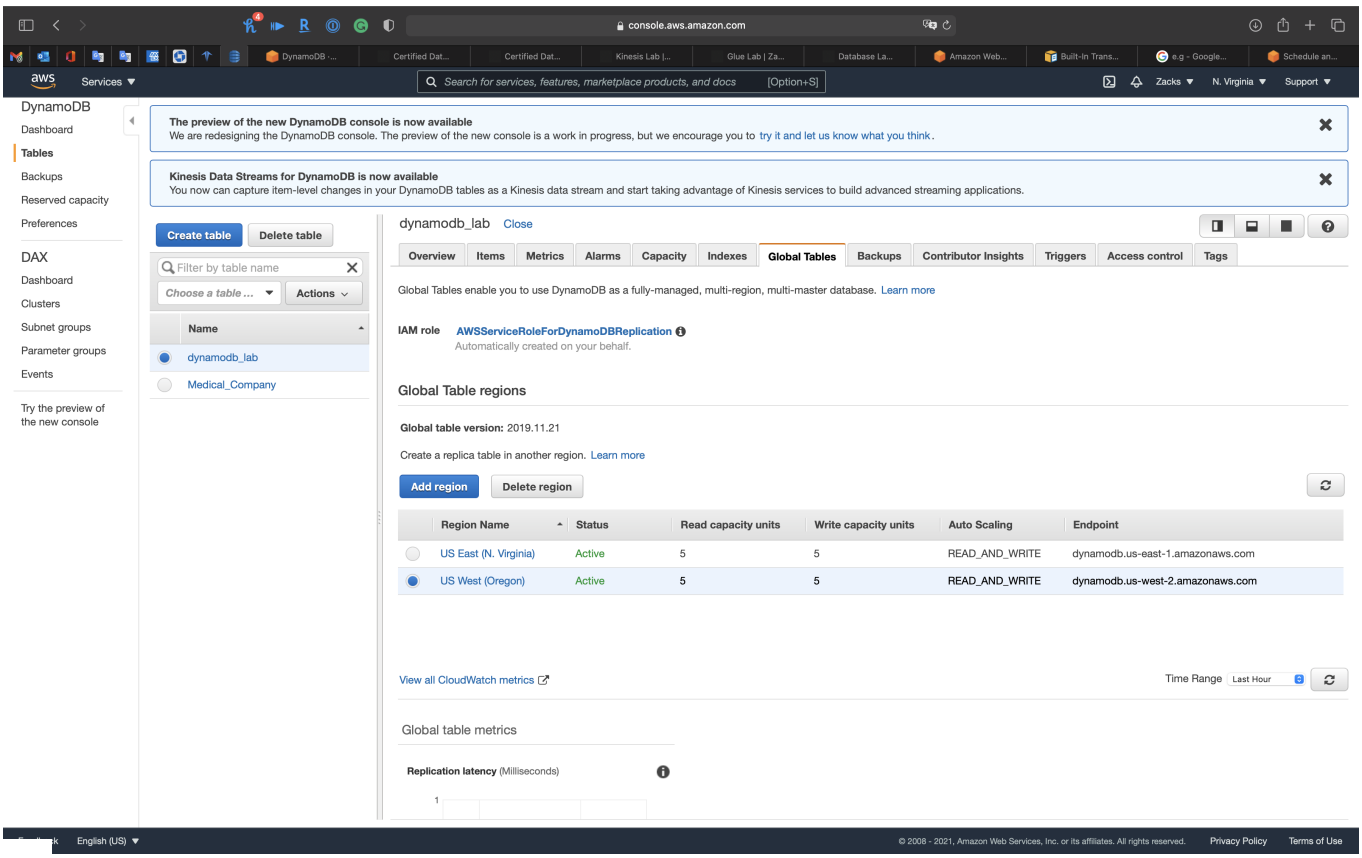
Click on `Create replica`

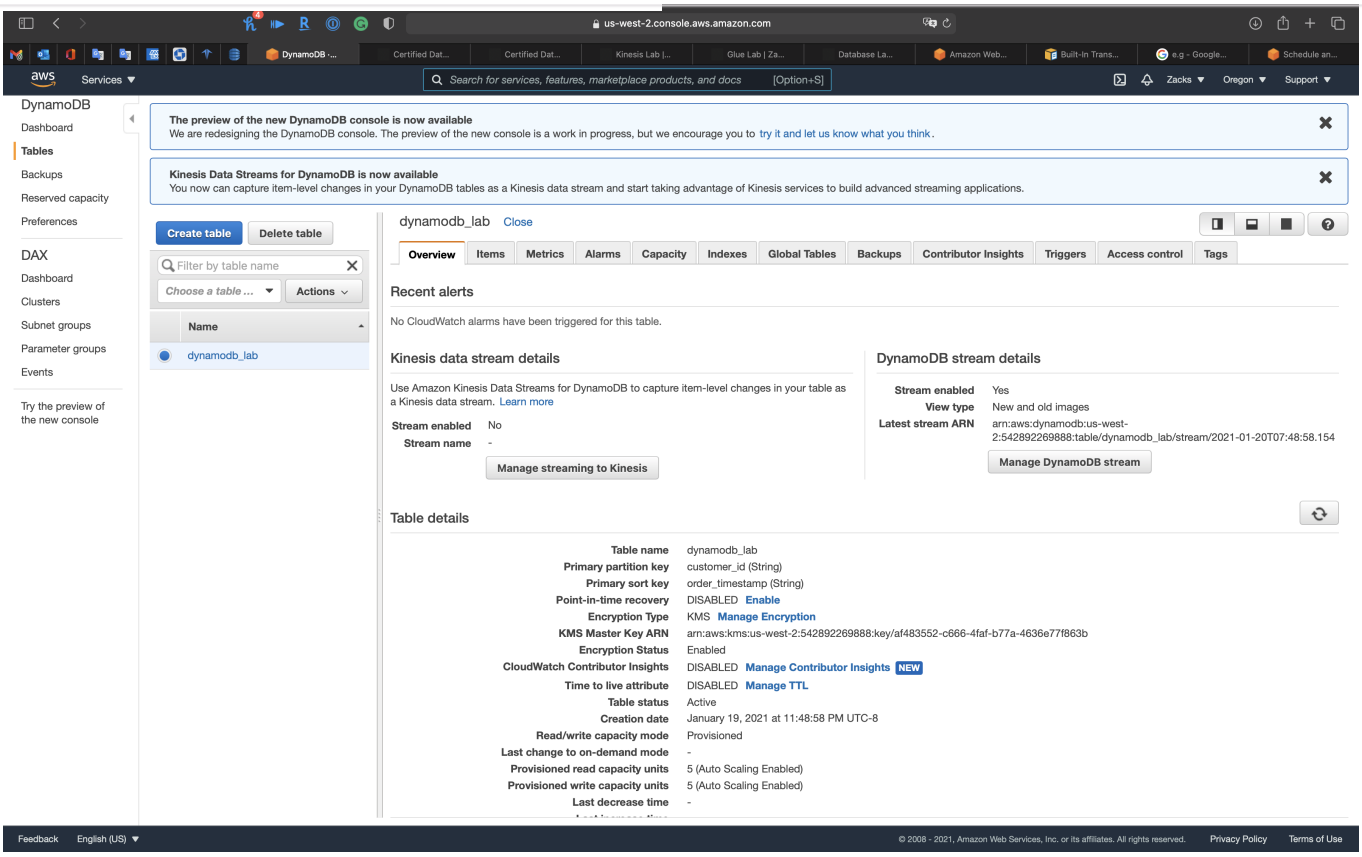


Review the settings and click on `Close`



We are now in `US EAST (N. Virginia)`. So we click on `US WEST (Oregon)` to see the replica.





Add items to the primary region by Python

Python3 DynamoDBLoader.py

```python
import boto3

# Get the DynamoDB service resource.
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')

# Instantiate a table resource object without actually
# creating a DynamoDB table in our code. Know that the attributes of
# the table are lazy-loaded: the attribute
# values are not populated until the attributes
# on the table resource are accessed or its load() method is called.
table = dynamodb.Table('dynamodb_lab')

# Print out some data about the table.
# This will cause a request to DynamoDB and the table attribute
# values will be set via the response.
print("\ntable creatin date-time {} \n\ntable key schema {}".format(
        table.creation_date_time, table.key_schema))

# Put an item into our table
table.put_item(
```

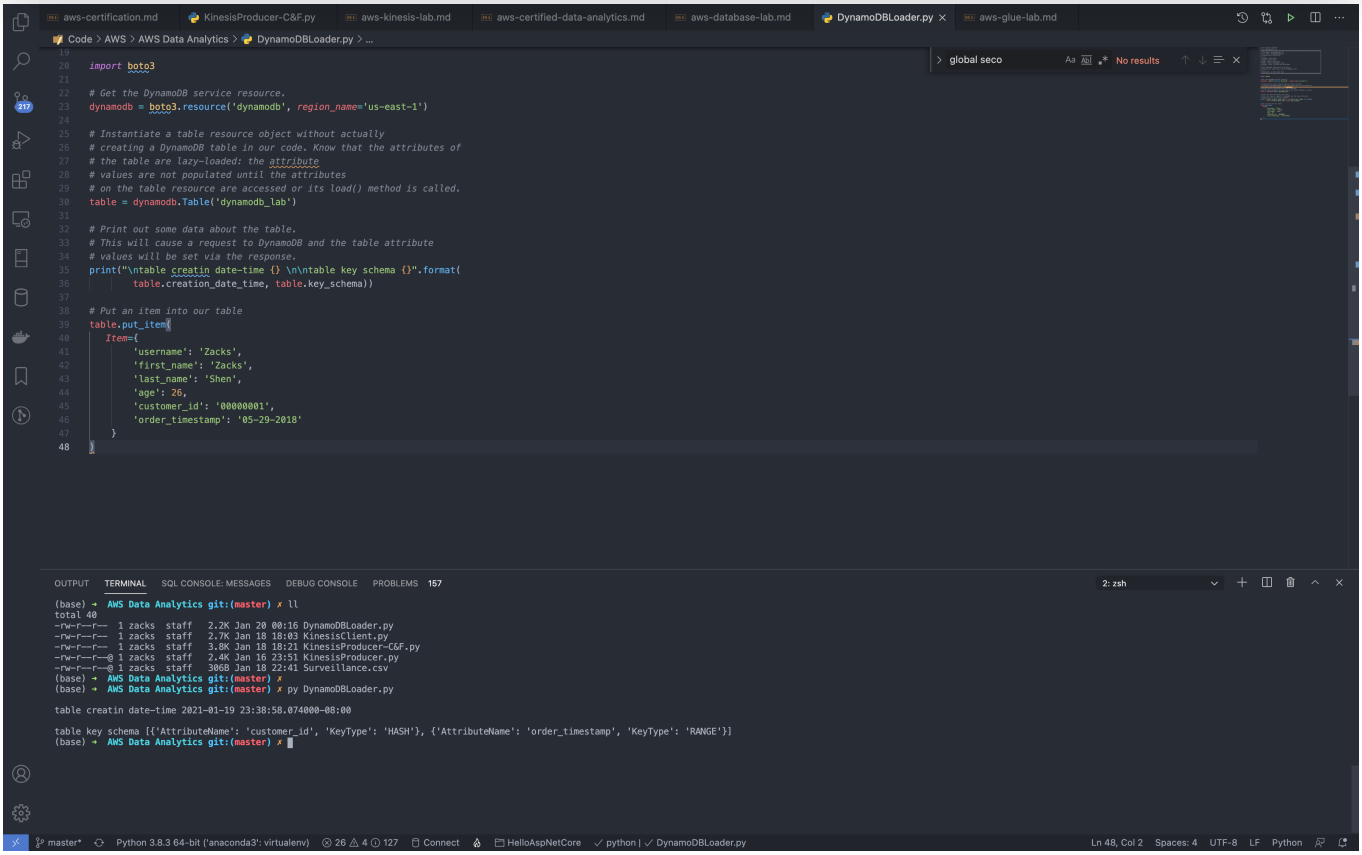
```
    Item={
        'username': 'Zacks',
        'first_name': 'Zacks',
        'last_name': 'Shen',
        'age': 26,
        'customer_id': '00000001',
        'order_timestamp': '05-29-2018'
    }
)
```

Save the code to your local machine then change directory to the program and run it.

If an permission error pops up, go check AWS CLI.

Python

```
python3 DynamoDBLoader.py
```



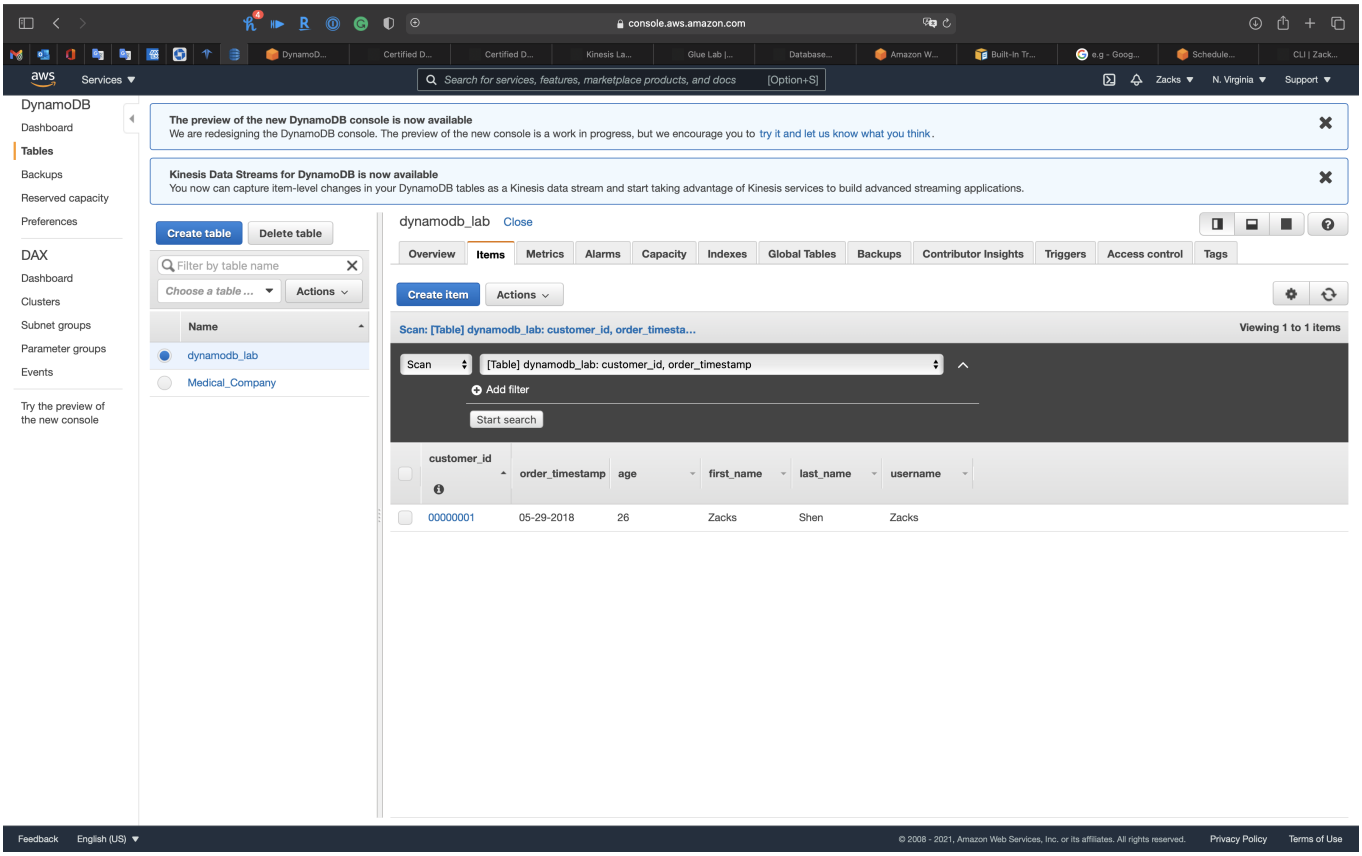
See the information from terminal: `table key schema [{'AttributeName': 'customer_id', 'KeyType': 'HASH'}, {'AttributeName': 'order_timestamp', 'KeyType': 'RANGE'}]`

Validation Test

Go to your regions to check the DynamoDB database.

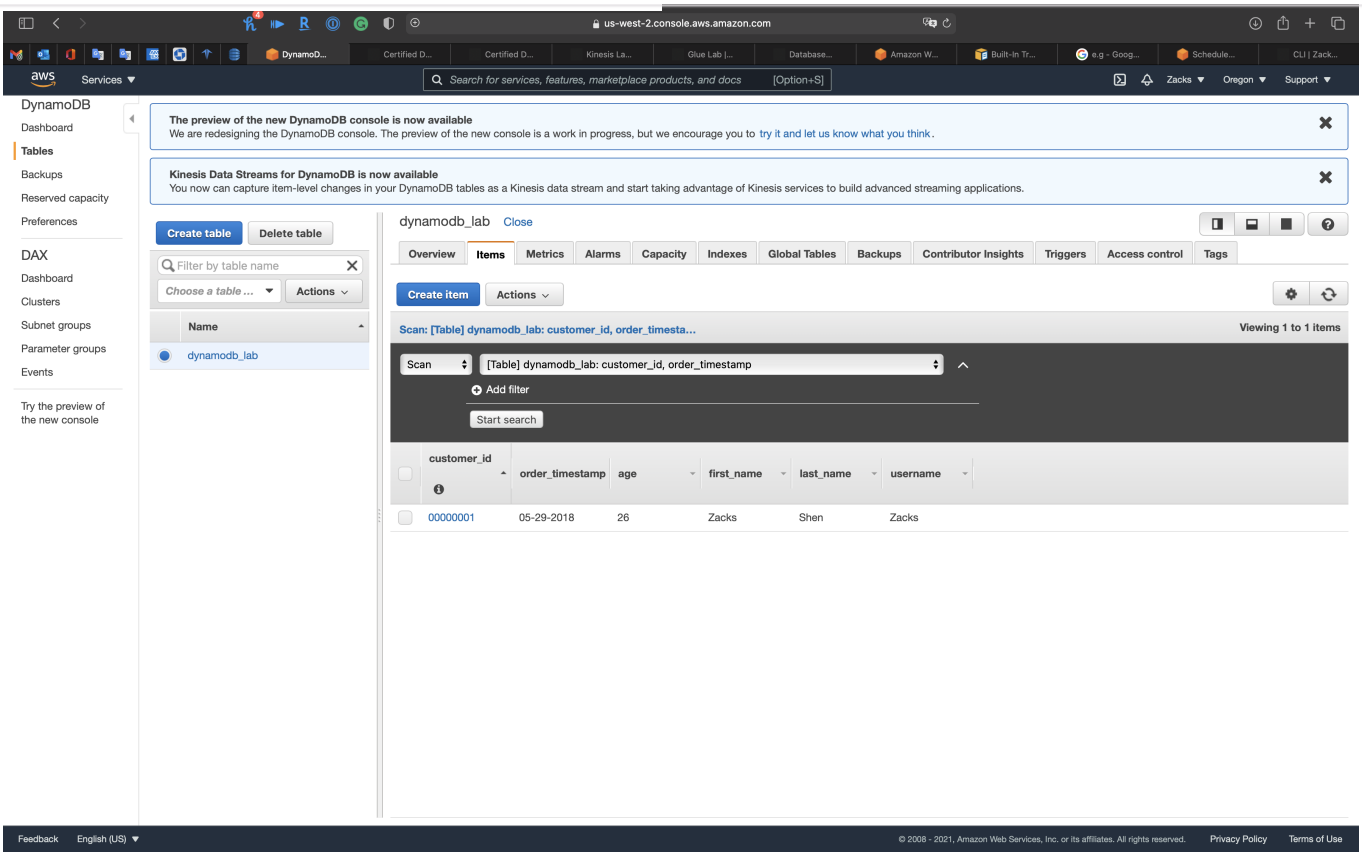
US EAST (N. Virginia)

Click on `Items` tab then click on refresh button.



US WEST (Oregon)

Click on `Items` tab then click on refresh button.



Add items to the replica region by Python

Modify the program, `region_name` and `Item`.

Python3 DynamoDBLoader.py

```
import boto3

# Get the DynamoDB service resource.
dynamodb = boto3.resource('dynamodb', region_name='us-west-2')

# Instantiate a table resource object without actually
# creating a DynamoDB table in our code. Know that the attributes of
# the table are lazy-loaded: the attribute
# values are not populated until the attributes
# on the table resource are accessed or its load() method is called.
table = dynamodb.Table('dynamodb_lab')

# Print out some data about the table.
# This will cause a request to DynamoDB and the table attribute
# values will be set via the response.
print("\ntable creatin date-time {} \n\ntable key schema {}".format(
        table.creation_date_time, table.key_schema))

```

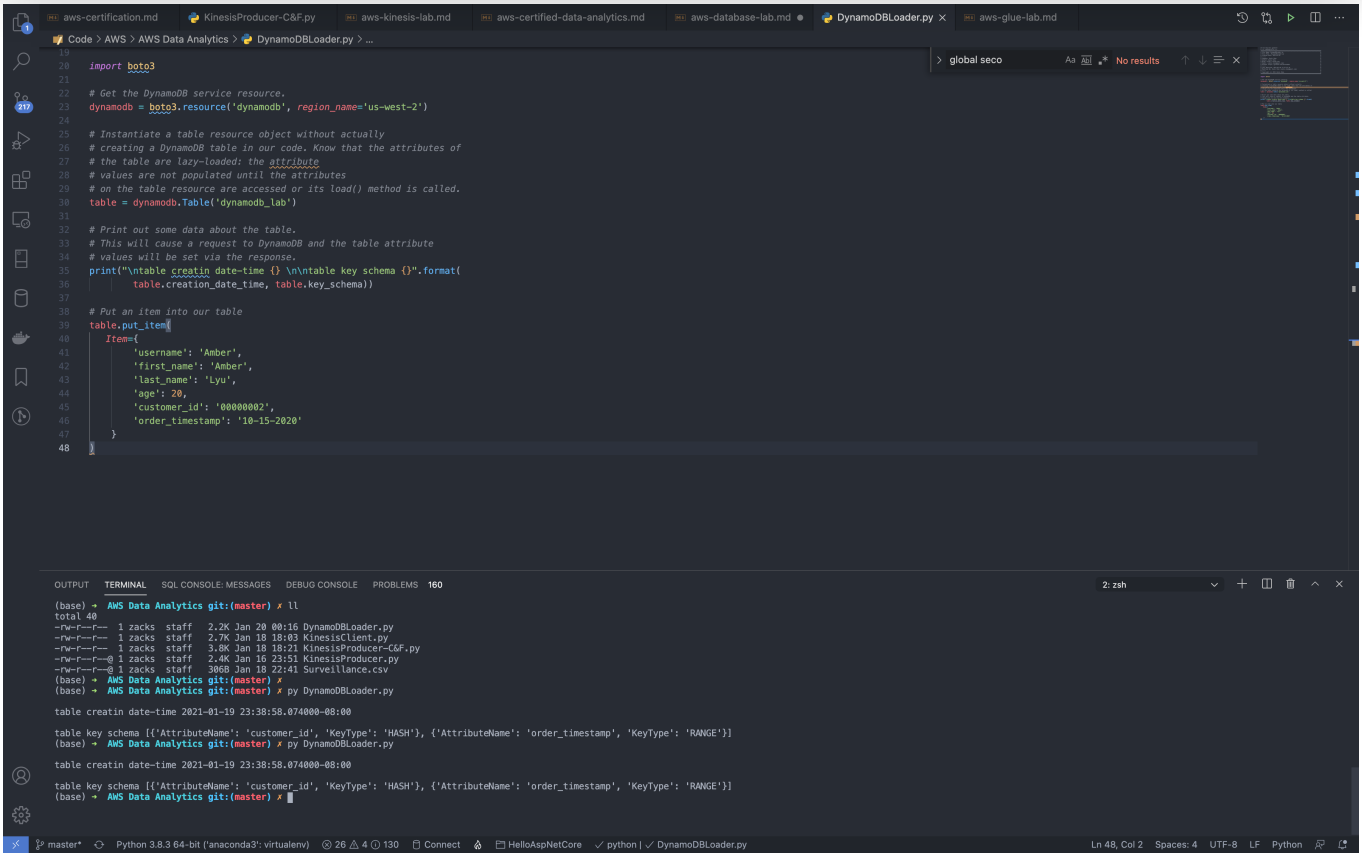
```
# Put an item into our table
table.put_item(
    Item={
         'username': 'Amber',
         'first_name': 'Amber',
         'last_name': 'Lyu',
         'age': 20,
         'customer_id': '00000002',
         'order_timestamp': '10-15-2020'
     }
)
```

Run the program.

If an permission error pops up, go check AWS CLI.

Python

```
python3 DynamoDBLoader.py
```



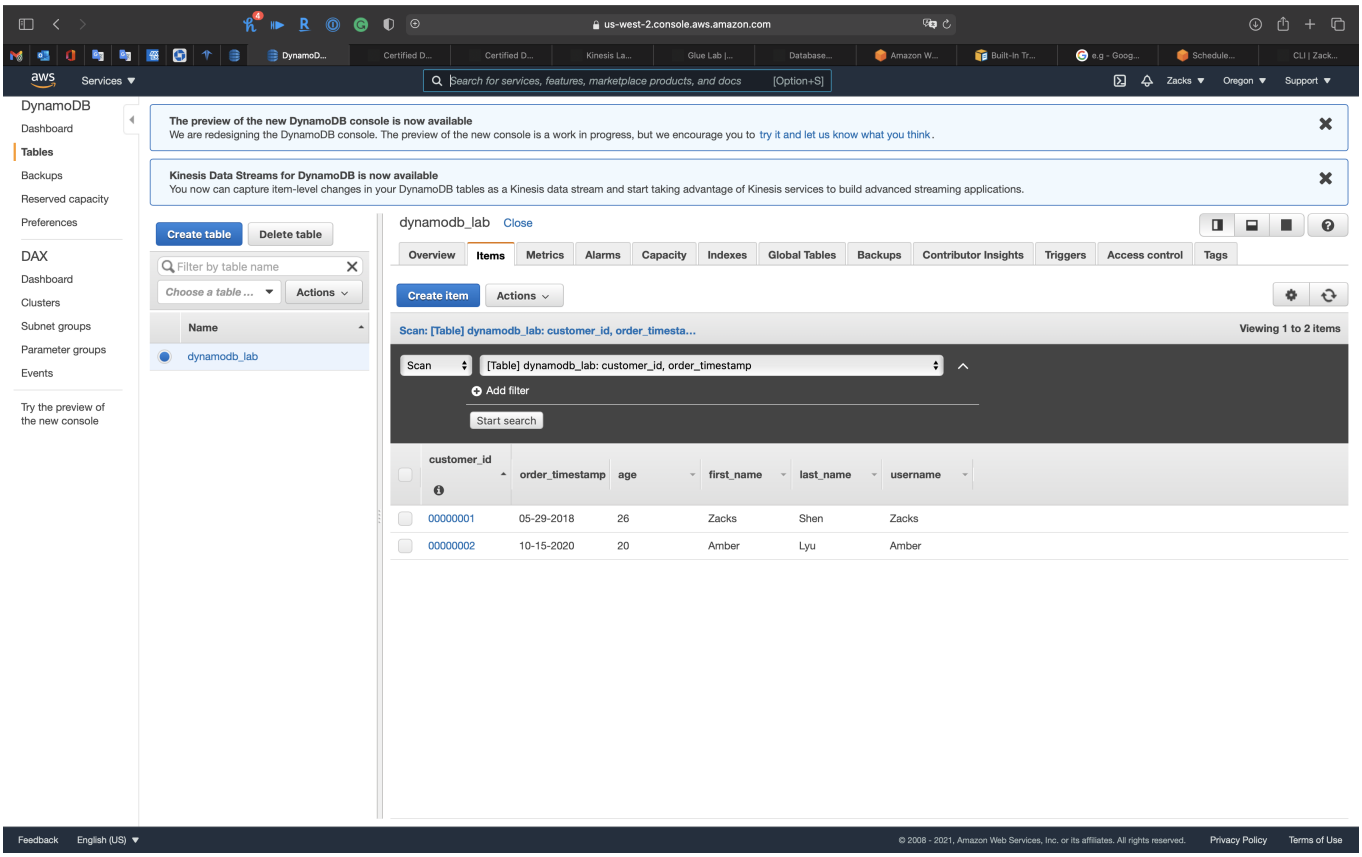
See the information from terminal: `table key schema [{'AttributeName': 'customer_id', 'KeyType': 'HASH'}, {'AttributeName': 'order_timestamp', 'KeyType': 'RANGE'}]`

Validation Test

Go to your regions to check the DynamoDB database.

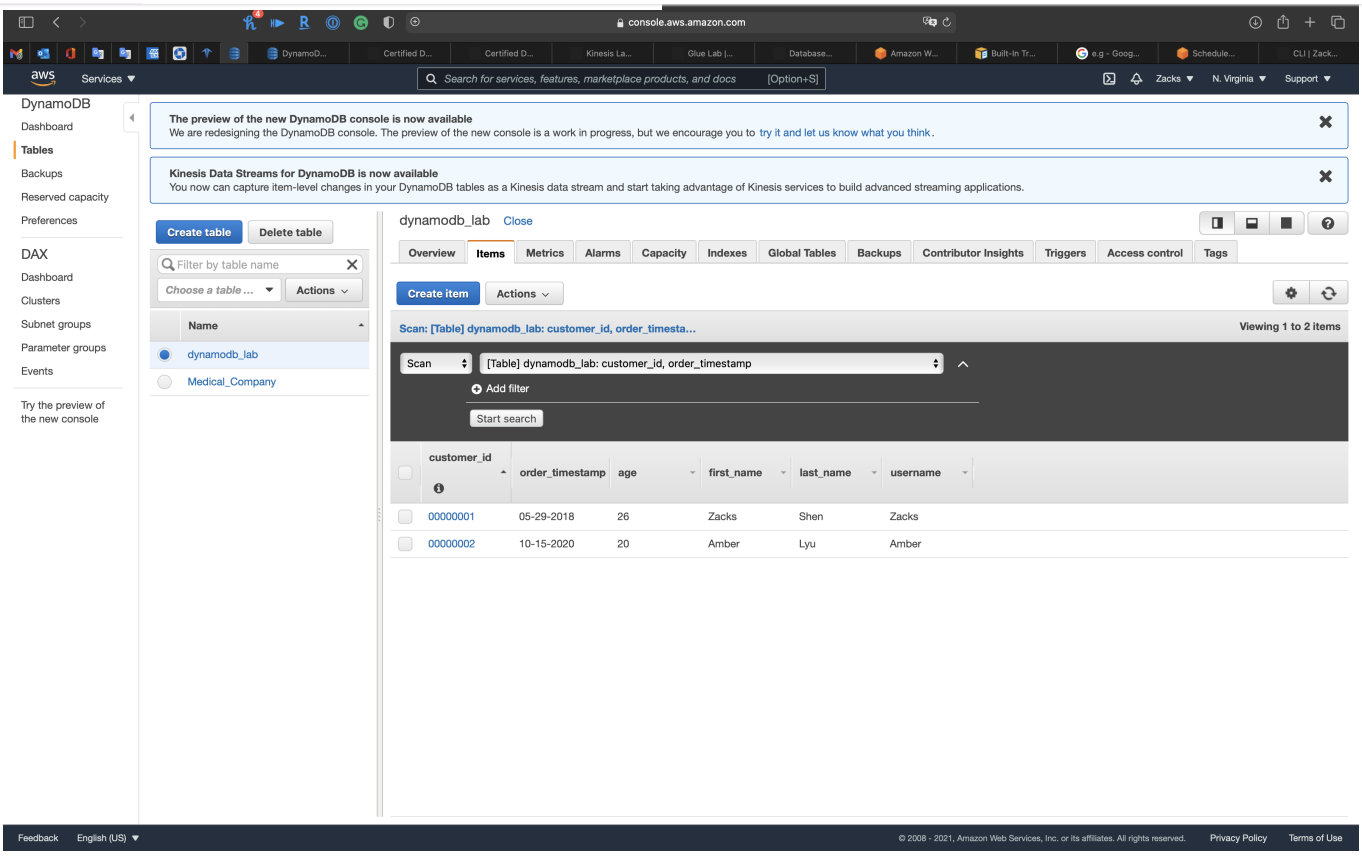
US WEST (Oregon)

Click on `Items` tab then click on refresh button.



US EAST (N. Virginia)

Click on `Items` tab then click on refresh button.



Tech AWS Labs AWS DAS DB Analytics NoSQL

‹ AWS Redshift Lab

AWS Elasticsearch Service ›

3.8m | 57:57

Powered by Hexo & NexT.Mist