



CS 470 Module Five Assignment Two Guide

In the previous assignment, you created and worked with your database tables all within the console. In this assignment, you will leverage the Lambda skills you developed in Module Four and write some code to work with the database.

NOTE: The source code referred to below is in the Module Five Source Code ZIP file in the following directories:

Name	Size	Packed Size
DeleteRecord	2 051	829
FindOneQuestion	4 236	1 557
GetSingleRecord	2 033	832
TableScan	3 747	1 387
UpsertAnswer	153 600	85 237
UpsertQuestion	153 732	84 949

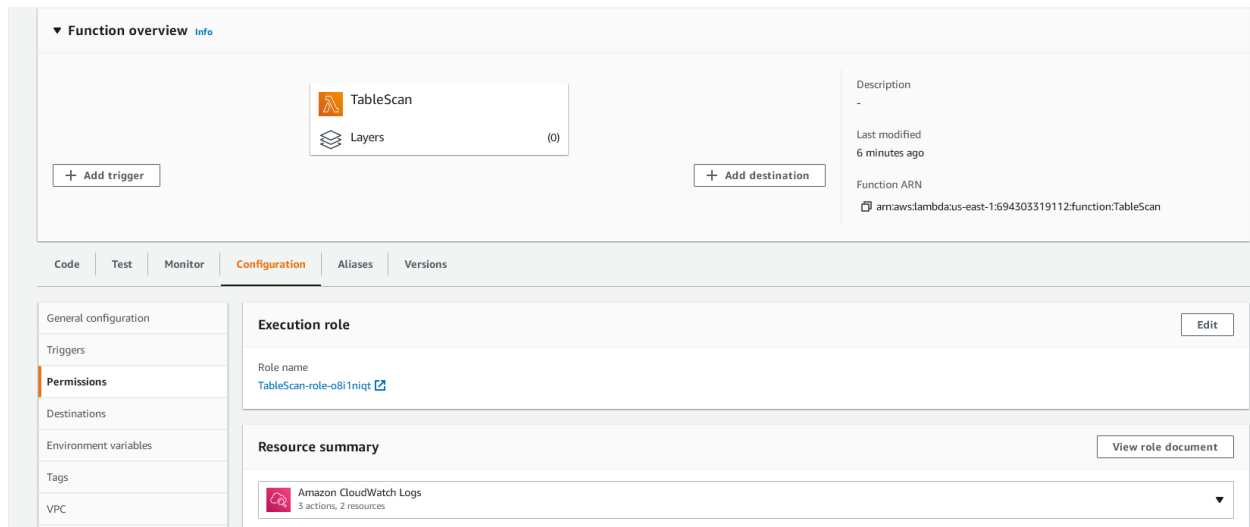
Part One – The Query Lambda

You are going to build a single Lambda that can query either the Question or Answer table, including the filter logic.

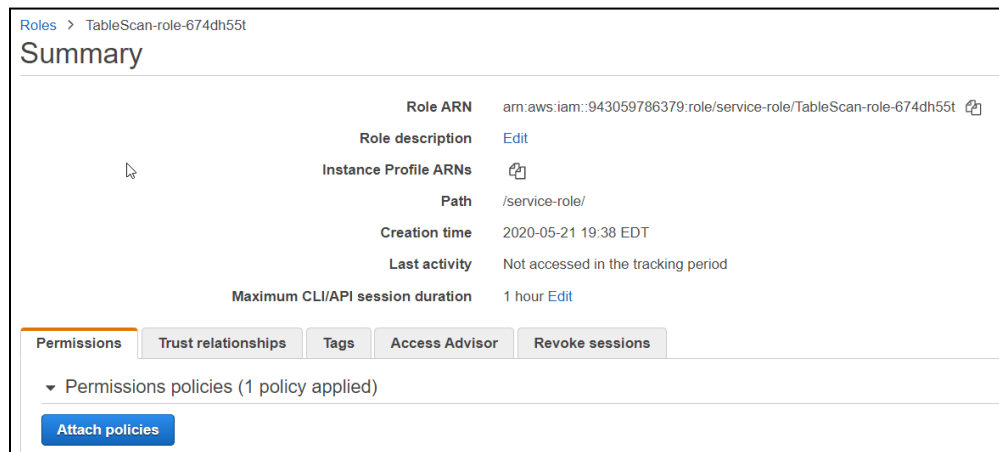
1. Create a Lambda named “TableScan”. See Module Four for how to create the Lambda.
2. Copy the source code from the index.js file in the TableScan folder and replace the default Lambda code.
3. Make sure to click **Deploy**.
4. Create a Lambda test event (review Module Four if needed) called “QuestionWithoutFilter” with the following JSON:

```
{
  "resource": "/Questions",
  "httpMethod": "GET",
  "queryStringParameters": {"include":{"relation":"answers"}},
  "multiValueQueryStringParameters": {"include":{"relation":"answers"}}
}
```

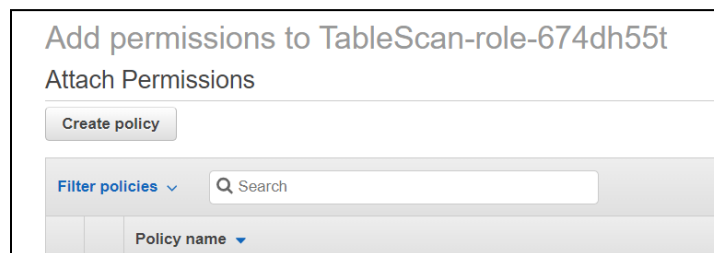
5. Before you can test, you will need to create a new security policy. Click the **Configuration** tab, then select **Permissions** for your TableScan Lambda.



6. Click the **Role name** starting with “TableScan-role...”.
7. The AWS Console will launch the IAM Summary page.



8. Click the **Attach Policies** button.
9. Click the **Create Policy** button.



10. The **Create Policy** screen will now launch.

Create policy

A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor and using JSON. [Learn more](#)

Visual editor **JSON** [Import managed policy](#)

[Expand all](#) | [Collapse all](#)

▼ Select a service [Clone](#) | [Remove](#)

► **Service** [Choose a service](#)

Actions Choose a service before defining actions

Resources Choose actions before applying resources

Request conditions Choose actions before specifying conditions

[Add additional permissions](#)

11. Click **Choose a service**.

12. Type “DynamoDB” into the search bar and select **DynamoDB**.

▼ **Service** **Select a service below**

[close](#)

Q dynamd I

DynamoDB [?](#) [DynamoDBAccelerator](#) [?](#)

13. The **Actions** section will now expand.

- Expand the **Read** section.
- Select “GetItem”, “Query”, and “Scan”.

▼ **Actions** **Specify the actions allowed in DynamoDB** [?](#) [Switch to deny permissions](#) [?](#)

[close](#)

Q Filter actions

Manual actions [\(add actions\)](#)

☐ All DynamoDB actions (dynamodb:*)

Access level

► ☐ List

▼ ☐ Read **(3 selected)**

☐ BatchGetItem [?](#) ☐ DescribeLimits [?](#) ☒ GetItem [?](#)

☐ ConditionCheckItem [?](#) ☐ DescribeReservedCapacity [?](#) ☐ GetRecords [?](#)

☐ DescribeBackup [?](#) ☐ DescribeReservedCapacityOfferings [?](#) ☐ GetShardIterator [?](#)

☐ DescribeContinuousBackups [?](#) ☐ DescribeStream [?](#) ☐ ListStreams [?](#)

☐ DescribeContributorInsights [?](#) ☐ DescribeTable [?](#) ☐ ListTagsOfResource [?](#)

☐ DescribeGlobalTable [?](#) ☐ DescribeTableReplicaAutoScaling [?](#) ☒ Query [?](#)

☐ DescribeGlobalTableSettings [?](#) ☐ DescribeTimeToLive [?](#) ☒ Scan [?](#)

[Expand all](#) | [Collapse all](#)

- c. Open the **Write** section.
- d. Select “DeleteItem”, “PutItem”, and “UpdateItem”.

▼

Write (3 selected)

☐ BatchWriteItem ?

☐ CreateBackup ?

☐ CreateGlobalTable ?

☐ CreateTable ?

☐ CreateTableReplica ?

☐ DeleteBackup ?

☒ DeleteItem ?

☐ DeleteTable ?

☐ DeleteTableReplica ?

☐ PurchaseReservedCapacityOfferin... ?

☒ PutItem ?

☐ RestoreTableFromBackup ?

☐ RestoreTableToPointInTime ?

☐ UpdateContinuousBackups ?

☐ UpdateContributorInsights ?

☐ UpdateGlobalTable ?

☐ UpdateGlobalTableSettings ?

☒ UpdateItem ?

☐ UpdateTable ?

☐ UpdateTableReplicaAutoScaling ?

☐ UpdateTimeToLive ?

- e. Click the orange link under **Resources** to “Specify **table** resource ARN for the **Query** and 5 more actions”.

► Resources

Specify **table** resource ARN for the **Query** and 5 more actions. ⓘ

- f. Click the link to **Add ARN** under **table**.

▼ Resources

● Specific

○ All resources

close

index ?

You have not specified resource with type **index**

Add ARN to restrict access

Any

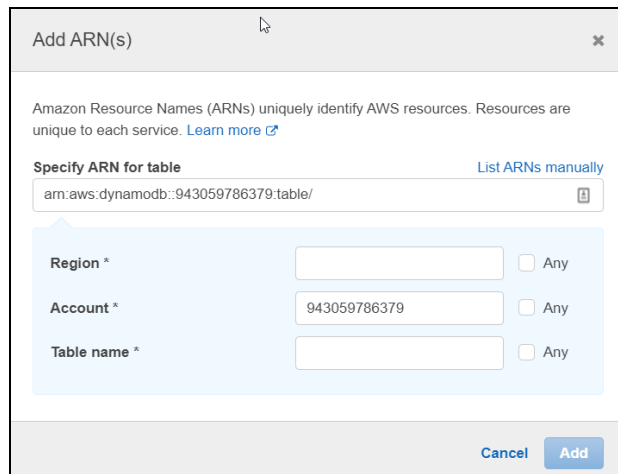
table ?

Specify **table** resource ARN for the **Query** and 5 more actions. ⓘ

Add ARN to restrict access

Any

- g. The console will pop up the **Add ARN(s)** dialog.



Add ARN(s)

Amazon Resource Names (ARNs) uniquely identify AWS resources. Resources are unique to each service. [Learn more](#)

Specify ARN for table [List ARNs manually](#)

arn:aws:dynamodb::943059786379:table/

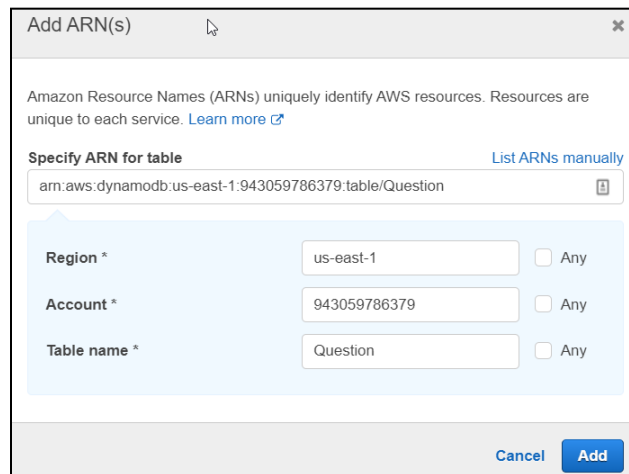
Region * ☐ Any

Account * ☐ Any

Table name * ☐ Any

[Cancel](#) [Add](#)

- h. Set **Region** to “us-east-1”.
- i. Do not change the **Account**.
- j. Set **Table name** to “Question”.
- k. Click **Add**.



Add ARN(s)

Amazon Resource Names (ARNs) uniquely identify AWS resources. Resources are unique to each service. [Learn more](#)

Specify ARN for table [List ARNs manually](#)

arn:aws:dynamodb:us-east-1:943059786379:table/Question

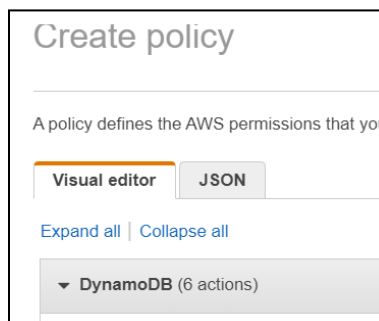
Region * ☐ Any

Account * ☐ Any

Table name * ☐ Any

[Cancel](#) [Add](#)

- l. The dialog box closes and returns you to the **Create Policy** page. Click the JSON tab.



Create policy

A policy defines the AWS permissions that you

[Visual editor](#) [JSON](#)

[Expand all](#) | [Collapse all](#)

▼ DynamoDB (6 actions)

- m. You are now looking at the policy you just created.

Create policy

12

A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor and using JSON. [Learn more](#)

Visual editor

JSON

Import managed policy

```

1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Sid": "VisualEditor0",
6       "Effect": "Allow",
7       "Action": [
8         "dynamodb:PutItem",
9         "dynamodb>DeleteItem",
10        "dynamodb:GetItem",
11        "dynamodb:Scan",
12        "dynamodb:Query",
13        "dynamodb:UpdateItem"
14      ],
15      "Resource": "arn:aws:dynamodb:us-east-1:943059786379:table/Question"
16    }
17  ]
18 }

```

- n. This time, you will add support for the Answer table manually. Change the “Resource” entry to an array containing a single entry as follows:

```

15   "Resource": [
16     "arn:aws:dynamodb:us-east-1:943059786379:table/Question"
17   ]

```

- o. If you made a mistake and the JSON is invalid, AWS will tell you with an X next to the line number. Below is an example of invalid JSON. Notice the X on line 18 and the error message, “Bad string”, because an extra comma was added on line 17. When the extra comma is removed, the error goes away.

```

15   "Resource": [
16     "arn:aws:dynamodb:us-east-1:943059786379:table/Question"
17   ],
18 }

```

Bad string

- p. To finish the policy, copy the line for Question and paste a copy right after it inside the array, with “Question” changed to “Answer”. Make sure to add a comma after the Question line. Your “Resource” array should now look like this:

```

15   "Resource": [
16     "arn:aws:dynamodb:us-east-1:943059786379:table/Question",
17     "arn:aws:dynamodb:us-east-1:943059786379:table/Answer"
18   ]

```

- q. Click the **Next: Tags**, then the **Next:Review** button. You will be taken to the **Review Policy** page.
- r. Name your policy “LambdaAccessToQuestionAndAnswerTable” and put in a description if you like.

Create policy

12

Review policy

Name*

LambdaAccessToQuestionAndAnswerTable

Use alphanumeric and '+-=, @-_' characters. Maximum 128 characters.

Description

This policy grants a Lambda access to the Question and Answer DynamoDB tables for Query, Scan, GetItem, DeleteItem, UpdateItem, and PutItem functions only.

Maximum 1000 characters. Use alphanumeric and '+-=, @-_' characters.

Summary

Q Filter

Service ▾	Access level	Resource	Request condition
Allow (1 of 230 services) Show remaining 229			
DynamoDB	Limited: Read, Write	Multiple	None

- s. Click the **Create Policy** button at the bottom of the page. When AWS is done creating the policy, it will show you the following banner:

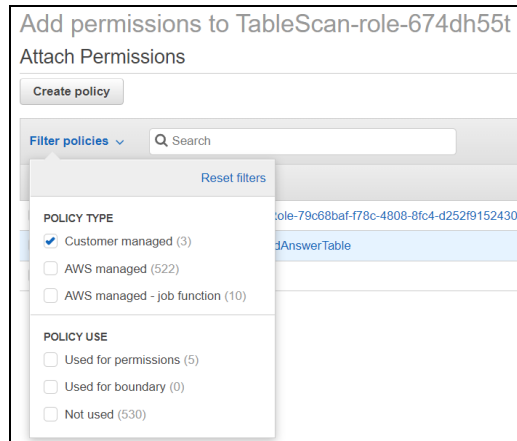
✓

LambdaAccessToQuestionAndAnswerTable has been created.

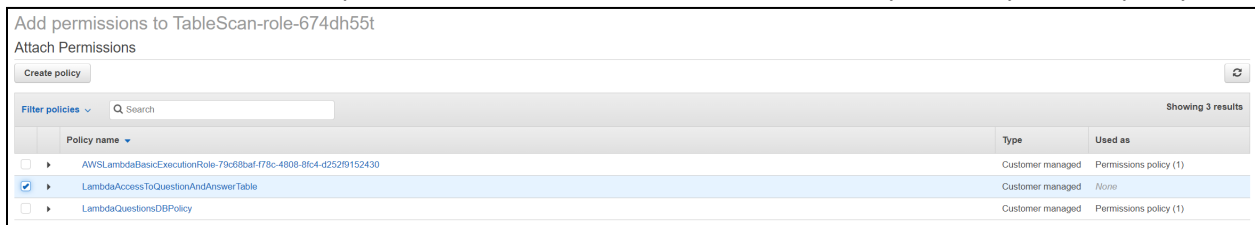
Create policy

Policy actions ▾

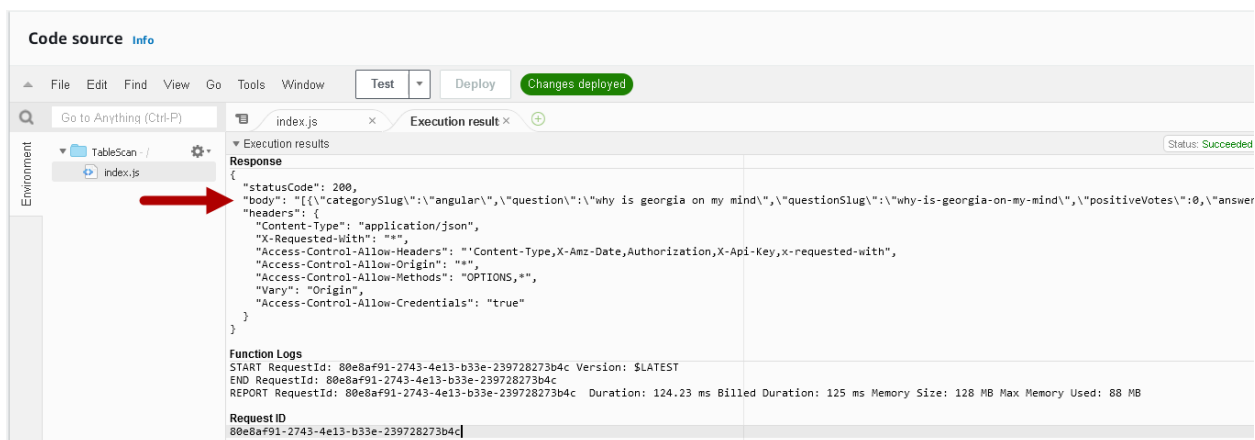
14. Go back to the browser tab from Step 9 that says **Attach Permissions**. In the **Filter policies** drop-down menu, select **Customer managed**.



15. Click outside the drop-down menu to close it. You should now see your newly created policy.



16. If you don't see your new policy, click the **Refresh** icon in the upper-left corner (the two arrows chasing each other in a circle). You may need to wait a minute or two.
17. Select the **LambdaAccessToQuestionAndAnswerTable** policy.
18. Click the **Attach Policy** button on the bottom of the page.
19. You have now granted your new Lambda permissions to perform six different actions against the Question and Answer tables. Go back to your Lambda page, click the **Code** tab, and run your new test, and you should see all the records in Question table returned.



20. Create a new test event called "QuestionWithFilter" with the following JSON:

```
{
```




```

    "resource": "/Questions",
    "httpMethod": "GET",
    "queryStringParameters": {
      "filter":
        "{\\"include\\":{\\"relation\\":\\"answers\\"},\\"where\\":{\\"categorySlug\\":\\"a
        ngular\\"}}"
    },
    "multiValueQueryStringParameters": {
      "filter": [

        "{\\"include\\":{\\"relation\\":\\"answers\\"},\\"where\\":{\\"categorySlug\\":\\"a
        ngular\\"}}"
      ]
    }
  }
}

```

21. Create a new test event called “AnswerWithoutFilter” with the following JSON:

```

{
  "resource": "/Answers",
  "httpMethod": "GET",
  "queryStringParameters": {},
  "multiValueQueryStringParameters": {}
}

```

22. Test your two new test events.
23. Congratulations! You have created a single Lambda to get filtered and unfiltered results from either table, and secured it so it can only perform six functions on two specific tables.

Part Two – Single-Record Fetch

You will build a single Lambda to work with both the Question and Answer tables again.

1. Create a Lambda named “GetSingleRecord”.
2. Copy the source code from the index.js file in the “GetSingleRecord” folder and replace the default Lambda code.
3. Make sure to click **Deploy**.
4. Attach the security policy “LambdaAccessToQuestionAndAnswerTable”, created in Part One, to this Lambda.
 - a. Click the **Permissions** tab.
 - b. Click the **Role Name**.
 - c. Click the **Attach Policies** button.
 - d. Filter your policies to just see **Customer Managed**.
 - e. Select **LambdaAccessToQuestionAndAnswerTable**.
 - f. Click **Attach Policy**.
5. Go back to the Lambda function tab and create a test event named “TestGetQuestion” with the following JSON:

```

{
  "resource": "/Questions/{id}",

```



```
"path": "/Questions/5eb59b7f80433e00045a7dfb",  
"httpMethod": "GET",  
"pathParameters": {  
  "id": "5eb59b7f80433e00045a7dfb"  
}  
}
```

```
5eb59b7f80433e00045a7dfb  
5eb59b7f80433e00045a7dfb
```

6. Create a test event named “TestGetAnswer” with the following JSON

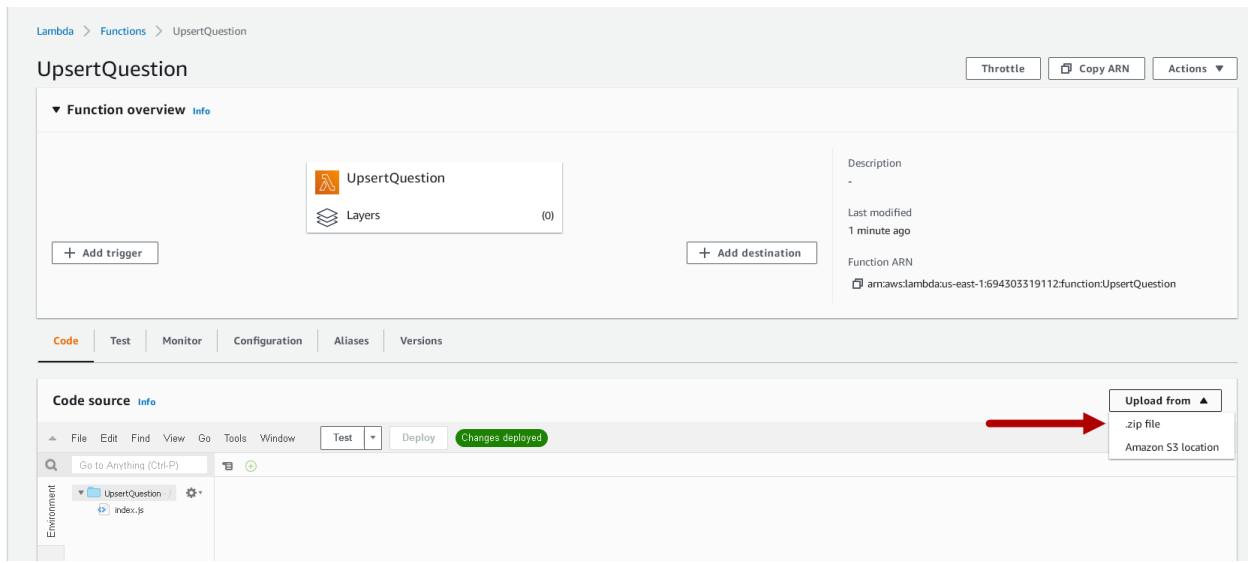
```
{  
  "resource": "/Answers/{id}",  
  "path": "/Answers/5b8629d2af53c20004793ac0",  
  "httpMethod": "GET",  
  "pathParameters": {  
    "id": "5b8629d2af53c20004793ac0"  
  }  
}
```

7. Test your new test events. That’s it; another single Lambda to handle both tables.
8. You need another type of single-record fetch as well. This one is only for Questions, and handles search scenarios required by the frontend. Create another Lambda named “FindOneQuestion”.
9. Copy the source code from the index.js file in the “FindOneQuestion” folder and replace the default Lambda code.
10. Make sure to click **Save**.
11. Attach the security policy “LambdaAccessToQuestionAndAnswerTable”, created in Part One, to this Lambda. You will skip testing this for now. That’s it; another pair of Lambdas to deal with single-record fetching.

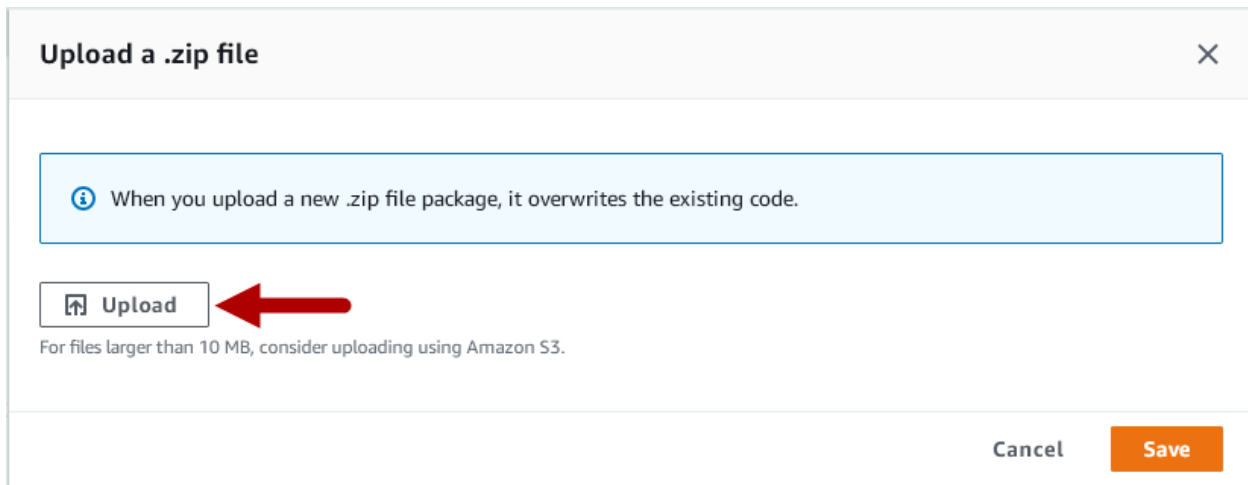
Part Three – Upserting Records

You can use a single Lambda for upserting (update if it exists or insert if it doesn’t), but not for both tables. To complete an upsert, you need to know the specific fields for each table. That would overcomplicate the Lambda, which is supposed to be a single function. Since insertion will require the generation of a new unique ID, our Lambda code will require an NPM dependency. When including dependencies with NodeJS, Lambda requires an uploaded ZIP file containing the source code and the dependencies. These two Lambdas will be a little different.

1. Create a Lambda named “UpsertQuestion” just as you created the other Lambdas.
2. Instead of modifying code in the editor to start, upload the “UpsertQuestion” ZIP file. Select the **Upload From** drop-down menu and select **.zip file**.




3. Click the **Upload** button.



4. Select the **UpsertQuestion** file from the ZIP folder provided to you in the Module Five Assignment Two Guidelines and Rubric and click **Open**.
5. Click the orange **Save** button in the upper-right corner.

Upload a .zip file

When you upload a new .zip file package, it overwrites the existing code.


Upload

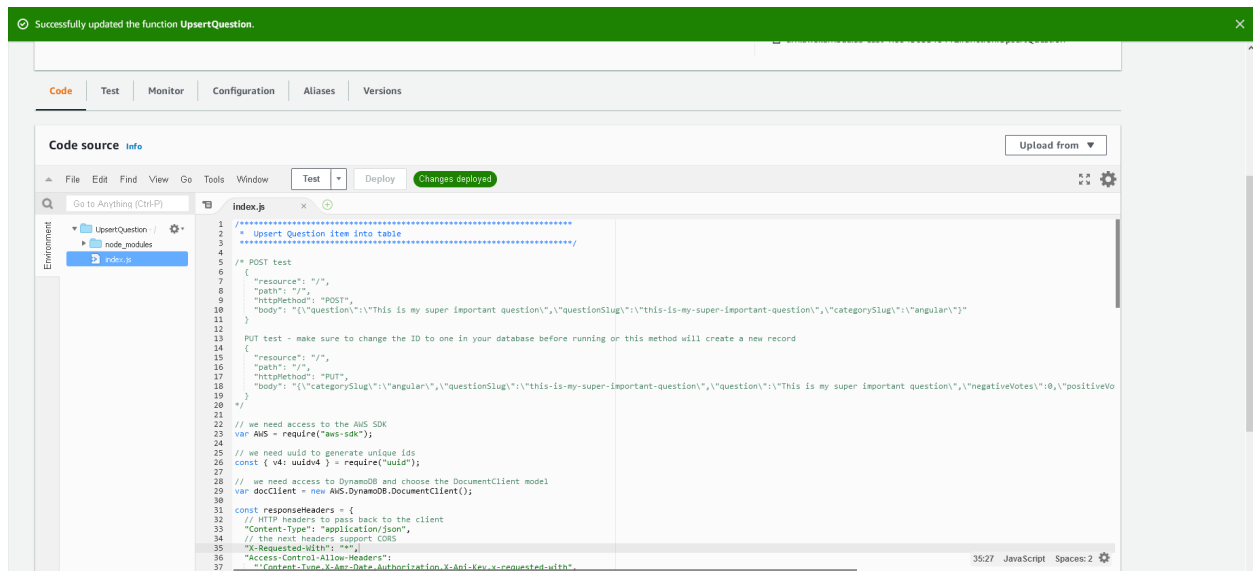
UpsertQuestion.zip (50.2 kB)

For files larger than 10 MB, consider uploading using Amazon S3.

Cancel

Save

- AWS will work for a few moments to load your code and present you with the following:



- You can edit your file without re-uploading the ZIP file, but you should not need to. Take note of the directory structure on the left side of the screen. It has your `index.js` file, and now it has a directory call `"node_modules"`. These are the items from inside the ZIP file.
- Now you need to attach the security policy created in Part One to this Lambda. Create a test event named `"TestPost"` with the following JSON:

```
{
  "resource": "/",
  "path": "/",
  "httpMethod": "POST",
  "body": "{\"question\": \"This is my super important question\", \"questionSlug\": \"this-is-my-super-important-question\", \"categorySlug\": \"angular\"}"
}
```



- Test your UpsertQuestion with the new TestPost event. NOTE: Copy the ID value from the test run – you will use it to test UpsertAnswer shortly.

▼ Execution results Status: Succeeded

Response

```
{
  "statusCode": 200,
  "body": "{\"Attributes\":{\"categorySlug\":\"\",\"question\":\"\",\"questionSlug\":\"\",\"id\":\"6dc7b151eb174255a2d6104296527636\",\"negativeVotes\":0,\"posit
  \"headers\": {
    \"Content-Type\": \"application/json\",
    \"X-Requested-With\": \"*\",
    \"Access-Control-Allow-Headers\": \"Content-Type,X-Amz-Date,Authorization,X-Api-Key,x-requested-with\",
    \"Access-Control-Allow-Origin\": \"*\",
    \"Access-Control-Allow-Methods\": \"OPTIONS,*\",
    \"Vary\": \"Origin\",
    \"Access-Control-Allow-Credentials\": \"true\"
  }
}
```

Function Logs

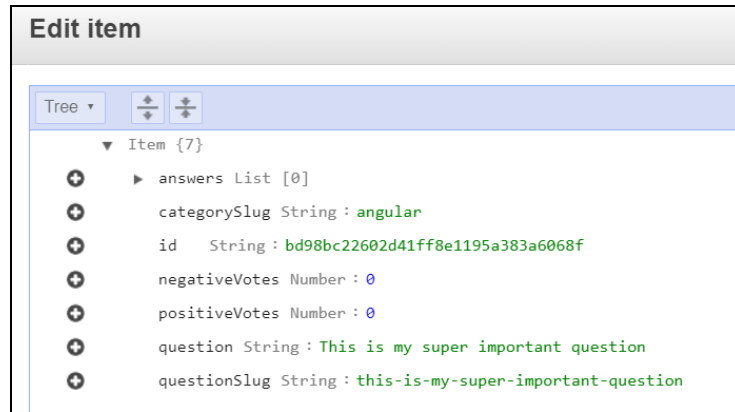
```
START RequestId: 2e5cc1a3-e5c0-4937-82b4-d87462ea92cd Version: $LATEST
END RequestId: 2e5cc1a3-e5c0-4937-82b4-d87462ea92cd
REPORT RequestId: 2e5cc1a3-e5c0-4937-82b4-d87462ea92cd Duration: 715.25 ms Billed Duration: 716 ms Memory Size: 128 MB Max Memory Used: 88 MB Init Duration: 41
```

Request ID
2e5cc1a3-e5c0-4937-82b4-d87462ea92cd

- You will not test updates at this time. This will be covered in Module Six.
- Create a Lambda named “UpsertAnswer”.
- Upload the **UpsertAnswer.zip** file as the code for your Lambda.
- Attach the security policy created in Part One to this Lambda.
- Create a test event named TestPost with the following JSON:

```
{
  "resource": "/",
  "path": "/",
  "httpMethod": "POST",
  "body": "{\"answer\":\"This is my super important
answer\",\"questionId\":\"bd98bc22602d41ff8e1195a383a6068f\"}"
}
```

- Replace the “questionId” value with the question ID from your test of **UpsertQuestion**.
 - If you don’t have the ID, then either run the UpsertQuestion test again and capture it, or get one from the DynamoDB console.
 - To get one from the AWS DynamoDB Console, go the console and select the Question table. Select any item by clicking on its link in the “id” column. This brings up the **Edit Item** window. Now click on the string by “ID”, and copy the value.



16. Test your UpsertAnswer with TestPost. Another pair of Lambdas done, and this time you learned how to include external Node modules in your code.

Part Four – Deleting Records

Time for the final Lambda. You can once again use a single Lambda here for both tables.

1. Create a Lambda named “DeleteRecord”.
2. Copy the source code from the index.js file in the “DeleteRecord” folder and replace the default Lambda code.
3. Make sure to click **Save**.
4. Attach the security policy “LambdaAccessToQuestionAndAnswerTable” created in Part One to this Lambda.
5. **Warning! Running the next two steps will delete the entries you put into your tables manually. You can always re-add them after you test.**
6. Create a test event named “TestDeleteQuestion” with the following JSON:

```
{
  "resource": "/Questions/{id}",
  "path": "/Questions/5eb59b7f80433e00045a7dfb",
  "httpMethod": "DELETE",
  "pathParameters": {
    "id": "5eb59b7f80433e00045a7dfb"
  }
}
```

7. Create a test event named “TestDeleteAnswer” with the following JSON:

```
{
  "resource": "/Answers/{id}",
  "path": "/Answers/5b8629d2af53c20004793ac0",
  "httpMethod": "DELETE",
  "pathParameters": {
    "id": "5b8629d2af53c20004793ac0"
  }
}
```



}

8. Done! You now have secure Lambdas to work on both tables for all four CRUD functions.