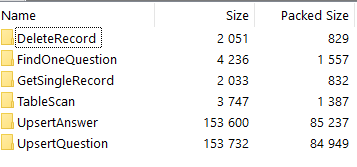
# CS 470 Module Five Assignment Two Guide

In Module Five Assignment One, you created and worked with your database tables all within the console. In this assignment, you will use 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:



## Part One: The Query Lambda

You will build a single Lambda that can query 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. Set the **Runtime** to **Node.js 20.x**. This runtime is the support for the legacy code that is used in this course. In production, the code would normally be adjusted to match the latest release version.
3. Under **Permissions**,select **Use Existing Role**.Then select **LabRole** from the drop-down menu.
4. Copy the source code from the index.js file in the TableScan folder and replace the default Lambda code.
5. Make certain to click **Deploy.**
6. Create a Lambda test event called QuestionWithoutFilter with the following JSON and then click **Save:**

{

"resource": "/Questions",

"httpMethod": "GET",

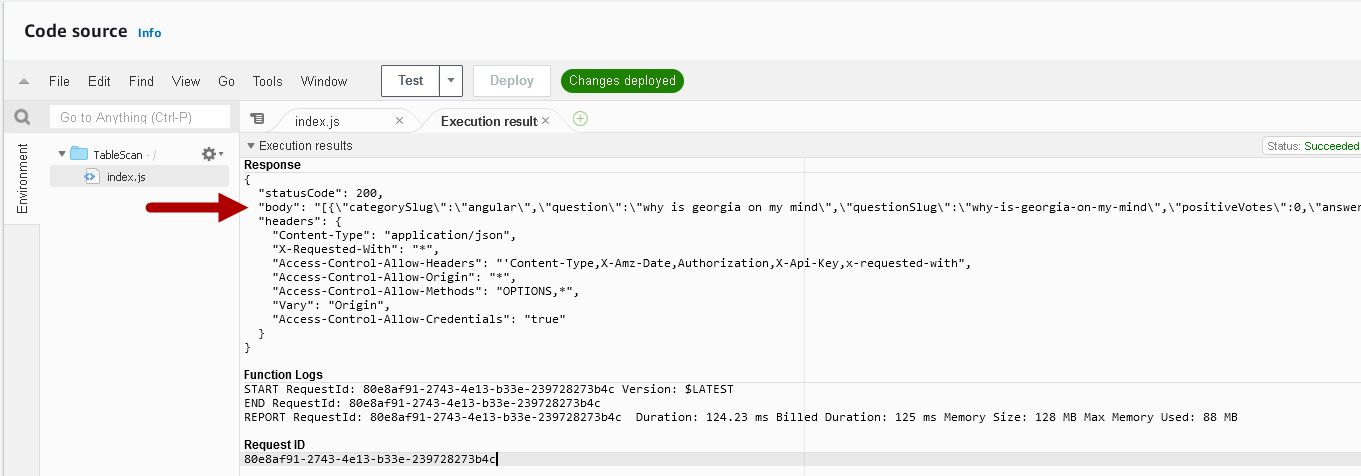
"queryStringParameters": {"include":{"relation":"answers"}},

"multiValueQueryStringParameters": {"include":{"relation":"answers"}}

}

Review Module Four if you need to remember how to create a Lambda test event

1. Click the **Code** tab on the main Lambda page and run your new test. You should see all the records in the Question table returned. The AWS console will launch the IAM Summary page.



1. Create a new test event called **QuestionWithFilter** with the following JSON:

{

"resource": "/Questions",

"httpMethod": "GET",

"queryStringParameters": {

"filter": "{\"include\":{\"relation\":\"answers\"},\"where\":{\"categorySlug\":\"angular\"}}"

},

"multiValueQueryStringParameters": {

"filter": [

"{\"include\":{\"relation\":\"answers\"},\"where\":{\"categorySlug\":\"angular\"}}"

]

}

}

1. Create a new test event called **AnswerWithoutFilter** with the following JSON:

{

"resource": "/Answers",

"httpMethod": "GET",

"queryStringParameters": {},

"multiValueQueryStringParameters": {}

}

1. Test your two new test events.

Congratulations! You have created a single Lambda to get filtered and unfiltered results from either table, and you’ve secured the Lambda 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 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 certain to click **Deploy**.
4. Return 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"

}

}

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

{

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

"path": "/Answers/5b8629d2af53c20004793ac0",

"httpMethod": "GET",

"pathParameters": {

"id": "5b8629d2af53c20004793ac0"

}

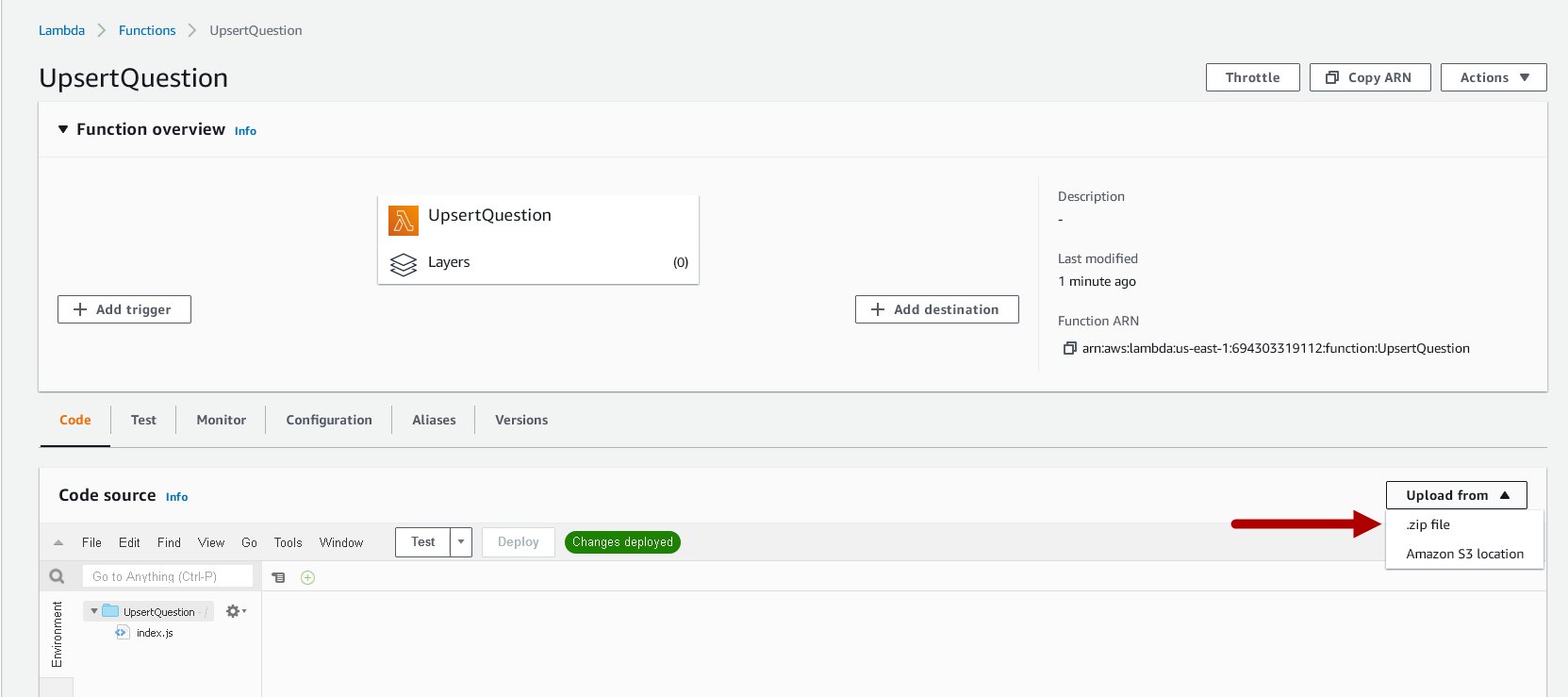
}

1. Test your new test events. That’s it! You have another single Lambda to handle both tables.
2. You need another type of single-record fetch as well. This one is only for Questions, and it handles search scenarios required by the frontend. Create another Lambda named “FindOneQuestion”.
3. Copy the source code from the index.js file in the **FindOneQuestion** folder and replace the default Lambda code.
4. Make certain to click **Save**.

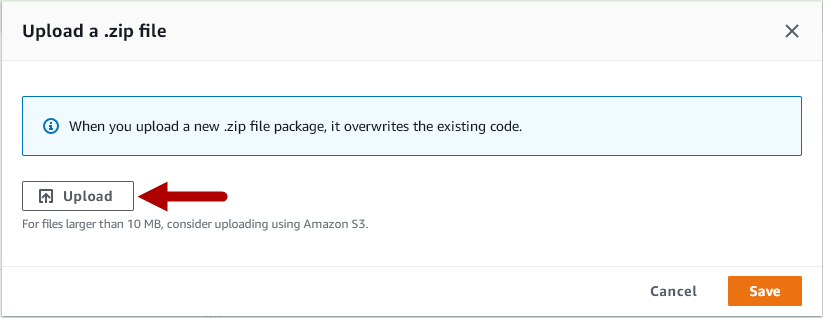
## Part Three: Upserting Records

Upserting is the act of inserting rows into a database table if they don’t already exist or updating the rows if they do exist. You can use a single Lambda for upserting but not for both tables. To complete an upsert, you must 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, your Lambda code will require an NPM dependency. When including dependencies with Node.js, Lambda requires an uploaded ZIP file containing the source code and the dependencies. These two Lambdas will be a little different.

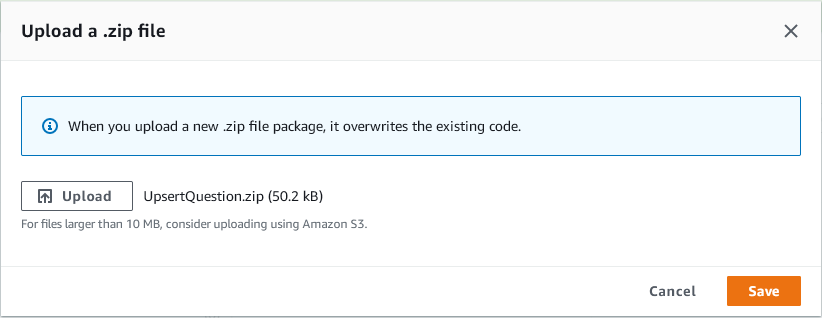
1. Create a Lambda named “UpsertQuestion” just as you created the other Lambdas.
2. Instead of modifying code in the editor to start, upload the **UpsertQuestion** **ZIP file**. Select the **Upload From** drop-down menu and select **.zip file**.



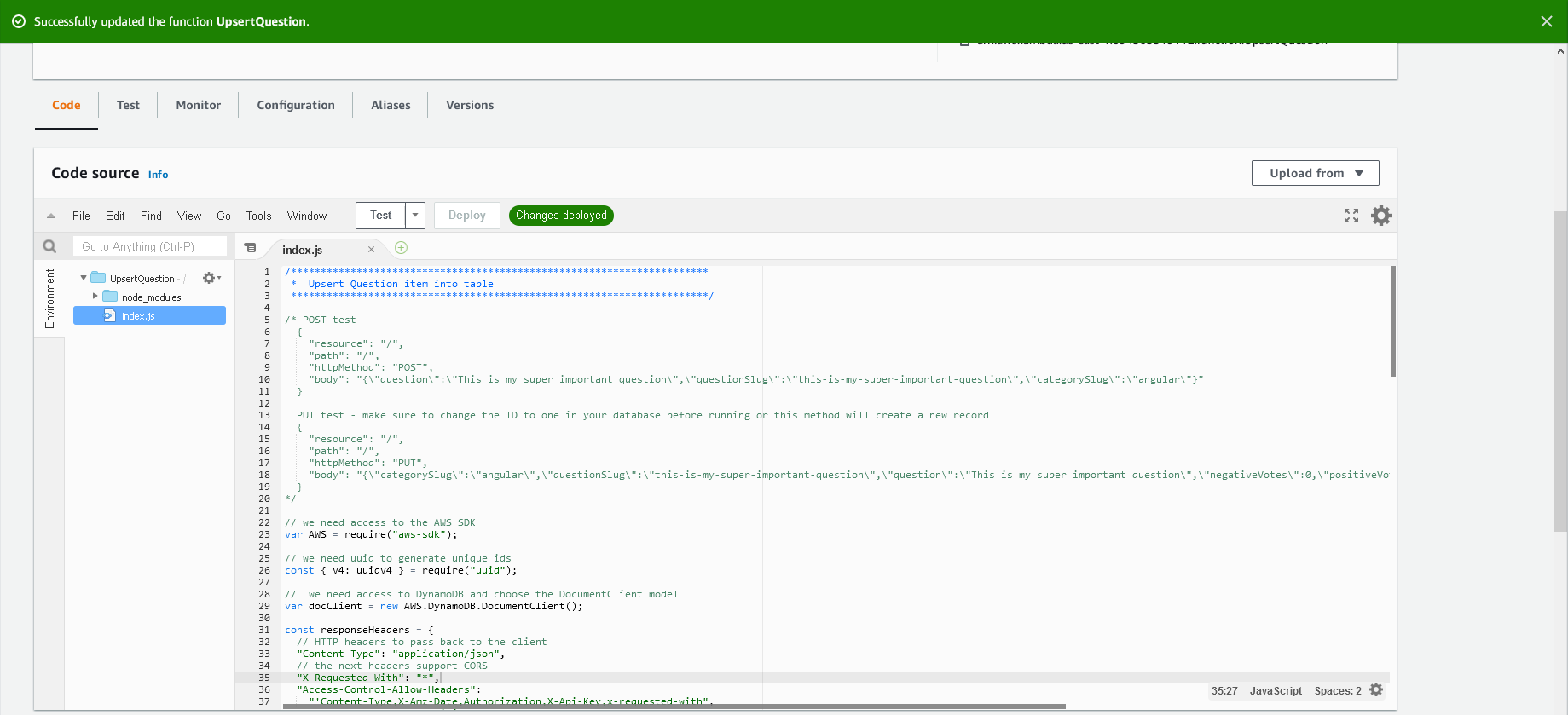
1. Click the **Upload** button.



1. Select the **UpsertQuestion** file from the ZIP folder provided in the Module Five Assignment Two Guidelines and Rubric and click **Open**.
2. Click the orange **Save** button in the lower-right corner.



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



1. 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. The directory structure has your index.mjs file. Now, the directory structure has a directory called **node\_modules**. These are the items from inside the ZIP file.

**Note:** Due to upgrades in legacy code, NodeJS 20.x only supports JS v3, a more modular type of JavaScript. You will notice that some of the JS files have .mjs extensions to be compatible with the new Node version.

1. 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\"}"

}

1. 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 with box around the id field:

Response
{
  "statusCode": 200,
  "body": "{\"Attributes\":{\"categorySlug\":\"\",\"question\":\"\",\"questionSlug\":\"\",\"id\":\"6dc7b151eb174255a2d6104296527636\",\"negativeVotes\":0,\"positiveVotes\":0}}",
  "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"
  }
}

1. You will not test updates at this time. Testing will be covered in Module Six.
2. Create a Lambda named “UpsertAnswer”.
3. Upload the **UpsertAnswer ZIP** **file** as the code for your Lambda.
4. Attach the security policy created in Part One to this Lambda.
5. Create a test event named “TestPost” with the JSON below. Make certain to replace the **questionId** value with the question ID from your test of **UpsertQuestion.** 
   1. If you don’t have the ID, either run the UpsertQuestion test again and capture it or get one from the DynamoDB console.
   2. To get an ID from the AWS DynamoDB Console, go to the console and select the Question table. Select any item by clicking on its link in the **id** column. This action brings up the **Edit Item** window. Now, click on the string by **ID** and copy the value.

{

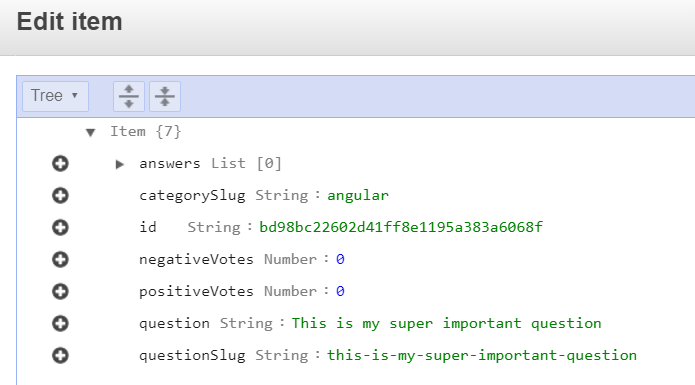
"resource": "/",

"path": "/",

"httpMethod": "POST",

"body": "{\"answer\":\"This is my super important answer\",\"questionId\":\"bd98bc22602d41ff8e1195a383a6068f\"}"

}



1. Test your UpsertAnswer with TestPost.

Another pair of Lambdas is done! 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.mjs file in the **DeleteRecord** folder and replace the default Lambda code.
3. Make certain 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 manually put into your tables. 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"

}

}

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

{

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

"path": "/Answers/5b8629d2af53c20004793ac0",

"httpMethod": "DELETE",

"pathParameters": {

"id": "5b8629d2af53c20004793ac0"

}

}

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