Exercise: Adding multiple items to Amazon DynamoDB using the AWS Software Development Kit (AWS SDK)

In this exercise, you will learn how to *develop* with Amazon DynamoDB by using the AWS Software Development Kit (AWS SDK). Following the scenario provided, you will add multiple items into multiple DynamoDB tables using the AWS SDK. This exercise gives you hands-on experience with both Amazon DynamoDB and AWS Cloud9.

Objectives

After completing this exercise, you will be able to use the AWS SDKs to do the following:

- Uploading multiple items to multiple DynamoDB tables.

Story continued

So you now have a proof of concept website that on page load will display dragon cards (albeit 2) using data stored in DynamoDB.

Your website communicates with your API Gateway backend, and currently returns everything in the database.

This is fine for a proof of concept, and when you call Mary and show her the website she is very happy with what you are doing. She tells you that she already has come up with an idea for a card template and asks if you can integrate that too?

You say no problem, and ask her to email it to you along with the JSON files for all the dragon data that she has been promising you all week.

She apologizes about the delay and tells you, you will have it in the next 5 minutes via email.

38 minutes later, you get the email. Your head sinks into your hands, as you see that she has a much more complex data requirement than the one you envisioned.

It is basically a relational data structure with dragons of different types, having different skills and modifiers. Essentially she has given you all the data that would be required for an actual game engine.

You are not building the game engine. She has other people lined up for that, thank goodness. However she still wants you to use DynamoDB to store this data for her, and leverage that to display card data on the website.

You already know how to upload one item at a time with code, but that isn't feasible with the amount of data she has given you.

You need to come up with a script that can upload multiple items to multiple tables (using batch processing).

The 4 JSON files she sent you have the following structure:

DragonStatsTableOne [25 items]

```
[{
    "damage_int": 9,
    "description_str": "Amazing dragon",
    "dragon_name_str": "Cassidiuma",
    "family_str": "red",
    "location_city_str": "las vegas",
    "location_country_str": "usa",
    "location_neighborhood_str": "spring valley",
    "location_state_str": "nevada",
    "protection_int": 3
}...]
```

DragonStatsTableTwo [25 items]

(as above) - *she just broke them into manageable files, rather than use one big file.*

DragonCurrentPowerTable [1 item] - sample item only (table is dynamic)

```
[{
  "current_endurance_int": 3,
  "current_will_not_fight_credits_int": 2,
  "dragon_name_str": "Cassidiuma",
  "game_id_str": "56syjdh8756",
}...]
```

DragonBonusAttackTable [5 items]

```
[{
  "breath_attack_str": "acid",
  "description_str": "spews acid",
  "extra_damage_int": 3,
  "range_int": 5
}...]
```

DragonFamilyTable [4 items]

```
[{
  "breath_attack_str": "acid",
  "damage_modifer_int": -2,
  "description_str": "Better defense",
  "family_str": "green",
  "protection_moodifier_int": 2
 }..]
```

You decide to create a table for each JSON file, and structure it a bit like this:

Dragon Stats Table

Using `dragon name` as the Primary Key (PK) as you will want to search for a dragon by name.

dragon_name (PK)	damage	description	protection	family	location_country	location_city	location_state	Location_neighborhood
cassidiuma	9	Amazing dragon	3	red	usa	las vegas	nevada	spring valley

Dragon current power table [1 per game]

Mary tells you that these entries will be created dynamically per game, and she has given you an example.

Although you won't be referring this data in your card viewing website, she gave it to you anyway for context.

You decide that the string `game_id` (String) should be the primary key (PK). They will want to search on `game_id` and get a list of all the dragons in play.

game_id (PK)	dragon_name	current_will_not_fight_credits	current_endurance (dynamic)
56syjdh8756	cassidiuma	2	8

Dragon Bonus Attack Table

As you think that they might want to search for details on a type of `breath_attack` you choose that as your primary key for this table.

Also you think it would be nice to see if an attack is "in range", so you add a sort key on `range` to find out if say "a water attack is in `range` to do damage".

breath_attack(PK)	description	extra_damage	range (SK)
acid	spews acid	3	5
electricty	bolts fly from mouth	3	5
water	high pressure jet over a large area	1	10
fear	Prevent all attacks next round	0	4
fire	Short blast of fire	8	4

Dragon Family Table

Later on in the game engine, the developers will likely need to be able to bring up information about modifiers relating to the dragon type (family). You decide to use `family` as the Primary Key (PK).

You feel that this table should probably be merged with the bonus attack table, but you keep it the way it is for now. *[You might optimize this later on]*.

breath_attack	damage_modifer	description	family (PK)	protection_modifier
acid	-2	Better defense	green	2
fire	2	Attacks faster	red	-2
water	1	Happy in water	blue	-1
fear	0	Prefers to bite	black	0

Time to load data

You remember reading about different ways to get data into DynamoDB, and you figure that using the SDK and the **batchWriteItem** method would be the best option here.

Prepare the exercise

Before you can start this exercise, you need to import some files and install some modules in the AWS Cloud9 environment that has been prepared for you.

1. Head over to Cloud9.
2. As you have been working with Cloud9 for the previous exercises we need to make sure you are in the right path in the thermal.

   ```
   cd ~/environment
   ```

You should probably close any AWS Cloud9 tabs you have open, and collapse any inactive folders, as it can get messy with too many code tabs open.

3. You will need to seed your AWS Cloud9 filesystem with the `lab3` content, so go to the AWS Cloud9 **bash terminal** (at the bottom of the page) and run the following `wget` command:

```
wget \
https://s3.amazonaws.com/awsu-hosting/edx_dynamo/c9/dynamo-seed/lab3.zip \
-P /home/ec2-user/environment
```

4. To unzip the workdir.zip file, run the following command:

```
unzip lab3.zip
```

5. To keep things clean, run the following commands to remove the zip file:

```
rm lab3.zip
```

6. Select the black arrow next to the `lab3` folder to expand it (collapsing lab 2 and 1 if they are still open). Notice that there is a solution folder in `lab3` along with a resources folder. Throughout this exercise, **don't peek at the solution unless you really get stuck. Always TRY to code first.** We will come back to the resources folder shortly. For now let's set the terminal path to the correct folder.

```
cd lab3
```

As this course is self-paced, often people will start the lab then come back to it later. In the interim period, we may have made adjustments to the code.

Ensure that inside your lab3 folder that the name of the version markdown file is matching the version number at the top of this document.

If they are out of sync, you will run into problems. To get them synced, simply remove the old folder, ensure you are in the right path in the terminal, and run through the wget steps above one more time.

This lab and future labs are especially prone to getting out of sync, because all the website iterations have alredy been loaded into lab2.

So, if your website is not working as expected for the lab, please double check that lab2 (that contains the website) has not had any updates recently. If your lab2 code versions don't match the version on the lab2 document, you will need to re-upload the website by following the steps one more time from lab2.

You are now ready to do the exercise tasks.

Step 1A): Create multiple DynamoDB tables using the SDK

You already know how to create tables using the AWS-SDK, so creating a few more should be a breeze. Follow these steps, to create the following tables with respective Primary Keys (see tables above).

You have done this before so replace the <FMI>'s in the following script.

1. Open up `lab3/create_multiple_tables.js` and edit as needed.
2. Once you are done choose `File` and `Save`.
3. Run the following:

```
$ node create_multiple_tables.js
```

You should see output like the following:

```
[ { TableDescription:
    { AttributeDefinitions: [Array],
      TableName: 'dragon_stats',
      KeySchema: [Array],
      TableStatus: 'CREATING',
      CreationDateTime: 2019-05-17T18:43:31.060Z,
      ProvisionedThroughput: [Object],
      TableSizeBytes: 0,
      ItemCount: 0,
      TableArn: 'arn:aws:dynamodb:us-east-1:
<000000000000>:table/dragon_stats',
      TableId: '6bde2328-7075-4f28-b5f7-7f21df30df36',
      BillingModeSummary: [Object] } },
  { TableDescription:
    { AttributeDefinitions: [Array],
      TableName: 'dragon_current_power',
      KeySchema: [Array],
      TableStatus: 'CREATING',
      CreationDateTime: 2019-05-17T18:43:31.015Z,
      ProvisionedThroughput: [Object],
      TableSizeBytes: 0,
      ItemCount: 0,
      TableArn: 'arn:aws:dynamodb:us-east-1:
<000000000000>:table/dragon_current_power',
      TableId: 'b8b7dbea-fa97-4f21-a59c-99210fd2192f',
      BillingModeSummary: [Object] } },
  { TableDescription:
    { AttributeDefinitions: [Array],
      TableName: 'dragon_bonus_attack',
      KeySchema: [Array],
      TableStatus: 'CREATING',
```

```
      CreationDateTime: 2019-05-17T18:43:31.050Z,
      ProvisionedThroughput: [Object],
      TableSizeBytes: 0,
      ItemCount: 0,
      TableArn: 'arn:aws:dynamodb:us-east-1:
<000000000000>:table/dragon_bonus_attack',
      TableId: 'aff78f2f-990c-432e-bd90-9eb0c00939aa',
      BillingModeSummary: [Object] } },
  { TableDescription:
    { AttributeDefinitions: [Array],
      TableName: 'dragon_family',
      KeySchema: [Array],
      TableStatus: 'CREATING',
      CreationDateTime: 2019-05-17T18:43:31.018Z,
      ProvisionedThroughput: [Object],
      TableSizeBytes: 0,
      ItemCount: 0,
      TableArn: 'arn:aws:dynamodb:us-east-1:
<000000000000>:table/dragon_family',
      TableId: '46498747-903c-4a11-b551-4207649c1930',
      BillingModeSummary: [Object] } } ]
HowFastWasThat: 133.393ms
```

Note: the only change in terms of code vs what you did last time in lab 1, is that we are using the Async Await feature of Node to allow us to use Promises. Don't get hung up on this stuff, it's just a way to carry out multiple calls to the DB in one script.

Also you may have to wait up to **five minutes** before all your tables show as **ACTIVE**.

○	dragon_bonus_attack	Active	breath_attack (String)	range (Number)	0	On-Demand
○	dragon_current_power	Active	game_id (String)	-	0	On-Demand
○	dragon_family	Active	family (String)	-	0	On-Demand
○	dragon_stats	Active	dragon_name (String)	-	0	On-Demand
○	dragons	Active	dragon_name (String)	-	0	On-Demand

*Make sure your tables say **Active** before moving on.*

Step 1B): Add Dragon data to multiple DynamoDB tables using the BatchWriteItem method (SDK)

1. Open the SDK docs for the language you want to code in, and find the method for creating new items as a batch inside an existing DynamoDB table. Find out the correct method name and establish what parameters you need to pass in.

Language	AWS documentation deep link
NODE.JS (8.16.0)	https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html#batchWriteItem-property

Time to write some code that adds a a batch of items to each of the DynamoDB tables.

1. Open up the `seed_dragons` file inside the code folder you are working from, by double clicking on it.
2. Have the SDK docs open (as above) to help you.
3. Replace the <FMI> sections of the code in that file, so that the code uploads all the dragon data into new tables. You table should have been called **dragons** and will be in **us-east-1**.
4. You have the dragon data in the JSON files.
5. Save the file.
6. Double check your code is correct by looking at the solution file.
7. Go to the terminal and run your file
8. `node seed_dragons.js`

IF YOU GET STUCK, OR IT IS NOT WORKING, SIMPLY COPY THE CODE SITTING IN THE RESPECTIVE SOLUTION FOLDER.

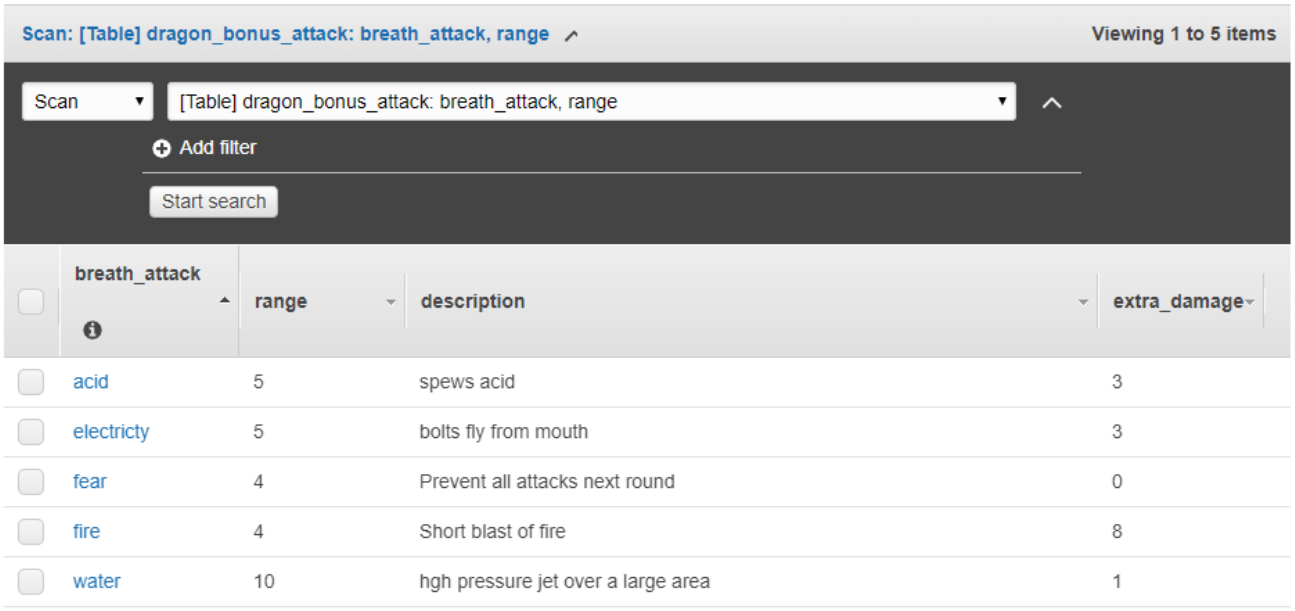
Confirm that your code worked.

You should see the following out after running:

`$ node seed_dragons.js`

```
[ { UnprocessedItems: {} },
  { UnprocessedItems: {} },
  { UnprocessedItems: {} },
  { UnprocessedItems: {} },
  { UnprocessedItems: {} } ]
HowFastWasThat: 183.082ms
```

We can also verify in the DynamoDB console refresh the page, and look at the **items** tab of one of the tables. In this case let's look at the `dragon_bonus_attack` table.



breath_attack	range	description	extra_damage
acid	5	spews acid	3
electricty	5	bolts fly from mouth	3
fear	4	Prevent all attacks next round	0
fire	4	Short blast of fire	8
water	10	hgh pressure jet over a large area	1

Awesome. We have all the dragon data in the database. The challenge is that the code you wrote earlier that scans the table (remember `lab2`) was written to accommodate a different table name **dragons**. And has completely different schemes.

Step 1C): Delete the old table

We are no longer going to use the mock dragon database you created in lab1. So this would be a great time to clean it up.

Run this command from the Cloud 9 terminal to remove the old dragon table. Out with the old in with the new.

```
aws dynamodb delete-table --table-name dragons
```

You should see something like this:

```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:<000000000000>:table/dragons",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 0,
            "ReadCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "TableName": "dragons",
        "BillingModeSummary": {
            "LastUpdateToPayPerRequestDateTime": 1558029188.272,
            "BillingMode": "PAY_PER_REQUEST"
        },
```

```
        "TableStatus": "DELETING",
        "TableId": "78e4aeb0-c70e-4678-bbee-60e97f7f3524",
        "ItemCount": 0
    }
}
```

Now our website will break because the API is pointing to a Lambda function that references the table you just removed.

Fear not. All we need to do is edit our Lambda function to point to the right table.

The other day you updated your website to allow a user to search for a specific dragon and bring up just that dragon. They can still select "show all dragons", but you thought it was a nice feature so you added it.

So while we are in there updating the Lambda function, we will have you update the code a little more, to allow for more functionality.

The website needs to show a single dragon card If the website passes a dragon name in the request. If they don't choose a dragon, we just give them a table dump ;)

Step 2: Adjust the lambda function to work with the new database, and filter for just a specific dragon (if asked)

So your current API function takes in an empty POST request and spits out everything in the dragon data table.

You will need to modify `lab3/scan_dragons.js` to allow a user to search for a dragon by name or search for all dragons!

CODE STEPS:

The solution can be found in the solution folder if you get stuck.

You "could" edit the code directly in the Lambda function, but you would like to test it in AWS Cloud9 first. Then you can copy and paste it into Lambda.

You want to be able to test a "scan all" and a show a specific dragon.

Open the SDK docs and find the method for scanning and filtering with DynamoDB. Find the correct method name and establish what parameters you need to pass in.

Language	AWS documentation deep link
NODE.JS (8.16.0)	https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html#scan-property

Time to write some code that adds a a batch of items to each of the DynamoDB tables.

1. Open up the `scan_dragons` file double clicking on it. (note use the scan_dragons.js file inside the lab3 folder **not** the old lab2 folder)
2. Collapse and close any files you are not using and have the SDK docs open (as above) to help you.
3. Replace the <FMI> sections of the code in that file, so that the code scans all the dragon data from the `dragon_scan` table.
4. Run the following:

```
node scan_dragons.js test "Cassidiuma"
```

5. You should see the following:

```
Local test for a dragon called Cassidiuma
null [ { location_neighborhood: { S: 'poplar str' },
    damage: { N: '7' },
    location_city: { S: 'colby' },
    family: { S: 'green' },
    description:
     { S: "Cassidiuma is the personal protector and knight of the dragon
queen Methryl. She is the queen's most loved and feared warrior." },
    protection: { N: '10' },
    location_country: { S: 'usa' },
    location_state: { S: 'kansas' },
    dragon_name: { S: 'Cassidiuma' } } ]
```

6. Run the following:

```
node scan_dragons.js test
```

It should return ALL the dragons in the table.

Step 3: Copy and paste new code into the lambda console (updating only Lambda and test steps)

1. Pivot back to the console by choosing **AWS Cloud9** in the upper left.
2. Choose **Go To Your Dashboard** and choose **Services** and search for **Lambda**.
3. Open the **DragonSearch** function.
4. Replace the contents of `index.js` with the new code from `scan_dragons`.
5. Choose **Save**.

6. Choose **Test** (using the **DragonScan** test event).
7. It should show you a list of dragons in the **Details** section under Execution results.
8. Now try searching for a dragon by name.
9. Click the drop down area next to **Test** and choose **Configure test events**.
10. Select the radio button next to **Create new test event**.
11. Under **Event name** enter `JustOneDragon`.
12. Replace the existing content with:

```
{
    "dragon_name_str": "Cassidiuma"
}
```

13. Choose **Create** and then choose **Test**.
14. In the **Execution results** you should see the following:

```
[
  {
    "location_neighborhood": {
      "S": "poplar st"
    },
    "damage": {
      "N": "7"
    },
    "location_city": {
      "S": "colby"
    },
    "family": {
      "S": "green"
    },
    "description": {
      "S": "Cassidiuma is the personal protector and knight of the dragon
queen Methryl. She is the queen's most loved and feared warrior."
    },
    "protection": {
      "N": "10"
    },
    "location_country": {
      "S": "usa"
    },
    "location_state": {
      "S": "kansas"
    },
```

```
    "dragon_name": {
      "S": "Cassidiuma"
    }
  }
]
```

Step 4: Test the existing REST API endpoint and connect it to your website

Good news the API you set up in lab2 is pointing to the ARN (Amazon Resource Name) of that Lambda function you just edited.

This means you do not need to re-deploy your API gateway. However, a quick test before checking the website is best practice. So, follow these steps.

1. Choose **Services** and search for **API Gateway**.
2. Choose the **DragonSearchAPI**.
3. Choose **POST** and select **TEST**.

   You can leave the "request body" blank or use "All" shown below:

   ```
   {
     "dragon_name_str": "All"
   }
   ```

4. Choose **Test**.

You will see all the dragon information returned at the right.

Now let's test for a specific dragon.

5. In the **Request Body** paste the following:

   ```
   {
     "dragon_name_str": "Cassidiuma"
   }
   ```

6. Choose **Test**.
7. You should see the following in the **Response Body**

   ```
   [
     {
       "location_neighborhood": {
         "S": "poplar st"
       },
   ```

```
    "damage": {
      "N": "7"
    },
    "location_city": {
      "S": "colby"
    },
    "family": {
      "S": "green"
    },
    "description": {
      "S": "Cassidiuma is the personal protector and knight of the dragon
queen Methryl. She is the queen's most loved and feared warrior."
    },
    "protection": {
      "N": "10"
    },
    "location_country": {
      "S": "usa"
    },
    "location_state": {
      "S": "kansas"
    },
    "dragon_name": {
      "S": "Cassidiuma"
    }
  }
]
```


Great news your web back-end API is all good to go. Your old website was expecting different data.

Luckily you had some time over the weekend to update your site and push it to S3.

So you can test the newer version 2.0 *(index2.html)* of your web front end.

NOTE: This was uploaded in lab 2, you just need to navigate to `index2.html`* Also the old website (index.html) will now no longer work, which is to expected as we just altered the backend.

If you don't have a tab with your website open in chrome, do the following to find that URL.

1. Choose **Services** and search for **s3**.
2. Choose the S3 bucket that was created in the previous lab: `2019-05-16-dynamolab-er-102`
3. Choose **Properties** and **Static website hosting**.
4. Open the Endpoint in a new browser tab and append `/index2.html` to it:

```
http://<your s3 bucket>.s3-website-us-east-1.amazonaws.com/index2.html
```

You should see all the dragons, AND be able to choose just one dragon!

Congrats Lab 3 completed, you have lots of Dragon data inside multiple tables in DynamoDB wired up to your website.

AWESOME! Mary will be happy!