Using DynamoDB

Overview:

AWS DynamoDB is a NO SQL solution to databases. It is a fully managed cloud database supportig both document and key-value store models. DynamoDB tables do not require a schema. You do not need to define any attributes, or data types at table creation time. You just need to define the Primary Key(s).

This document will provide an overview of DynamoDB and how to access and use many of the features using the AWS CLI and the AWS SDK.

Introduction to DynamoDB:

Similar to other AWS services, with DynamoDB you don't have to get involved with hardware provisioning, setup and configuration, replication, software patching, or other database maintenance or set-up steps.

There are many useful features in DynamoDB. One worth noting is the ability to automatically delete expired or older, no longer needed items from tables to reduce storage usage and cost. DynamoDB also scales as needed and doesn't suffer some of the issues associated with scaling large relational databases.

Core components of DynamoDB include tables, items, and attributes. Similar to relational databases, a table is a collection of data. Each table can contain multiple items which are related to the theme or topic of the table. Items can be thought of as rows or records in a relational database. Items are comprised of multiple attributes. For example, a Student table will have multiple items and each item might consist of attributes such as first name, last name, and email address.

Each item in the table will have a unique identifier or primary key. For example, in the Student table the primary key might be the email address or some other unique student identifier. One main difference between SQL and NoSQL databases is that with NoSQL databases, neither the attributes nor their data types need to be defined at creation. Each item can have its own distinct attributes. This makes for some very interesting and ad-hoc data structures. For example, one item in the Student table may include attributes for first name, last name and email address whereas another item may contain first name, last name, email address as well as the student's middle name and country of birth.

Primary Keys:

DynamoDB supports simple and composite primary keys. A simple primary key is one attribute that clearly and uniquely identifies the item in the table. A composite primary key consists of up to two attributes that uniquely identifies an item in the table. An example of a good simple primary key would be a Driver's License number or Email Address. Each of these keys would uniquely identify an individual. However; sometimes more than one item is required to unique identify an item. For example, consider a Semesters table. Attributes such as Semester (Fall, Spring, Summer) would be a component of the primary key but would not be unique. If we added the Year as another attribute, the composite key should uniquely identify the semester. For example, Fall 2019, or Spring 2020.

Note that indexes are available in NoSQL databases as well. They are not required but they can be used to further enhance queries and searches for items within a table.

DynamoDB uses primary keys to uniquely identify each item in a table and secondary indexes to provide more query flexibility

Core Elements in DynamoDB:

DynamoDB has functions, callable through AWS CLI and other Programming API's, supporting most table and database requirements. As described in the AWS DynamoDB documentation, the following functions are available:

- CreateTable – Creates a new table. Optionally, you can create one or more secondary indexes, and enable DynamoDB Streams for the table.
- DescribeTable– Returns information about a table, such as its primary key schema, throughput settings, index information, and so on.
- ListTables – Returns the names of all of your tables in a list.
- UpdateTable – Modifies the settings of a table or its indexes, creates or remove new indexes on a table, or modifies DynamoDB Streams settings for a table.
- DeleteTable – Removes a table and all of its dependent objects from DynamoDB.
- PutItem – Writes a single item to a table. You must specify the primary key attributes, but you don't have to specify other attributes.
- BatchWriteItem – Writes up to 25 items to a table. This is more efficient than calling PutItem multiple times because your application only needs a single network round trip to write the items. You can also use BatchWriteItem for deleting multiple items from one or more tables.
- GetItem – Retrieves a single item from a table. You must specify the primary key for the item that you want. You can retrieve the entire item, or just a subset of its attributes.
- BatchGetItem – Retrieves up to 100 items from one or more tables. This is more efficient than calling GetItem multiple times because your application only needs a single network round trip to read the items.
- Query – Retrieves all items that have a specific partition key. You must specify the partition key value. You can retrieve entire items, or just a subset of their attributes. Optionally, you can apply a condition to the sort key values, so that you only retrieve a subset of the data that has the same partition key. You can use this operation on a table, provided that the table has both a partition key and a sort key. You can also use this operation on an index, provided that the index has both a partition key and a sort key.
- Scan – Retrieves all items in the specified table or index. You can retrieve entire items, or just a subset of their attributes. Optionally, you can apply a filtering condition to return only the values that you are interested in and discard the rest.
- UpdateItem – Modifies one or more attributes in an item. You must specify the primary key for the item that you want to modify. You can add new attributes and modify or remove existing attributes. You can also perform conditional updates, so that the update is only successful when a user-defined condition is met. Optionally, you can implement an atomic counter, which increments or decrements a numeric attribute without interfering with other write requests.
- DeleteItem – Deletes a single item from a table. You must specify the primary key for the item that you want to delete.

- BatchWriteItem – Deletes up to 25 items from one or more tables. This is more efficient than calling DeleteItem multiple times because your application only needs a single network round trip to delete the items. You can also use BatchWriteItem for adding multiple items to one or more tables.

AWS-CLI Commands

The section will provide examples of calling a few of the more popular functions for creating a table, putting an item in the table, getting an items from the table, updating an item in the table and deleting an item in the table. The examples will use the AWS CLI to perform the functions and provide verification through screen captures and views in the AWS management console.

Recall, we use the Cloud9 environment to run the AWS CLI commands for this class.

Documentation for the AWS-CLI for specific commands, sub-commands and options for DynamoDB is found here:

https://docs.aws.amazon.com/cli/latest/reference/dynamodb/index.html#cli-aws-dynamodb

Create a table:

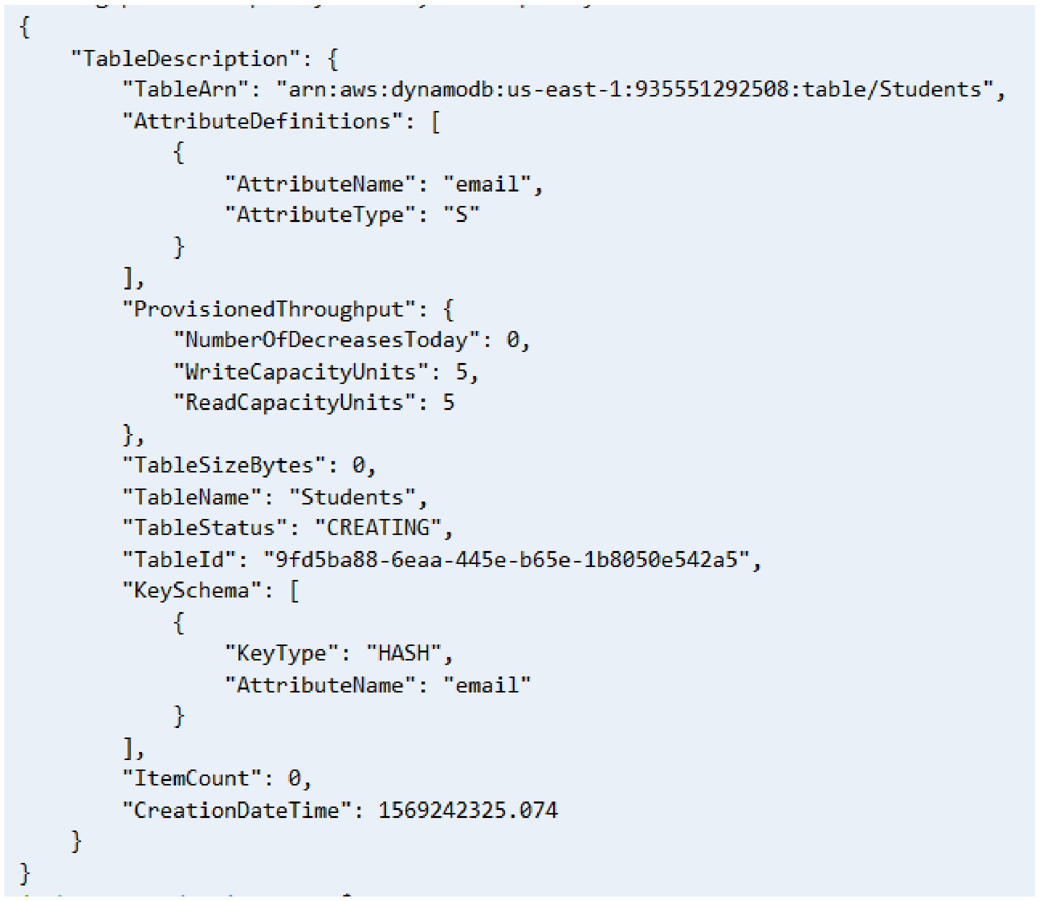
To create a table named Students with a primary key of email, the following command should be executed at the command prompt:

```
aws dynamodb create-table --table-name Students --attribute-definitions
AttributeName=email,AttributeType=S --key-schema
AttributeName=email,KeyType=HASH --provisioned-throughput
ReadCapacityUnits=5,WriteCapacityUnits=5
```

Note a couple of important requirements for this call:

1. The Attribute definitions should only list the required Attributes that help uniquely identify the item in the table. In this example, only email is listed. The email will be the HASH.
2. When you add items to the table, additional attributes (e.g. Lastname, Firstname, StateofResidence) can be added on the fly
3. Attributes are case sensitive. Email is not the same as email.
4. Valid AttributeTypes for primary keys include S (for string), N(for number) and B (for binary). Non-primary key attributes may also be Lists, Sets and Maps.

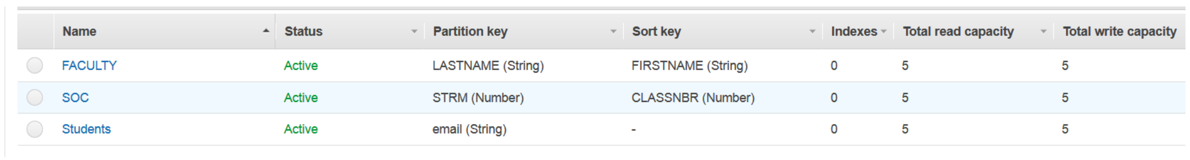
A successful response will return in output in JSON format as shown in Figure 1.



```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:935551292508:table/Students",
        "AttributeDefinitions": [
            {
                "AttributeName": "email",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "Students",
        "TableStatus": "CREATING",
        "TableId": "9fd5ba88-6eaa-445e-b65e-1b8050e542a5",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "email"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1569242325.074
    }
}
```

Figure 1 AWS CLI Create Table

Navigating to the DynamoDB service in the AWS Management Console, will show the addition of the new Students table with the primary key of “email” as shown in figure 2. Note, previously created tables will also be displayed.



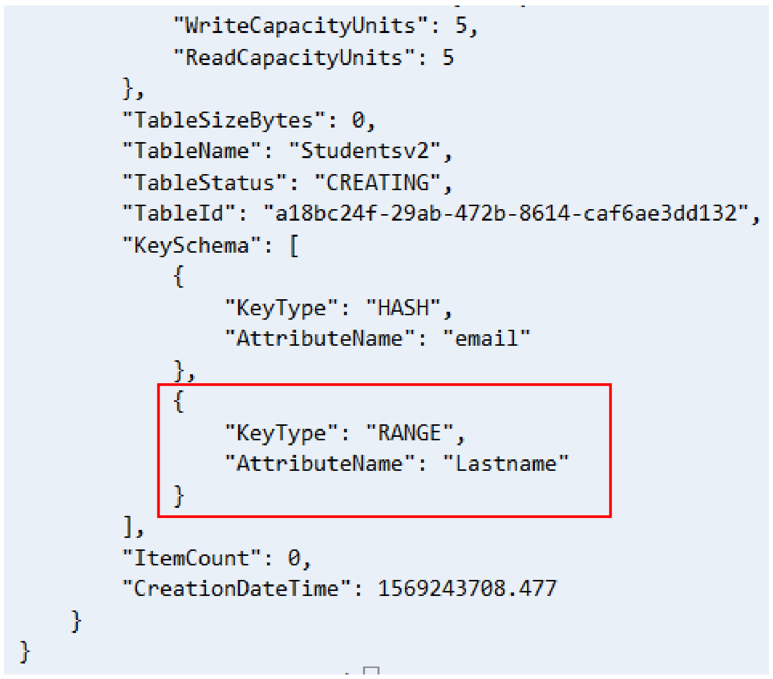
Name	Status	Partition key	Sort key	Indexes	Total read capacity	Total write capacity
FACULTY	Active	LASTNAME (String)	FIRSTNAME (String)	0	5	5
SOC	Active	STRM (Number)	CLASSNBR (Number)	0	5	5
Students	Active	email (String)	-	0	5	5

Figure 2 AWS Management Console Confirmation of Table Creation

If you wanted to create a table named Studentsv2 with a composite primary key consisting of email and Lastname, the following command would be entered.

```
aws dynamodb create-table --table-name Studentsv2 --attribute-definitions
AttributeName=Lastname,AttributeType=S AttributeName=email,AttributeType=S --
key-schema AttributeName=email,KeyType=HASH
AttributeName=Lastname,KeyType=RANGE --provisioned-throughput
ReadCapacityUnits=5,WriteCapacityUnits=5
```

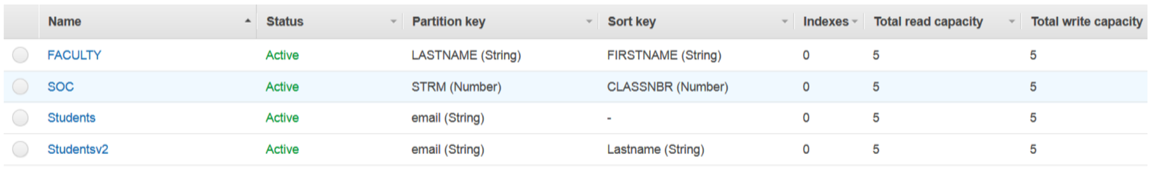
As shown in figure 3, the resulting script run from the Cloud9 environment shell will result in a similar output as before but will include the additional composite key attribute.



```
                "WriteCapacityUnits": 5,
                "ReadCapacityUnits": 5
            },
            "TableSizeBytes": 0,
            "TableName": "Studentsv2",
            "TableStatus": "CREATING",
            "TableId": "a18bc24f-29ab-472b-8614-caf6ae3dd132",
            "KeySchema": [
                {
                    "KeyType": "HASH",
                    "AttributeName": "email"
                },
                {
                    "KeyType": "RANGE",
                    "AttributeName": "Lastname"
                }
            ],
            "ItemCount": 0,
            "CreationDateTime": 1569243708.477
        }
}
```

Figure 3 JSON output snippet for Studentsv2

After logging into the AWS management console and selecting the DynamoDB service, the console will provide the expected tables along with their primary keys as shown in Figure 4.



Name	Status	Partition key	Sort key	Indexes	Total read capacity	Total write capacity
FACULTY	Active	LASTNAME (String)	FIRSTNAME (String)	0	5	5
SOC	Active	STRM (Number)	CLASSNBR (Number)	0	5	5
Students	Active	email (String)	-	0	5	5
Studentsv2	Active	email (String)	Lastname (String)	0	5	5

Figure 4 AWS Console Showing Created Tables

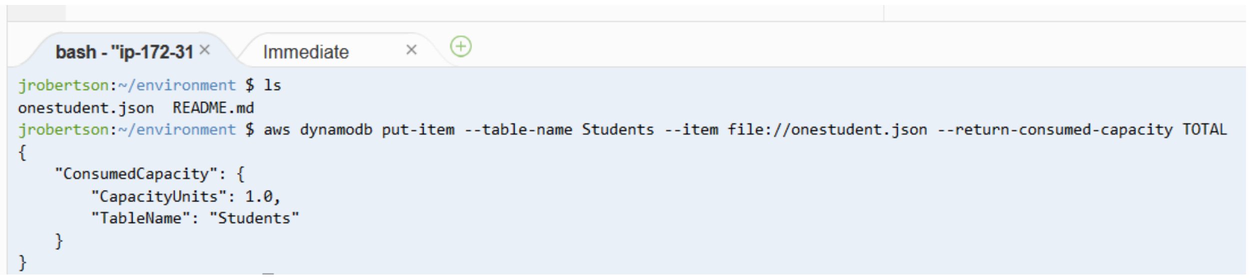
Put an Item in the Students Table:

The AWS CLI syntax to put an item in a DynamoDB table uses the `put-item` sub-command and a reference to a specific JSON file that includes the values for each attribute to be input into the table. The following command will insert one item into the Students DynamoDB table.

```
aws dynamodb put-item --table-name Students --item file://onestudent.json --
return-consumed-capacity TOTAL
```

The data is listed in the file named onestudent.json and includes the following information. Also, when invoking the command be sure the file exists in the same directory you currently reside. Notice in the screen capture below, the onestudent.json file is in the directory from where the AWS CLI command was called.

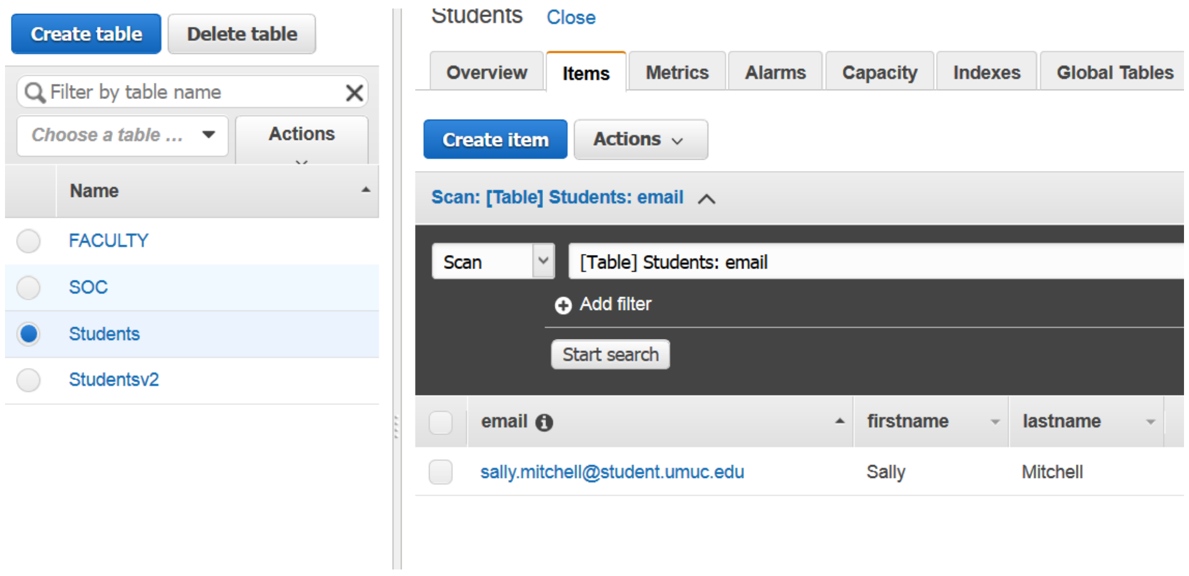
```
    {
        "email": {"S": "sally.mitchell@student.umuc.edu"},
        "firstname": {"S": "Sally"} ,
        "lastname": {"S": "Mitchell"}
    }
```



```
jrobertson:~/environment $ ls
onestudent.json  README.md
jrobertson:~/environment $ aws dynamodb put-item --table-name Students --item file://onestudent.json --return-consumed-capacity TOTAL
{
    "ConsumedCapacity": {
        "CapacityUnits": 1.0,
        "TableName": "Students"
    }
}
```

Figure 5 Putting an Item in the Students Table

Logging into the AWS Console and selecting the Students table and then the items tab, will reveal the newly entered item as shown in Figure 6.



email	firstname	lastname
sally.mitchell@student.umuc.edu	Sally	Mitchell

Figure 6 AWS Console View of New DynamoDB item

Get an Item from the Students table:

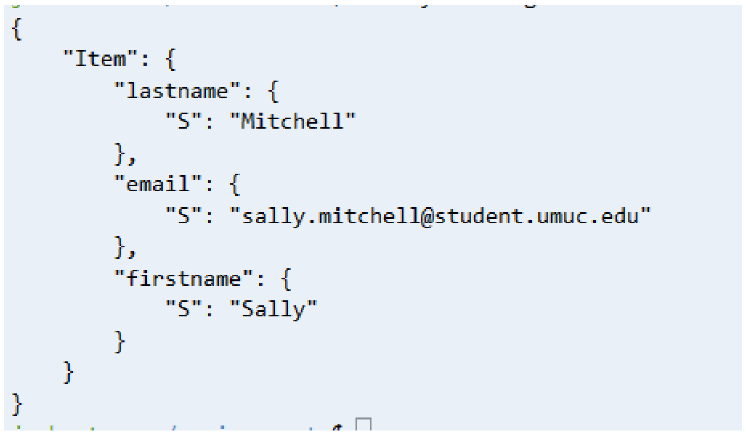
To retrieve an item using the AWS CLI command option a JSON file needs to be created that includes the specific keys you are searching. For example, to get the item from the Students table with the email key of sally.mitchell@student.umuc.edu , the following command would be entered:

```
aws dynamodb get-item --table-name Students --key file://keyItem.json
```

The keyItem.json file would include the search criteria and be saved in the same location where the command was launched.

```
    {
        "email": {"S": "sally.mitchell@student.umuc.edu"}
    }
```

Figure 7 shows the response from AWS for the get-item command.



```
{
    "Item": {
        "lastname": {
            "S": "Mitchell"
        },
        "email": {
            "S": "sally.mitchell@student.umuc.edu"
        },
        "firstname": {
            "S": "Sally"
        }
    }
}
```

Figure 7 Getting a Students Item

A couple of quick notes about the get-item command:

1. The text is case sensitive. Be sure you match your queries with the correct case for both the attributes (e.g. email) and the values (e.g. sally.mitchell@student.umuc.edu).
2. Make sure the JSON syntax is correct, uses braces and commas between keys, and is placed in the directory where your current command prompt resides. (i.e. if you call "ls" you should see that file.

The JSON syntax below would retrieve a composite key created for the Studentsv2 table.

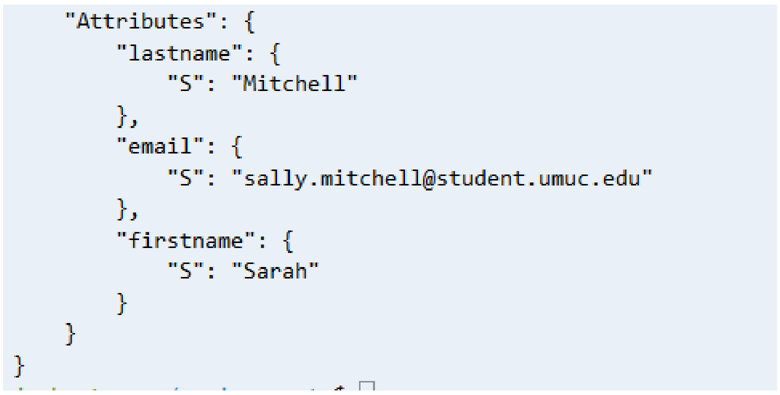
```
    {
        "email": {"S": "sally.mitchell@student.umuc.edu"},
        "Lastname": {"S": "Mitchell"}
    }
```

Update an item in the Students table:

To update an item in a table, you use the `update-item` sub-command. Similar to the put and get command, a JSON file is used for updating the attributes. To update an item in the Students table the following syntax would be used:

```
aws dynamodb update-item --table-name Students --key file://keyItem.json --
update-expression "SET #F = :f" --expression-attribute-names
file://ExpressionField.json --expression-attribute-values
file://ExpressionValue.json  --return-values ALL_NEW
```

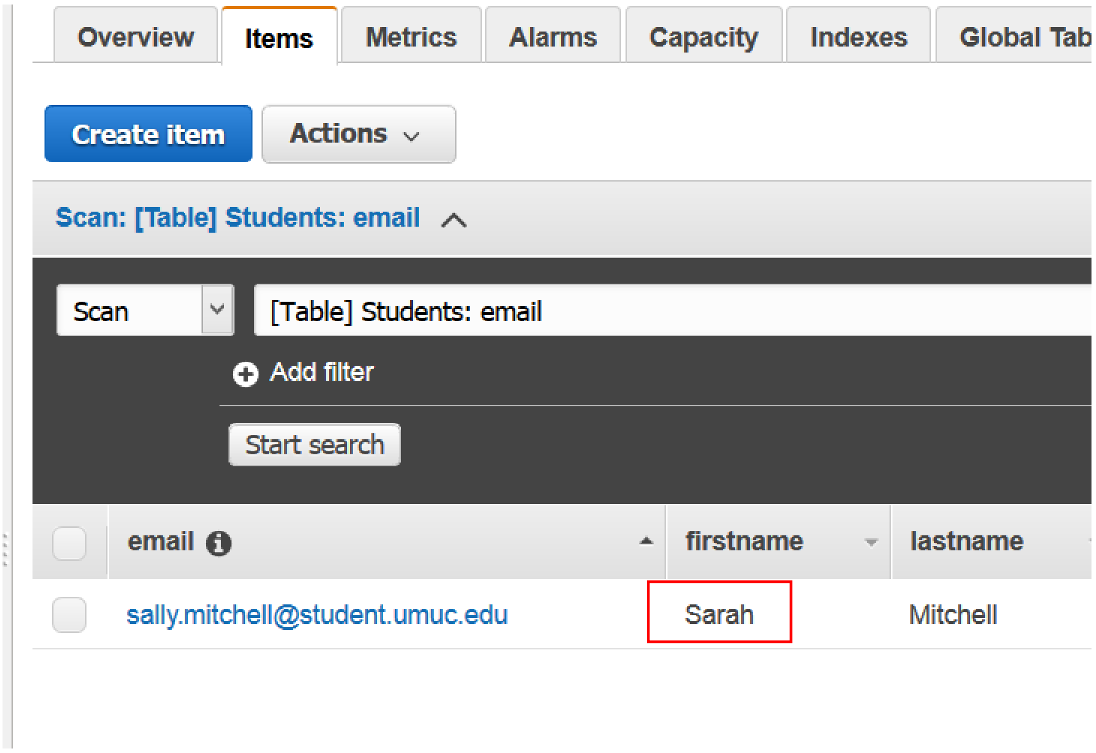
As shown in Figure 8, a response to a successful `update-item` sub-command will provide a JSON string with the updated item.



```
    "Attributes": {
        "lastname": {
            "S": "Mitchell"
        },
        "email": {
            "S": "sally.mitchell@student.umuc.edu"
        },
        "firstname": {
            "S": "Sarah"
        }
    }
}
```

Figure 8 Updating an Item in the Students Table

The results can also be verified by logging into the AWS management console, selecting the DynamoDB service and viewing the item associated with the Students table as shown in Figure 9.



email	firstname	lastname
sally.mitchell@student.umuc.edu	Sarah	Mitchell

Figure 9 Viewing the Updated Item in the AWS Management Console

Some additional information related to the command calls and the files used are needed here.

The key file is the same file we used before. It includes the following parameters:

```
{
    "email": {"S": "sally.mitchell@student.umuc.edu"}
}
```

The key file is used to point to a specific item to be updated.

The command that lists `update-expression "SET #F = :f"` is used in conjunction with the Expression and values JSON files. These are just reference values to ask a specific attribute to be updated with a specific value. The attribute is named in the ExpressionField.json file:

```
{
    "#F":"firstname"
}
```

The value is named in this ExpressionValue.json file:

```
{
    ":f":{"S": "Jimmy"}

}
```

Note, the #F references the firstname attribute and :f lists the value and the type to be update for this attribute. As long as the attribute already exists, you can update that attribute as well. For example, if a joinyear attribute was available for that item, you could update that field with this syntax:

```
aws dynamodb update-item --table-name Students --key file://keyItem.json --
update-expression "SET #F = :f, #H = :h" --expression-attribute-names
file://ExpressionField.json --expression-attribute-values
file://Expressionvalues.json  --return-values ALL_NEW
```

Notice the set command includes an additional reference of `#H = :h` separated by a comma. Additional references may also be added, each separated by a comma as needed. Then the corresponding ExpressionField and Value files would need to include the new information:

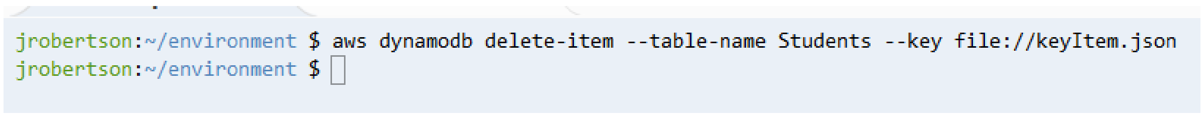
```
{
    "#F":"firstname",
    "#H":"joinyear"
}


{
    ":f":{"S": "Sally"},
    ":h":{"S": "2018"},

}
```

Delete an Item from the Student table:

Finally, to delete an item from the table, you use the JSON key file along with the `delete-item` sub-command. For example, to delete the same item we previously entered from the Students table, the following syntax would be used:

```
aws dynamodb delete-item --table-name Students --key file://keyItem.json
```

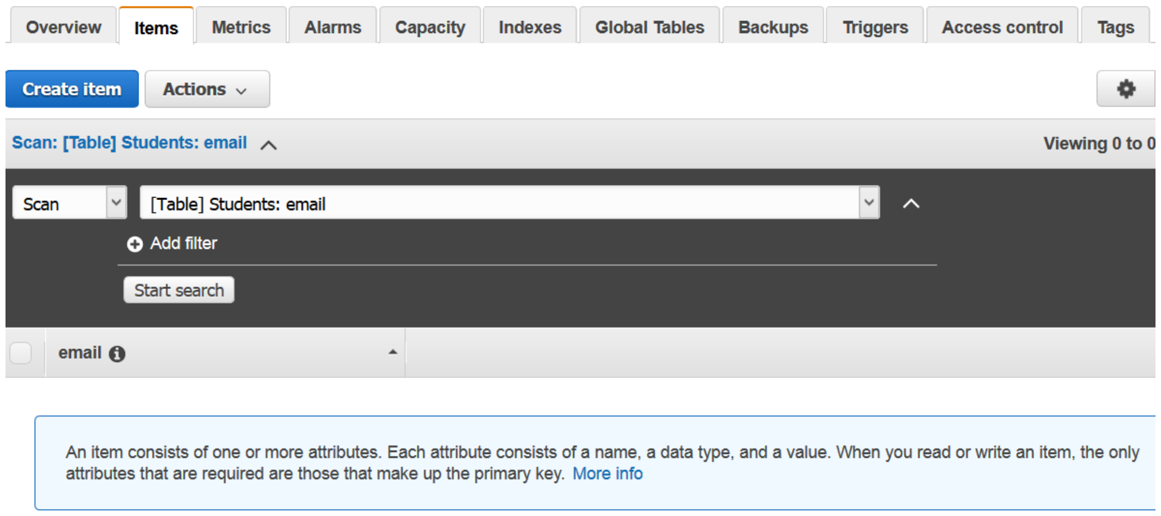
Figure 10 shows the response upon successful deletion of the item.



```
jrobertson:~/environment $ aws dynamodb delete-item --table-name Students --key file://keyItem.json
jrobertson:~/environment $
```

Figure 10 Deleting an Item from the Students Table

Note there is no response from AWS upon successful deletion of the item. To verify the item is no longer present, use the AWS console to display the items in the table. As shown in Figure 11, no items currently exist in the Students table.



An item consists of one or more attributes. Each attribute consists of a name, a data type, and a value. When you read or write an item, the only attributes that are required are those that make up the primary key. More info

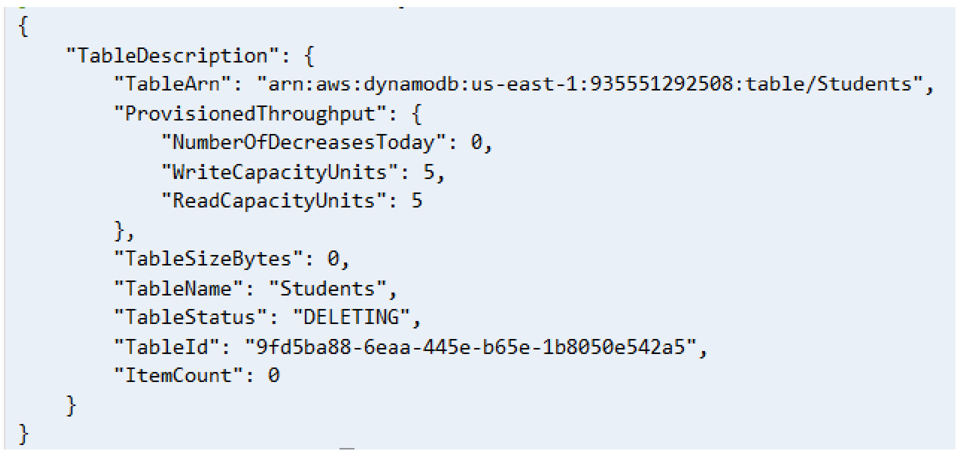
Figure 11 Confirming the Item was Deleted using the AWS Management Console

Deleting the Students Table:

The AWS CLI `delete-table` sub-command can be used to delete the table and all of the content as shown below:

```
aws dynamodb delete-table --table-name Students
```

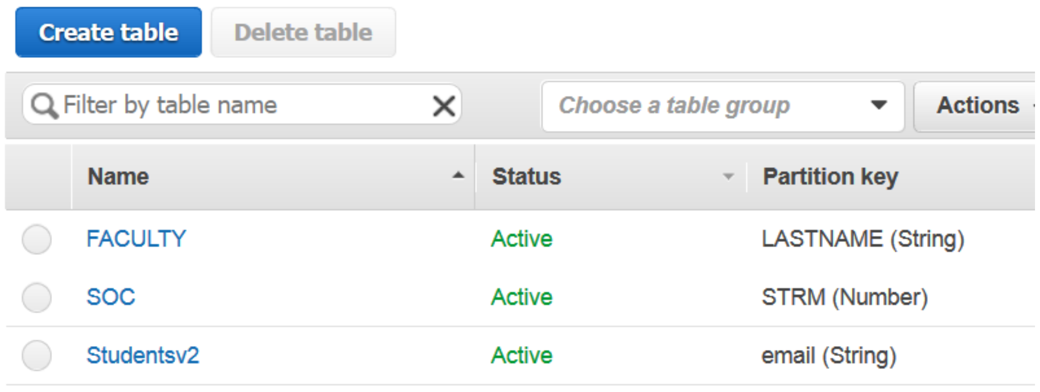
A response from AWS will be received after issuing this command as shown in Figure 12:



```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:935551292508:table/Students",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "Students",
        "TableStatus": "DELETING",
        "TableId": "9fd5ba88-6eaa-445e-b65e-1b8050e542a5",
        "ItemCount": 0
    }
}
```

Figure 12 Deleting the Students table

The AWS management console may then be used to confirm the Students table was deleted as shown in Figure 13.



	Name	Status	Partition key
	FACULTY	Active	LASTNAME (String)
	SOC	Active	STRM (Number)
	Studentsv2	Active	email (String)

Figure 13 Confirming the Students Table was Deleted

The AWS DynamoDB service provides additional functions accessible through AWS CLI such as describing the tables, batch-write-item and batch-get-item. The homework for this week will give you the opportunity to experiment with multiple commands. You will need to use the AWS CLI DynamoDB API documentation to guide you through the completion of the exercises.

The API can be found at this URL:

https://docs.aws.amazon.com/cli/latest/reference/dynamodb/index.html#cli-aws-dynamodb

AWS-SDK with DynamoDB:

Similar to the S3 boto3 examples in Cloud9, we can interact and perform powerful programming functions in AWS for DynamoDB. To use the examples provided for this week, you should already have the boto3 environment installed in your Cloud9 environment. Review the UsingS3 document, found in the Week 2 content, in case you need to create a new Cloud9 environment.

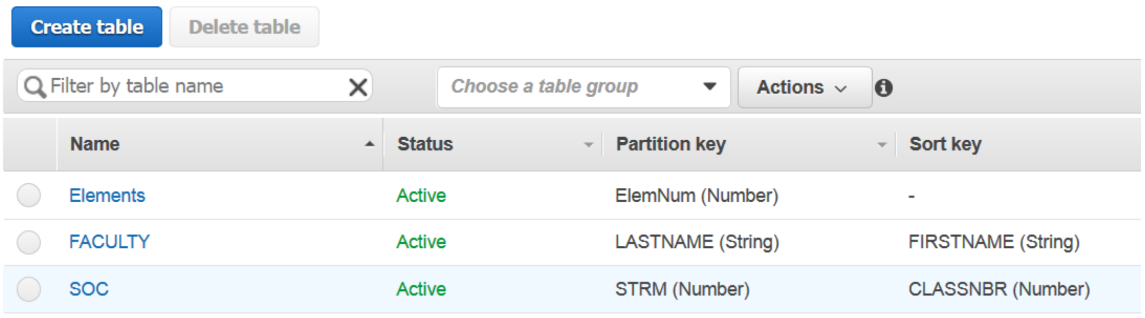
AWS provides a number of Python examples for interacting with DynamoDB. Functionality including creating tables, scanning tables, querying tables, and uploading data are included. These examples are provided as is from the AWS Web site. You may need to make some edits to run them in this region. For example, in most cases the default Region and endpoint are used. So instead of specifically listing them in the resource call, just use the service name:

```
dynamodb = boto3.resource('dynamodb')
```

The following code will create a table named Elements and return the status:

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.create_table(
    TableName='Elements',
    KeySchema=[
        {
            'AttributeName': 'ElemNum',
            'KeyType': 'HASH'
        }
    ],
    AttributeDefinitions=[
        {
            'AttributeName': 'ElemNum',
            'AttributeType': 'N'
        },


    ],
    ProvisionedThroughput={
        'ReadCapacityUnits': 5,
        'WriteCapacityUnits': 5
    }
)
print("Table status:", table.table_status)
```

As shown in figure 14, the Elements table is created in the DynamoDB service.



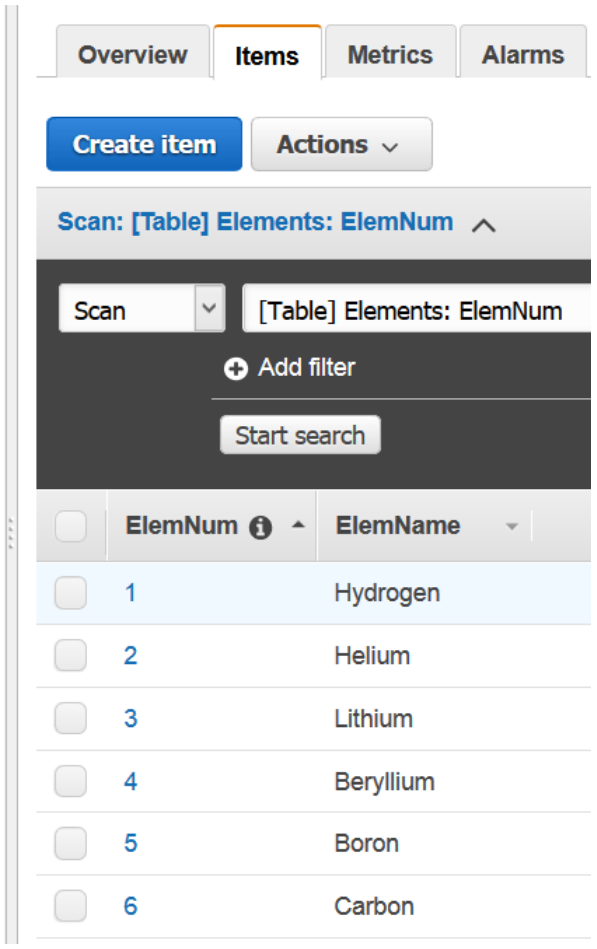
Name	Status	Partition key	Sort key
Elements	Active	ElemNum (Number)	-
FACULTY	Active	LASTNAME (String)	FIRSTNAME (String)
SOC	Active	STRM (Number)	CLASSNBR (Number)

Figure 14 Creating the Elements Table

To add data from an array, the following Python code could be used:

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('Elements')
elements = [[1,'Hydrogen'],
            [2,'Helium'],
            [3,'Lithium'],
            [4,'Beryllium'],
            [5,'Boron'],
            [6,'Carbon']]

for element in elements:
   num = element[0]
   name = element[1]

   table.put_item(
      Item={
          'ElemNum': num,
           'ElemName': name
      }
   )
```

After successful running of the code, the Elements table is populated with 6 items as shown in figure 15.



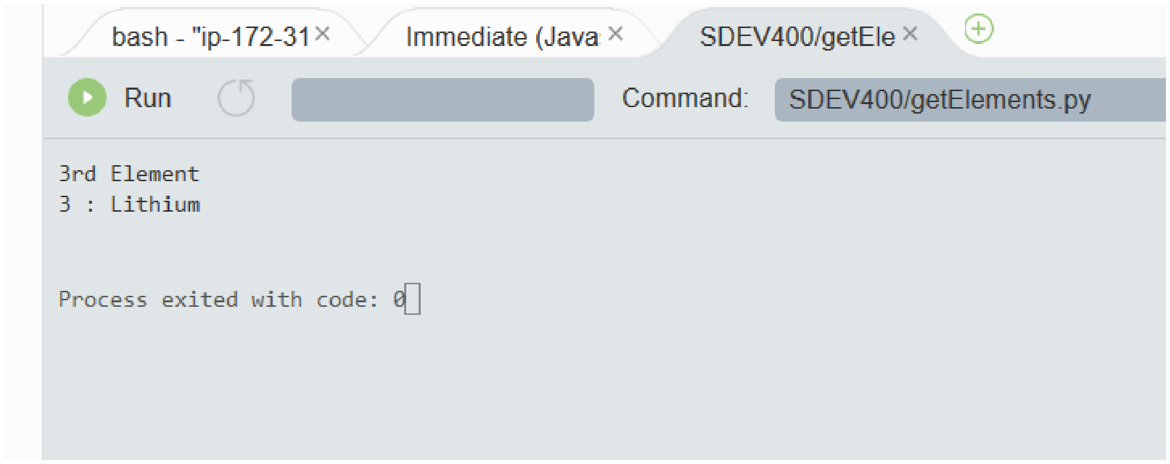
ElemNum	ElemName
1	Hydrogen
2	Helium
3	Lithium
4	Beryllium
5	Boron
6	Carbon

Figure 15 Inserting 6 Elements

To retrieve the elements with third element, the following code could be used:

```
import boto3
from boto3.dynamodb.conditions import Key, Attr

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('Elements')

print("3rd Element")

response = table.query(
    KeyConditionExpression=Key('ElemNum').eq(3)
)

for i in response['Items']:
    print(i['ElemNum'], ":", i['ElemName'])
```

Figure 16 shows the results of running this Python code from the Cloud9 environment.



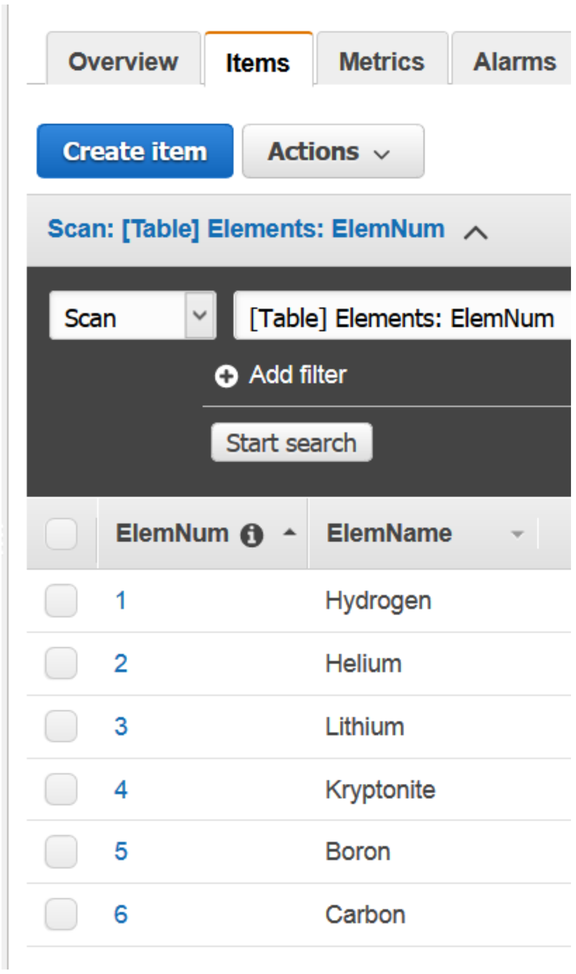
```
3rd Element
3 : Lithium


Process exited with code: 0
```

Figure 16 Getting the 3rd Element

To update an item, you can use set the key and the expression with the new value as shown in the following code:

```
import boto3
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('Elements')
response = table.update_item(
     Key={
       'ElemNum':4
     },
     UpdateExpression='SET ElemName = :values',
        ExpressionAttributeValues={
          ':values': 'Kryptonite'
        }
     )
```

Even though we really wouldn't want to update the fourth element in the periodic table to Kryptonite permanently, we have accomplished a temporary element name change. (See figure 17).



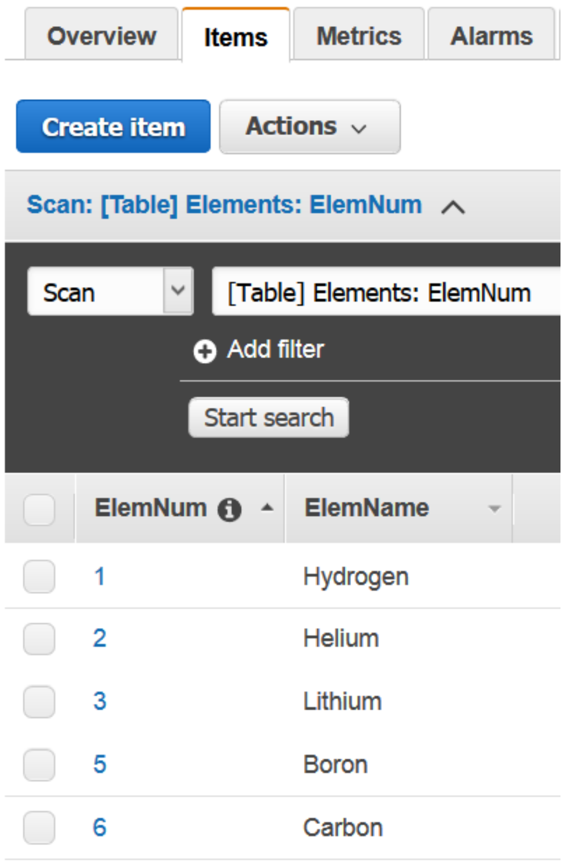
ElemNum	ElemName
1	Hydrogen
2	Helium
3	Lithium
4	Kryptonite
5	Boron
6	Carbon

Figure 17 Updating an Element Name

To delete an item, the following could be used:

```
import boto3
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('Elements')
response = table.delete_item(
      Key={
        'ElemNum':4
      }
)
print(response)
```

After running this code and returning to the DynamoDB AWS Management Console view, the element number of 4 has been removed. See figure 18.



ElemNum	ElemName
1	Hydrogen
2	Helium
3	Lithium
5	Boron
6	Carbon

Figure 18 Deleting an Item

Finally, you can use the AWS-SDK to delete the Table and all of the contents using code similar to the following:

```
import boto3

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('Elements')
table.delete()
```

You should experiment with these code samples as well as the one provided in this week's content area to help you form the code needed to complete the homework for this week.