Amazon DynamoDB CRUD Activities Using the AWS CLI and SDK

**SPL-TF-200-DBDYL1-10-EN - Version 1.0.4**

© 2024 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. All trademarks are the property of their owners.

Note: Do not include any personal, identifying, or confidential information into the lab environment. Information entered may be visible to others.

Corrections, feedback, or other questions? Contact us at [*AWS Training and Certification*](https://support.aws.amazon.com/#/contacts/aws-training).

**Lab overview**

AnyCompany is developing a new movie database application. The application will allow users to search for movies by title, year they were released, actors in the movie, rating, and so on. AnyCompany chose Amazon DynamoDB as the database service to store the information about all of the movies that the application will reference.

Amazon DynamoDB is a serverless NoSQL, also known as nonrelational, database that supports key-value and document data models. With DynamoDB, you can start small and quickly scale globally as your application and user base grows, with the ability to support petabytes of data and tens of millions of read and write requests per second. AWS manages the DynamoDB service, so there are no servers to update or maintain. For more information, refer to *Amazon DynamoDB Features* and *Core Components of Amazon DynamoDB* in the **Additional resources** section.

As a newly hired senior application developer at AnyCompany, you have been asked to make yourself familiar with DynamoDB and the various ways to manage tables and data stored in the service. In the future, you will support the new movie database application, which is built using Python.

TOPICS COVERED

By the end of this lab, you will be able to:

* Create and manage a DynamoDB table using the AWS Command Line Interface (AWS CLI).
* Create and manage an Amazon DynamoDB table using the AWS SDK for Python.

TECHNICAL KNOWLEDGE PREREQUISITES

To successfully complete this lab, you should be familiar with basic navigation of the AWS Management Console, NoSQL database concepts, the Python programming language, and be comfortable editing scripts using a text editor.

ICON KEY

Various icons are used throughout this lab to call attention to certain aspects of the guide. The following list explains the purpose for each one:

* The keyboard icon specifies that you must run a command.
* The clipboard icon indicates that you can verify the output of a command or edited file by comparing it to the provided example.
* The note icon specifies important hints, tips, guidance, or advice.
* The “i” circle icon specifies where to find more information.
* The person with a check mark icon indicates an opportunity to check your knowledge and test what you have learned.

**Start lab**

1. To launch the lab, at the top of the page, choose **Start lab**.

**Caution:** You must wait for the provisioned AWS services to be ready before you can continue.

1. To open the lab, choose **Open Console**.

You are automatically signed in to the AWS Management Console in a new web browser tab.

**WARNING:** **Do not change the Region unless instructed.**

COMMON SIGN-IN ERRORS

**Error: You must first sign out**



If you see the message, **You must first log out before logging into a different AWS account:**

* Choose the **click here** link.
* Close your **Amazon Web Services Sign In** web browser tab and return to your initial lab page.
* Choose **Open Console** again.

**Error: Choosing Start Lab has no effect**

In some cases, certain pop-up or script blocker web browser extensions might prevent the **Start Lab** button from working as intended. If you experience an issue starting the lab:

* Add the lab domain name to your pop-up or script blocker’s allow list or turn it off.
* Refresh the page and try again.

**Task 1: Use the AWS CLI to create and manage a DynamoDB table**

The AWS CLI is the first method of interacting with DynamoDB that you have decided to explore.

In this task, you connect to the AWS Cloud9 integrated development environment (IDE) and verify it has the tools necessary to interact with DynamoDB. You then use AWS CLI to recreate the table and items from the previous task to compare the two methods.

TASK 1.1: CONNECT TO THE AWS CLOUD9 IDE

1. At the top of the AWS Management Console, in the search bar, search for and choose

Cloud9

.

1. On the **Environments** page, choose the **Open** link for the **Lab\_Cloud9\_IDE** environment.

The AWS Cloud9 environment opens in a new browser tab.

1. To the right of the **Welcome** tab, choose the plus  icon, and then select **New Terminal**.

 As of the writing of this lab, the default version of the AWS CLI installed on the AWS Cloud9 instance is 2.15.

1. To verify AWS CLI active version, run the following command:

aws --version

**Expected Output** The output should show aws-cli 2.x.x, similar to the following:

\*\*\* EXPECTED OUTPUT \*\*\*

aws-cli/2.15.23 Python/3.11.6 Linux/5.10.209-198.858.amzn2.x86\_64 exe/x86\_64.amzn.2 prompt/off

TASK 1.2: USE THE AWS CLI TO CREATE A DYNAMODB TABLE

Now that you have verified that the latest version of the AWS CLI is installed, you can start creating the DynamoDB table.

1. To create a DynamoDB table named **movies**, run the following command:

aws dynamodb create-table \

--table-name movies \

--attribute-definitions \

AttributeName=year,AttributeType=N \

AttributeName=title,AttributeType=S \

--key-schema \

AttributeName=year,KeyType=HASH \

AttributeName=title,KeyType=RANGE \

--billing-mode PROVISIONED \

--provisioned-throughput \

ReadCapacityUnits=10,WriteCapacityUnits=10

The following list describes each part of the command that you entered:

* aws dynamodb create-table

 is the base command to create a new table.

* --table-name

 specifies the name of the table.

* --attribute-definitions

 specifies which initial attributes to create. You must include at least the attributes that you plan to use as the primary key for the table.

* + AttributeName=year,AttributeType=N

 specifies a number type attribute named **year**.

* + AttributeName=title,AttributeType=S

 specifies a string type attribute named **title**.

* --key-schema

 specifies the attributes used for the primary key.

* + AttributeName=year,KeyType=HASH

 specifies the attribute to use as the partition (HASH) key.

* + AttributeName=title,KeyType=RANGE

 specifies the attribute to use as the sort (RANGE) key.

* --billing-mode

 specifies how you are charged for read and write throughput. In this example, you use **PROVISIONED** to set Provisioned Mode because the workload is predictable. For workloads that may be unpredictable, you can use PAY\_PER\_REQUEST to set On-Demand Mode and pay for each read and write. For more information, refer to *Read/Write Capacity Mode* in the **Additional resources** section.

* --provisioned-throughput

 specifies the provisioned throughput settings for the table. In this example, you set a low number of read and write capacity units because you will not generate a large number of requests to the table throughout the course of this lab. For more information, refer to *Provisioned Mode* in the **Additional resources** section.

 For more information, refer to *create-table CLI Command* in the **Additional resources** section.

**Expected Output** The output should display the details of the table in JSON format, similar to the following:

\*\*\* EXPECTED OUTPUT \*\*\*

{

"TableDescription": {

"AttributeDefinitions": [

{

"AttributeName": "title",

"AttributeType": "S"

},

{

"AttributeName": "year",

"AttributeType": "N"

}

],

"TableName": "movies",

"KeySchema": [

{

"AttributeName": "year",

"KeyType": "HASH"

},

{

"AttributeName": "title",

"KeyType": "RANGE"

}

],

"TableStatus": "CREATING",

"CreationDateTime": "2020-11-21T01:53:20.080000+00:00",

"ProvisionedThroughput": {

"NumberOfDecreasesToday": 0,

"ReadCapacityUnits": 10,

"WriteCapacityUnits": 10

},

"TableSizeBytes": 0,

"ItemCount": 0,

"TableArn": "arn:aws:dynamodb:us-west-2:012345678901:table/movies",

"TableId": "03bcd8c5-7d31-4c45-ba41-11e9e4487c00"

}

}

The table must be **active** before you can add items to it.

1. To check the status of the table, run the following command:

aws dynamodb describe-table \

--table-name movies \

--query "Table.TableStatus"

If the table status is not **ACTIVE**, wait one minute and then rerun the previous command.

TASK 1.3: USE THE AWS CLI TO ADD ITEMS TO A DYNAMODB TABLE

1. To create a new item in the **movies** table with details about the movie *The Godfather*, run the following command:

aws dynamodb put-item \

--table-name movies \

--item '{

"year": {"N": "1972"},

"title": {"S": "The Godfather"},

"actors": {"SS": ["Marlon Brando", "Al Pacino", "James Caan"]}

}'

 For more information, refer to *put-item CLI Command* in the **Additional resources** section.

 Use a text editor to easily adjust the command for the below two additional entries.

1. Repeat the previous step to create two more items with the following information:

* Item one:
  + year:

1972

* + title:

Deliverance

* + actors:

Jon Voight

,

Burt Reynolds

,

Ned Beatty

* Item two:
  + year:

1994

* + title:

The Shawshank Redemption

* + actors:

Tim Robbins

,

Morgan Freeman

,

Bob Gunton

TASK 1.4: USE THE AWS CLI TO FETCH ITEMS WITH THE SCAN OPERATION AND FILTERS

Now that the table has a few items, you can use various commands to fetch data from it, such as get-item, batch-get-item, query, and scan. In this lab, you use scan and query.

In this task, you run a Scan operation to find specific movies.

 The *Scan* operation accesses every item in the table and returns results based on the filters that you select. If you do not specify a filter, all items in the table are returned. For more information, refer to *Working with Scans in DynamoDB* in the **Additional resources** section.

First, perform a scan of the **movies** table without a filter to discover that it returns all items.

1. To scan the entire table, which returns a list of all of the movies in the table, run the following command:

aws dynamodb scan \

--table-name movies

 For more information, refer to *scan CLI Command* in the **Additional resources** section.

 You may use the down-arrow key to scroll to the end of the output and pressing “q” to exit back to the shell.

**Expected Output** The output should display the contents of the table, similar to the following:

\*\*\*EXPECTED OUTPUT\*\*\*

{

"Items": [

{

"year": {

"N": "1994"

},

"actors": {

"SS": [

"Bob Gunton",

"Morgan Freeman",

"Tim Robbins"

]

},

"title": {

"S": "The Shawshank Redemption"

}

},

{

"year": {

"N": "1972"

},

"actors": {

"SS": [

"Burt Reynolds",

"Jon Voight",

"Ned Beatty"

]

},

"title": {

"S": "Deliverance"

}

},

{

"year": {

"N": "1972"

},

"actors": {

"SS": [

"Al Pacino",

"James Caan",

"Marlon Brando"

]

},

"title": {

"S": "The Godfather"

}

}

],

"Count": 3,

"ScannedCount": 3,

"ConsumedCapacity": null

}

Next, apply a filter to the scan results.

1. To scan the entire table, but only return items with a **title** attribute of **The Godfather**, run the following command:

aws dynamodb scan \

--table-name movies \

--filter-expression "title = :name" \

--expression-attribute-values '{":name": {"S": "The Godfather"}}'

**Expected Output** The output should display the details of the movie *The Godfather*, similar to the following:

\*\*\* EXPECTED OUTPUT \*\*\*

{

"Items": [

{

"year": {

"N": "1972"

},

"actors": {

"SS": [

"Al Pacino",

"James Caan",

"Marlon Brando"

]

},

"title": {

"S": "The Godfather"

}

}

],

"Count": 1,

"ScannedCount": 3,

"ConsumedCapacity": null

}

TASK 1.5: USE THE AWS CLI TO FETCH ITEMS WITH THE QUERY OPERATION

Next, use the Query operation to search for all movies released in 1972.

 The *Query* operation fetches items based on the primary key. You must provide at least the value for the partition key when performing a query. For more information, refer to *Working with Queries in DynamoDB* in the **Additional resources** section.

1. To query the table for all movies released in 1972, run the following command:

aws dynamodb query \

--table-name movies \

--projection-expression "title" \

--key-condition-expression "#y = :yr" \

--expression-attribute-names '{"#y":"year"}' \

--expression-attribute-values '{":yr":{"N":"1972"}}'

The following list describes each part of the command that you entered:

* aws dynamodb query

 is the base command to query a table.

* --table-name

 specifies the name of the table.

* --projection-expression

 specifies which attributes to display in the output of the command. In this example, the output only displays the title of the movies that match the query.

* --key-condition-expression

 specifies the partition key attribute to retrieve.

* --expression-attribute-names

 specifies a value to use in place of the attribute name. The word **year** is a reserved word in DynamoDB, which means you cannot use it in any expressions. In this example, you used **#y** as the expression attribute name in place of **year**. If your table contains attribute names that are on the list of reserved words, you can use an expression attribute name in its place. For more information, refer to *Reserved Words in DynamoDB* and *Expression Attribute Names in DynamoDB* in the **Additional resources** section.

* --expression-attribute-values

 specifies the value of the attribute in the key condition expression to query for. In this example, you search for any item with a **year** value of **1972**.

 For more information, refer to *query CLI Command* in the **Additional resources** section.

**Expected Output** If you added all three items to the table, the output should display the details of two movies that were released in 1972, *The Godfather* and *Deliverance*, similar to the following:

\*\*\* EXPECTED OUTPUT \*\*\*

{

"Items": [

{

"title": {

"S": "Deliverance"

}

},

{

"title": {

"S": "The Godfather"

}

}

],

"Count": 2,

"ScannedCount": 2,

"ConsumedCapacity": null

}

TASK 1.6: USE THE AWS CLI TO UPDATE AN ITEM

Now that you know how to add and search for items, add an attribute to track the rating of each movie.

1. To update *The Godfather* with a new attribute named **rating**, run the following command:

aws dynamodb update-item \

--table-name movies \

--key '{

"year": {"N": "1972"},

"title": {"S" :"The Godfather"}

}' \

--update-expression 'SET #rating = :rate' \

--expression-attribute-names '{

"#rating": "rating"

}' \

--expression-attribute-values '{":rate": {"N": "9.2"}}'

 For more information, refer to *update-item CLI Command* in the **Additional resources** section.

 Challenge yourself! Repeat the previous step to add ratings to the following movies:

| **Movie title** | **Rating** |
| --- | --- |
| Deliverance | 7.8 |
| The Shawshank Redemption | 9.3 |

1. To query the table for *The Godfather*, run the following command:

aws dynamodb get-item --table-name movies --key '{"year": {"N": "1972"}, "title": {"S": "The Godfather"}}'

 Notice that the returned entry now contains a **rating** attribute.

TASK 1.7: USE THE AWS CLI TO DELETE AN ITEM

Next, delete a single item from the table.

1. To delete *The Godfather*, run the following command:

aws dynamodb delete-item \

--table-name movies \

--key '{

"year": {"N": "1972"},

"title": {"S" :"The Godfather"}

}'

 For more information, refer to *delete-item CLI Command* in the **Additional resources** section.

1. To perform a Scan of the movies table and verify *The Godfather* has been deleted, run the following command:

aws dynamodb scan \

--table-name movies

TASK 1.8: USE THE AWS CLI TO DELETE A TABLE

In the next task, you use the AWS SDK for Python to create the same table. Therefore, you can delete the one that you just created.

1. To delete the entire movies table, run the following command:

aws dynamodb delete-table \

--table-name movies

**Expected Output** The output should be the details of the table being removed, similar to this:

\*\*\* EXPECTED OUTPUT \*\*\*

{

"TableDescription": {

"TableName": "movies",

"TableStatus": "DELETING",

"ProvisionedThroughput": {

"NumberOfDecreasesToday": 0,

"ReadCapacityUnits": 10,

"WriteCapacityUnits": 10

},

"TableSizeBytes": 0,

"ItemCount": 0,

"TableArn": "arn:aws:dynamodb:us-east-2:996290967850:table/movies",

"TableId": "89ae15bb-ee79-437c-b54b-74c15b6d6402"

}

}

 For more information, refer to *delete-table CLI Command* in the **Additional resources** section.

1. To list all DynamoDB tables and verify that the **movies** table was deleted, run the following command:

aws dynamodb list-tables

**Expected Output** The output should be an empty list of tables, similar to this:

\*\*\* EXPECTED OUTPUT \*\*\*

{

"TableNames": []

}

 Congratulations! You have successfully created a DynamoDB table to store information about movies, created and updated items, performed queries to look up specific movies, deleted items, and deleted a table using the AWS CLI.

**Task 2: Use the AWS SDK for Python to create and manage a DynamoDB table**

Now that you are more comfortable interacting with DynamoDB from the AWS CLI, you’re ready to start creating Python scripts to interact with DynamoDB programmatically.

TASK 2.1: INSTALL THE AWS SDK FOR PYTHON (BOTO3)

In this task, you install the AWS SDK for Python (Boto3). You then use Python scripts to recreate the table and items from the previous task to compare both methods.

1. In the AWS Cloud9 terminal, to install Boto3, the AWS SDK for Python, run the following command:

sudo python3 -m pip install boto3

 For more information, refer to *AWS SDK for Python (Boto3)* in the **Additional resources** section.

TASK 2.2: USE THE AWS SDK TO CREATE A DYNAMODB TABLE

Now you are ready to create a table.

1. In the AWS Cloud9 terminal, on the **File** menu, choose **New File**.
2. In the **Untitled1** editor tab, copy and paste the following Python script, which is used to create a new DynamoDB table:

import boto3

def create\_movie\_table():

region=boto3.session.Session().region\_name

dynamodb = boto3.resource('dynamodb', region\_name=region) # low-level client

table = dynamodb.create\_table(

TableName='movies',

KeySchema=[

{

'AttributeName': 'year',

'KeyType': 'HASH' # Partition key

},

{

'AttributeName': 'title',

'KeyType': 'RANGE' # Sort key

}

],

AttributeDefinitions=[

{

'AttributeName': 'year',

'AttributeType': 'N'

},

{

'AttributeName': 'title',

'AttributeType': 'S'

},

],

ProvisionedThroughput={

'ReadCapacityUnits': 10,

'WriteCapacityUnits': 10

},

BillingMode='PROVISIONED'

)

return table

if \_\_name\_\_ == '\_\_main\_\_':

movie\_table = create\_movie\_table()

print("Table status:", movie\_table.table\_status)

 For more information, refer to *CreateTable Operation* in the **Additional resources** section.

1. On the **File** menu, choose **Save As…** and save the file as

MoviesCreateTable.py

1. Close the **MoviesCreateTable.py** tab.
2. To run the **MoviesCreateTable.py** script and create the **movies** table, run the following command:

python MoviesCreateTable.py

The output should show the current state of the table:

Table status: CREATING

1. To list all of the DynamoDB tables and verify that the **movies** table was created, run the following command:

aws dynamodb describe-table --table-name movies

**Expected Output** The output should show the details of the **movies** table, including the current status, similar to the following:

\*\*\* EXPECTED OUTPUT \*\*\*

{

"Table": {

"AttributeDefinitions": [

{

"AttributeName": "title",

"AttributeType": "S"

},

{

"AttributeName": "year",

"AttributeType": "N"

}

],

"TableName": "movies",

"KeySchema": [

{

"AttributeName": "year",

"KeyType": "HASH"

},

{

"AttributeName": "title",

"KeyType": "RANGE"

}

],

"TableStatus": "ACTIVE",

"CreationDateTime": "2020-11-17T15:50:02.642000+00:00",

"ProvisionedThroughput": {

"NumberOfDecreasesToday": 0,

"ReadCapacityUnits": 10,

"WriteCapacityUnits": 10

},

"TableSizeBytes": 0,

"ItemCount": 0,

"TableArn": "arn:aws:dynamodb:us-west-2:012345678901:table/movies",

"TableId": "d68bb72c-8937-4ddb-a4a9-3e76f6409109"

}

}

1. Verify that the **movies** table was created successfully and that the **TableStatus** value is **Active**.

TASK 2.3: USE THE AWS SDK TO ADD ITEMS TO A DYNAMODB TABLE

Now that you’ve created the table, you can add items to it.

1. On the **File** menu, choose **New File**.
2. In the **Untitled1** editor tab, copy and paste the following Python script, which is used to add an item to the **movies** table:

from pprint import pprint

import boto3

def put\_movie(title, year, actors):

region=boto3.session.Session().region\_name

dynamodb = boto3.resource('dynamodb', region\_name=region) # low-level client

table = dynamodb.Table('movies')

response = table.put\_item(

Item={

'year': year,

'title': title,

'info': {

'actors': actors

}

}

)

return response

if \_\_name\_\_ == '\_\_main\_\_':

movie\_title = "The Godfather"

movie\_year = 1972

movie\_actors = {"Marlon Brando", "Al Pacino", "James Caan"}

movie\_resp = put\_movie(movie\_title, movie\_year, movie\_actors)

print("Put movie succeeded:")

pprint(movie\_resp)

 For more information, refer to *PutItem Operation* in the **Additional resources** section.

1. On the **File** menu, choose **Save As…** and save the file as

MoviesPutItem.py

1. Close the **MoviesPutItem.py** tab.
2. To run the **MoviesPutItem.py** script to add an item for the movie *The Godfather* as defined in the file, run the following command:

python MoviesPutItem.py

 The output should indicate that the put operation was successful, similar to the following:

\*\*\* EXPECTED OUTPUT \*\*\*

Put movie succeeded:

{'ResponseMetadata': {'HTTPHeaders': {'connection': 'keep-alive',

'content-length': '2',

'content-type': 'application/x-amz-json-1.0',

'date': 'Thu, 05 Nov 2020 20:08:16 GMT',

'server': 'Server',

'x-amz-crc32': '2745614147',

'x-amzn-requestid': 'O3CPHI1T4V2C3MK7QPBUN8L9CVVV4KQNSO5AEMVJF66Q9ASUAAJG'},

'HTTPStatusCode': 200,

'RequestId': 'O3CPHI1T4V2C3MK7QPBUN8L9CVVV4KQNSO5AEMVJF66Q9ASUAAJG',

'RetryAttempts': 0}}

 Challenge yourself! Modify the **MoviesPutItem.py** script to add two more items to the table:

**Hint:** Use the current structure of the script. Modify it, and run it twice, one time for each new addition.

* Item one:
  + year:

1972

* + title:

Deliverance

* + actors:

Jon Voight

,

Burt Reynolds

,

Ned Beatty

* Item two:
  + year:

1994

* + title:

The Shawshank Redemption

* + actors:

Tim Robbins

,

Morgan Freeman

,

Bob Gunton

TASK 2.4: USE THE AWS SDK TO FETCH ITEMS WITH THE SCAN OPERATION AND FILTERS

Next, use the Scan operation with filters to locate movies within a specified time frame.

1. On the **File** menu, choose **New File**.
2. In the **Untitled1** editor tab, copy and paste the following Python script, which scans the table for all movies released between two specified years:

from pprint import pprint

import boto3

from boto3.dynamodb.conditions import Key

import argparse

def scan\_movies(year\_range, display\_movies):

region=boto3.session.Session().region\_name

dynamodb = boto3.resource('dynamodb', region\_name=region) # low-level client

table = dynamodb.Table('movies')

scan\_kwargs = {

'FilterExpression': Key('year').between(\*year\_range),

'ProjectionExpression': "#yr, title, info.actors",

'ExpressionAttributeNames': {"#yr": "year"}

}

done = False

start\_key = None

while not done:

if start\_key:

scan\_kwargs['ExclusiveStartKey'] = start\_key

response = table.scan(\*\*scan\_kwargs)

display\_movies(response.get('Items', []))

start\_key = response.get('LastEvaluatedKey', None)

done = start\_key is None

if \_\_name\_\_ == '\_\_main\_\_':

parser = argparse.ArgumentParser()

parser.add\_argument("start\_year", help="Starting year, ex: 1950")

parser.add\_argument("end\_year", help="Ending year. ex: 1959")

args = parser.parse\_args()

year1 = int(args.start\_year)

year2 = int(args.end\_year)

query\_range = (year1, year2)

def print\_movies(movies):

for movie in movies:

print(f"\n{movie['year']} : {movie['title']}")

if 'info' in movie:

pprint(movie['info'])

else:

print("{'rating': NA}")

print(f"Scanning for movies released from {query\_range[0]} to {query\_range[1]}...")

scan\_movies(query\_range, print\_movies)

 For more information, refer to *Scan Operation* in the **Additional resources** section.

1. On the **File** menu, choose **Save As…** and save the file as

MoviesScan.py

1. Close the **MoviesScan.py** tab.
2. To run the **MoviesScan.py** script to query the table for all movies released between 1970 and 1980, run the following command:

python MoviesScan.py 1970 1980

**Expected Output** The output should display all movies in the table with a **year** value between 1970 and 1980, similar to the following:

\*\*\* EXPECTED OUTPUT \*\*\*

Scanning for movies released from 1970 to 1980...

1972 : Deliverance

{'actors': {'Jon Voight', 'Burt Reynolds', 'Ned Beatty'}}

1972 : The Godfather

{'actors': {'Al Pacino', 'James Caan', 'Marlon Brando'}}

TASK 2.5: USE THE AWS SDK TO FETCH ITEMS WITH THE QUERY OPERATION

Next, use the Query operation to locate movies released in a specific year.

1. On the **File** menu, choose **New File**.
2. In the **Untitled1** editor tab, copy and paste the following Python script, which is used to query the table for all movies that were released in a specific year:

from pprint import pprint

import boto3

from boto3.dynamodb.conditions import Key

import argparse

def query\_movies(year):

region=boto3.session.Session().region\_name

dynamodb = boto3.resource('dynamodb', region\_name=region) # low-level client

table = dynamodb.Table('movies')

response = table.query(

KeyConditionExpression=Key('year').eq(year)

)

return response['Items']

if \_\_name\_\_ == '\_\_main\_\_':

parser = argparse.ArgumentParser()

parser.add\_argument("Year", help="Search year, ex: 1950")

args = parser.parse\_args()

query\_year = int(args.Year)

print(f"Movies released in {query\_year}")

movies = query\_movies(query\_year)

for movie in movies:

print(movie['year'], ":", movie['title'])

 For more information, refer to *Query Operation* in the **Additional resources** section.

1. On the **File** menu, choose **Save As…** and save the file as

MoviesQueryItems.py

1. Close the **MoviesQueryItems.py** tab.
2. To run the **MoviesQueryItems.py** script to query the table for all movies released in 1972, run the following command:

python MoviesQueryItems.py 1972

**Expected Output** The output should display all movies in the table with a **year** value of 1972, similar to the following:

\*\*\* EXPECTED OUTPUT \*\*\*

Movies released in 1972

1972 : Deliverance

1972 : The Godfather

TASK 2.6: USE THE AWS SDK TO UPDATE AN ITEM

You’ve established a method for adding items to the table and then finding then. Now you want to add a brief summary of the plot for a movie.

1. On the **File** menu, choose **New File**.
2. In the **Untitled1** editor tab, copy and paste the following Python script, which updates a movie in the table with information about the plot:

from decimal import Decimal

from pprint import pprint

from botocore.exceptions import ClientError

import boto3

def update\_movie(title, year, rating, plot, actors):

region=boto3.session.Session().region\_name

dynamodb = boto3.resource('dynamodb', region\_name=region) # low-level client

table = dynamodb.Table('movies')

try:

response = table.update\_item(

Key={

'year': year,

'title': title

},

UpdateExpression="set info.rating=:r, info.plot=:p, info.actors=:a",

ExpressionAttributeValues={

':r': Decimal(rating),

':p': plot,

':a': actors

},

ReturnValues="UPDATED\_NEW"

)

except ClientError as e:

print("Update movie failed:",e.response['Error']['Message'])

else:

print("Update movie succeeded:","\n")

pprint(response)

return response

if \_\_name\_\_ == '\_\_main\_\_':

movie\_title = "The Godfather"

movie\_year = 1972

movie\_rating = Decimal('9.2')

movie\_plot = "The aging patriarch of an organized crime dynasty transfers control of his clandestine empire to his reluctant son."

movie\_actors = ["Marlon Brando", "Al Pacino", "James Caan"]

print("\n","Updating...","\n")

update\_movie(movie\_title, movie\_year, movie\_rating, movie\_plot, movie\_actors)

 For more information, refer to *UpdateItem Operation* in the **Additional resources** section.

1. On the **File** menu, choose **Save As…** and save the file as

MoviesUpdateItem.py

1. Close the **MoviesUpdateItem.py** tab.
2. To run the **MoviesUpdateItem.py** script to update the movie *The Godfather* with the plot, run the following command:

python MoviesUpdateItem.py

**Expected Output** The output should show that the update completed successfully, similar to the following:

\*\*\* EXPECTED OUTPUT \*\*\*

Updating...

Update movie succeeded:

{'Attributes': {'info': {'actors': ['Marlon Brando', 'Al Pacino', 'James Caan'],

'plot': 'The aging patriarch of an organized crime '

'dynasty transfers control of his clandestine '

'empire to his reluctant son.',

'rating': Decimal('9.2')}},

'ResponseMetadata': {'HTTPHeaders': {'connection': 'keep-alive',

'content-length': '259',

'content-type': 'application/x-amz-json-1.0',

'date': 'Thu, 05 Nov 2020 21:26:49 GMT',

'server': 'Server',

'x-amz-crc32': '1777035368',

'x-amzn-requestid': 'BD0T11P3TQH2QUQMMD999P3E83VV4KQNSO5AEMVJF66Q9ASUAAJG'},

'HTTPStatusCode': 200,

'RequestId': 'BD0T11P3TQH2QUQMMD999P3E83VV4KQNSO5AEMVJF66Q9ASUAAJG',

'RetryAttempts': 0}}

1. To query the table for *The Godfather* and verify the plot was added successfully, run the following command:

aws dynamodb get-item --table-name movies --key '{"year": {"N": "1972"}, "title": {"S": "The Godfather"}}'

TASK 2.7: USE THE AWS SDK TO DELETE AN ITEM

Now you want to be able to delete an entire item from the table.

1. On the **File** menu, choose **New File**.
2. In the **Untitled1** editor tab, copy and paste the following Python script, which is used to delete an item from a table:

from decimal import Decimal

from pprint import pprint

from botocore.exceptions import ClientError

import boto3

def update\_movie(title, year):

region=boto3.session.Session().region\_name

dynamodb = boto3.resource('dynamodb', region\_name=region) # low-level client

table = dynamodb.Table('movies')

try:

response = table.delete\_item(

Key={

'year': year,

'title': title

},

ConditionExpression = "attribute\_exists(info.actors)",

ReturnValues="ALL\_OLD"

)

except ClientError as e:

if e.response['Error']['Code'] == "ConditionalCheckFailedException":

print("Delete movie failed:",e.response['Error']['Message'])

else:

raise e

else:

if 'Attributes' in response:

print("Delete movie succeeded:","\n")

pprint(response['Attributes'])

else:

print("Movie not found")

if \_\_name\_\_ == '\_\_main\_\_':

movie\_title = "The Godfather"

movie\_year = 1972

print("\n","Deleting...","\n")

update\_movie(movie\_title, movie\_year)

 For more information, refer to *DeleteItem Operation* in the **Additional resources** section.

1. On the **File** menu, choose **Save As…** and save the file as

MoviesDeleteItem.py

1. Close the **MoviesDeleteItem.py** tab.
2. To run the **MoviesDeleteItem.py** script to delete the movie *The Godfather* from the table, run the following command.

python MoviesDeleteItem.py

**Expected Output** The output should show that the item was deleted successfully, similar to the following:

\*\*\* EXPECTED OUTPUT \*\*\*

Deleting...

Delete movie succeeded:

{'info': {'plot': 'The aging patriarch of an organized crime dynasty transfers '

'control of his clandestine empire to his reluctant son.',

'rating': Decimal('9')},

'title': 'The Godfather',

'year': Decimal('1972')}

1. To re-run the **MoviesQueryItems.py** script that you created previously to query the table for all movies released in 1972, run the following command:

python MoviesQueryItems.py 1972

The movie *The Godfather* should not show in the output.

TASK 2.8: USE THE AWS SDK TO DELETE A TABLE

You’ve finished your testing and are ready to delete the table.

1. On the **File** menu, choose **New File**.
2. In the **Untitled1** editor tab, copy and paste the following Python script, which deletes a table:

import boto3

def delete\_movie\_table():

region=boto3.session.Session().region\_name

dynamodb = boto3.resource('dynamodb', region\_name=region) # low-level client

table = dynamodb.Table('movies')

table.delete()

if \_\_name\_\_ == '\_\_main\_\_':

delete\_movie\_table()

print("movies table deleted.")

 For more information, refer to *DeleteTable Operation* in the **Additional resources** section.

1. On the **File** menu, choose **Save As…** and save the file as

MoviesDeleteTable.py

1. Close the **MoviesDeleteTable.py** tab.
2. To run the **MoviesDeleteTable.py** script to delete the **movies** table, run the following command:

python MoviesDeleteTable.py

**Expected Output** The output should show that the table was deleted successfully, similar to the following:

\*\*\* EXPECTED OUTPUT \*\*\*

movies table deleted.

1. To list all DynamoDB tables and verify that the **movies** table was deleted, run the following command:

aws dynamodb list-tables

 Congratulations! You have successfully created a DynamoDB table to store information about movies, created and updated items, performed queries to look up specific movies, deleted items, and deleted a table using the AWS SDK for Python.

**Task 3: Challenge activity**

Applying everything you have learned so far, perform the following tasks:

* Use the AWS CLI to create a new DynamoDB table using a JSON file that contains the table configuration. The table should use the following parameters:
  + Table name:

movies

* + Partition key:

year

* + Sort key:

title

* Use the AWS CLI to add items to the table using data contained in a JSON file.
* Create a Python script to add items to the table using data contained in a JSON file.

**Hints:**

* Use the [AWS CLI command reference for DynamoDB](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/dynamodb/index.html) and the [Boto3 documentation](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/dynamodb.html) to help inform your solution.
* While the

aws dynamodb put-item

 command is useful to add a single item to a table, you can’t use it to add multiple items from a file. Try

aws dynamodb batch-write-item

 instead.

 Refer to the **Answer key** section for the answers to this challenge.

**Conclusion**

 Congratulations! You now have successfully:

* Created and managed a DynamoDB table using the AWS CLI.
* Created and managed a DynamoDB table using the AWS SDK for Python.

**End lab**

Follow these steps to close the console and end your lab.

1. Return to the **AWS Management Console**.
2. At the upper-right corner of the page, choose **AWSLabsUser**, and then choose **Sign out**.
3. Choose **End lab** and then confirm that you want to end your lab.

**Additional resources**

* [Amazon DynamoDB Features](https://aws.amazon.com/dynamodb/features/)
* [Core Components of Amazon DynamoDB](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html)
* [Installing, Updating, and Uninstalling the AWS CLI](https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html#migrating)
* [Read/Write Capacity Mode](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadWriteCapacityMode.html)
* [Provisioned Mode](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadWriteCapacityMode.html#HowItWorks.ProvisionedThroughput.Manual)
* [create-table CLI Command](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/dynamodb/create-table.html)
* [put-item CLI Command](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/dynamodb/put-item.html)
* [Working with Scans in DynamoDB](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Scan.html)
* [scan CLI Command](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/dynamodb/scan.html)
* [Working with Queries in DynamoDB](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Query.html)
* [query CLI Command](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/dynamodb/query.html)
* [Reserved Words in DynamoDB](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ReservedWords.html)
* [Expression Attribute Names in DynamoDB](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Expressions.ExpressionAttributeNames.html)
* [update-item CLI Command](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/dynamodb/update-item.html)
* [delete-item CLI Command](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/dynamodb/delete-item.html)
* [delete-table CLI Command](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/dynamodb/delete-table.html)
* [AWS SDK for Python (Boto3)](https://aws.amazon.com/sdk-for-python/)
* [CreateTable Operation](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_CreateTable.html)
* [PutItem Operation](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_PutItem.html)
* [Scan Operation](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_Scan.html)
* [Query Operation](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_Query.html)
* [UpdateItem Operation](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_UpdateItem.html)
* [DeleteItem Operation](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_DeleteItem.html)
* [DeleteTable Operation](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_DeleteTable.html)

For more information about AWS Training and Certification, see [*https://aws.amazon.com/training/*](https://aws.amazon.com/training/).

*Your feedback is welcome and appreciated.*  
If you would like to share any feedback, suggestions, or corrections, please provide the details in our [*AWS Training and Certification Contact Form*](https://support.aws.amazon.com/#/contacts/aws-training).

**Answer key**

TASK 3

The following command creates a DynamoDB table from the settings in a JSON file (you can download a copy of a sample file [here](https://us-west-2-tcprod.s3.us-west-2.amazonaws.com/courses/SPL-TF-200-DBDYL1/v1.0.4.prod-e44781fa/scripts/movies-table.json)):

aws dynamodb create-table --cli-input-json file://movies-table.json

The following command adds the items in a JSON file to a DynamoDB table named *movies* (you can download a copy of a sample file [here](https://us-west-2-tcprod.s3.us-west-2.amazonaws.com/courses/SPL-TF-200-DBDYL1/v1.0.4.prod-e44781fa/scripts/moviedata-cli.json)):

aws dynamodb batch-write-item --request-items file://moviedata-cli.json