Lab Information

Contents

AWS Services are loading.

aws training and certification

Lab 3 (Python) - Develop Solutions Using Amazon DynamoDB

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* .

Duration

This lab will require **60 minutes** to complete.

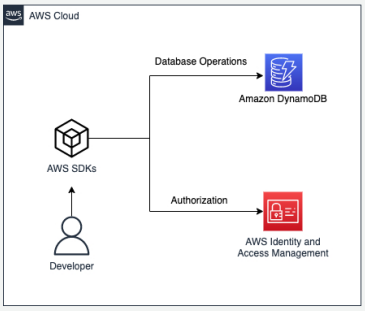
Prerequisites

This lab requires:

- Access to a Microsoft Windows or MacOS notebook computer with a WI-Fi connection.
- An Internet browser such as Chrome, Firefox, or IE9+.
- **Note:** Previous versions of Internet Explorer are not supported.
- **Note:** You can use an iPad or tablet device to access these directions in the lab console.
- **Additional information:** Review additional lab environment specific details in the Appendix.

Overview

In this lab, you will develop programs to create an Amazon DynamoDB table to store notes for your lab application. You will then programmatically perform CRUD operations on the table.



To develop the code for this lab, you will use the AWS SDK for Python (Boto3) Documentation . To optimize time, basic coding components are provided in AWS Cloud9 IDE. You will be challenged to complete the code in place of TODO sections during this lab. If you need additional help to complete the challenges, you can refer to the in-line solution clues or the completed program files located in the Solutions folder.

OBJECTIVES

After completing this lab, you will be able to:

- Interact with DynamoDB programmatically using low-level, document, and high-level APIs in your programs.
- Create a table using waiters with a partition key, sort key, and desired provisioned throughput.
- Load a table by reading JSON objects from a file.
- Retrieve items from a table using key attributes, filters, expressions, and paginations.
- Update items by adding new attributes and changing data conditionally.
- Access DynamoDB data using PartiQL.

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.

2. 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

You must first log out before logging into a different AWS account.

To logout, click here

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: Create an Amazon DynamoDB table

In this task, you will use the AWS Cloud9 IDE to complete the **createTable.py** program. Your task is to create a DynamoDB table with the name `Notes`, the partition key as **UserId**, the sort key as **NoteId**, and provisioned throughput of 5 read and write capacity units. You will use waiters as needed to ensure the availability of the table.

The **labRepo/createTable.py** file is processed in the following order by the **main** function:

- Read the config.ini file and use the settings for all static values.
- **createTable** creates the Amazon DynamoDB table in your lab AWS account.
- **waitForTableCreation** pauses the script until the table is ready.
- **getTableInfo** pulls the configuration of the table created in your account for confirmation.

Consider: This lab is designed for both experienced and newer developers:

- For more experienced developers who enjoy a challenge, there are **High-Level Instructions** before each task that should provide you enough information to help you complete the task.
- Once you complete the updates test your code to ensure it works, troubleshoot if needed, and then move on to the next task.
- For newer developers, there are **Detailed Instructions** to guide you through each step of the lab.

High-Level Instructions

Note: Make sure the Cloud9 environment is **ready**. You can verify this by checking the folder structure for a folder named **labRepo** and a folder named **Lab-Is-Ready**. If you don't see them, just wait a few moments for the final configurations to complete.

Note: If you see the following message choose **Accept**.

- **.c9/project.settings have been changed on disk**
  - The project settings file (.c9/project.settings) was updated outside the IDE. If you did not make the changes we suggest reviewing the file before accepting. Do you want to accept the new settings?

- Finish **labRepo/createTable.py** to create a new Amazon DynamoDB table.
- Complete the **TODO 1** section to create an Amazon DynamoDB service client
- Complete **TODO 2** to create the DynamoDB table.
- Complete **TODO 3** so that your script will wait until the table exists before continuing.
- Run the script and resolve any errors.

Detailed Instructions

TASK 1.1: CONFIGURE YOUR AWS CLOUD9 ENVIRONMENT AND INSPECT THE CONFIGURATION SETTINGS FOR THIS LAB

3. From the **AWS Management Console**, use the **AWS search bar** to search for `Cloud9` and then choose the service from the list of results.
4. On the **Environments** page, next to the **Lab3** environment listing, choose **Open**.

Note: Make sure the Cloud9 environment is **ready**. You can verify this by checking the folder structure for a folder named **labRepo** and a folder named **Lab-Is-Ready**. If you don't see them, just wait a few moments for the final configurations to complete.

Note: If you see the following message choose **Accept**.

- **.c9/project.settings have been changed on disk**
  - The project settings file (.c9/project.settings) was updated outside the IDE. If you did not make the changes we suggest reviewing the file before accepting. Do you want to accept the new settings?

5. Open the **labRepo/config.ini** file.

This configuration file has all the static values for all scripts in this lab. Review the parameters - table name, key names, and capacity levels. You do not need to change any of these settings. Keep the script open so that you can refer back to it as you develop the other scripts used in this lab.

TASK 1.2: CREATE AN AMAZON DYNAMODB SERVICE CLIENT

The scripts in this lab are organized with the same structure as the previous lab. All dependencies are imported, all of the functions needed are defined, the service client is created, and the **main** function is called.

6. Open the **labRepo/createTable.py** file

This file will be used to create the table. You have to complete the code so that the script successfully performs that function. In the file, after the functions are defined, you have the first TODO section.

7. **Copy/Paste:** Choose and insert the correct option to create a service client for **TODO 1**.

- Choice A

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

- Choice B

```
client = boto3.client('rds')
```

Answer: You can find the solution to this step in the TODO 1 solution section at the bottom of these instructions.

After the client is created, invoke the main function and pass it the service client to use. If any unhandled client or parameter validation errors are not handled within the main function, they will be detected at this level and displayed in the output.

TASK 1.3: DEVELOP CODE TO CREATE THE TABLE

To make the code in this lab easier to follow, all logic is called from the **main** function. All code challenges and any code that may be needed to format results, and is not a part of the learning objectives, is broken out into separate

functions.

In the main function, the information for the table to be created is read from the **config.ini** file.

In the first code challenge, you need to create an Amazon DynamoDB table. The **createTable** function is passed the DynamoDB client and an object with the information to create your table. Since the entire config object is passed, you have access to everything from the **config.ini** as the tableDefinition variable.

8. **Copy/Paste:** **TODO 2** : Choose and insert the correct code snippet to update the **createTable** function.

- Choice A

```
response = ddbClient.create_table(tableDefinition)
```

- Choice B

```
response = ddbClient.create_table(
        AttributeDefinitions=[
            {
                'AttributeName': tableDefinition["partitionKey"],
                'AttributeType': 'S',
            },
            {
                'AttributeName': tableDefinition["sortKey"],
                'AttributeType': 'N',
            },
        ],
        KeySchema=[
            {
                'AttributeName': tableDefinition["partitionKey"],
                'KeyType': 'HASH',
            },
            {
                'AttributeName': tableDefinition["sortKey"],
                'KeyType': 'RANGE',
            },
        ],
        ProvisionedThroughput={
            'ReadCapacityUnits': int(tableDefinition["readCapacity"]),
            'WriteCapacityUnits': int(tableDefinition["writeCapacity"]),
        },
        TableName=tableDefinition["tableName"]
    )
```

Answer: You can find the solution to this step in the TODO 2 solution section at the bottom of these instructions.

TASK 1.4: DEVELOP CODE TO ONLY CONTINUE AFTER THE TABLE IS READY AND RUN YOUR CODE

Creating a DynamoDB table is an asynchronous action and the response will return that the table is being created. Complete **TODO 3** so your code will wait for the table to be created.

9. **Copy/Paste:** Choose and insert the correct code snippet to complete **TODO 3** .

- Choice A

```
waiter = ddbClient.get_waiter('table_exists')
    waiter.wait(TableName=tableName)
```

- Choice B

```
waiter = ddbClient.get_waiter('table_not_exists')
    waiter.wait(TableName=tableName)
```

Answer: You can find the solution to this step in the TODO 3 solution section at the bottom of these instructions.

10. Save and close **createTable.py** .

With both code challenges complete, run the script.

11. Run the following command in the **AWS Cloud9 terminal** to change directories into the **~/environment** directory:

```
cd ~/environment
```

12. **Command:** Run **createTable.py** with the command below:

```
python labRepo/createTable.py
```

Expected output:

```
*******************************
**** This is OUTPUT ONLY. ****
*******************************

Creating an Amazon DynamoDB table "Notes"
with a partition key: "UserId"
and sort key: "NoteId".


Table Status: CREATING

Waiting for the table to be available...


Table is now available.


Table Status: ACTIVE
```

The script returns the response from the **create_table** call, waits, and then displays the information from the **describe_table** call.

Congratulations! You have successfully created a table in Amazon DynamoDB.

Task 2: Load data into the table

Now that you have successfully created the table, you can import the lab application data. In this task, complete the **loadData.py** file and use it to import the notes from the **notes.json** file. The **notes.json** file is the same file that you created in the previous lab, and it has already been downloaded to your **labRepo** folder.

The **labRepo/loadData.py** script uses the DynamoDB Table resource instead of the low level client and is processed in the following order:

- Read the config.ini file and use the settings for all static values.
- Create a list of JSON notes from the source JSON file.
- Create a table resource for the table created in the last task
- Iterate through the list of notes to add each to the table with the **putNote** function.

High-Level Instructions

- Finish **labRepo/loadData.py** to import notes to your new table.
- Complete the **TODO 4** section to add an item to the table using the resource and note passed to the function.
- Run the script and resolve any errors.

Detailed Instructions

TASK 2.1: DEVELOP CODE TO LOAD DATA INTO YOUR TABLE AND RUN THE CODE

13. Open **labRepo/loadData.py** .

This script imports notes from the **notes.json** file and then uses a DynamoDB table resource to add each note to the DynamoDB table you created in the last task.

14. **Copy/Paste:** **TODO 4** : Complete the putNote function with one of the following choices to add each note to the DynamoDB table.

- Choice A

```
table.put_item(
        Item={
            'UserId': note["UserId"],
            'Note': note["Note"]
        }
    )
```

- Choice B

```
table.put_item(
        Item={
            'UserId': note["UserId"],
            'NoteId': int(note["NoteId"]),
            'Note': note["Note"]
        }
    )
```

Answer: You can find the solution to this step in the TODO 4 solution section at the bottom of these instructions.

15. Save and close the **loadData.py** file.

16. **Command:** Run the **loadData.py** script from the **~/environment** directory to create records in the table with the command below:

```
python labRepo/loadData.py
```

Expected output:

```
*******************************
**** This is OUTPUT ONLY. ****
*******************************

 Loading "Notes" table with data from file "notes.json"


loading note {'UserId': 'testuser', 'NoteId': '001', 'Note': 'hello'}
loading note {'UserId': 'testuser', 'NoteId': '002', 'Note': 'this is my first note'}
loading note {'UserId': 'newbie', 'NoteId': '001', 'Note': 'Free swag code: 1234'}
loading note {'UserId': 'newbie', 'NoteId': '002', 'Note': 'I love DynamoDB'}
loading note {'UserId': 'student', 'NoteId': '001', 'Note': 'DynamoDB is NoSQL'}
loading note {'UserId': 'student', 'NoteId': '002', 'Note': 'A DynamoDB table is schemaless'}
loading note {'UserId': 'student', 'NoteId': '003', 'Note': 'PartiQL is a SQL compatible language for DynamoDB'}
loading note {'UserId': 'student', 'NoteId': '005', 'Note': 'Maximum size of an item is ____ KB ?'}
loading note {'UserId': 'student', 'NoteId': '004', 'Note': 'I love DyDB'}

Finished loading notes from the JSON file.
```

Congratulations! You have successfully loaded items into your table.

Task 3: Query data using a partition key and projections

It is just as important to get data from your DynamoDB table as it is to store data. In this task, you will complete the **queryData.py** file to get data from the table using only the partition key. You also display only values that you specify with a projection expression.

The **labRepo/queryData.py** file is processed in the following order by the **main** function:

- Read the config.ini file and use the settings for all static values.
- **queryNotesByPartitionKey** uses a DynamoDB query to return a specific item where the partition key is known, using a projection expression.

High-Level Instructions

- Finish **labRepo/queryData.py** to find notes for a specific UserId.
- Complete the **TODO 5** to query your table.
- Run the script and resolve any errors.

Detailed Instructions

TASK 3.1: DEVELOP CODE TO RETURN A SPECIFIC NOTE FROM YOUR TABLE

In this task, you will query the DynamoDB table to pull a specific item using the partition key. You will also use a Projection Expression to show only the NoteId and Note from the returned items. The **printNotes** function is used to convert DynamoDB JSON to a more readable format when printing them to the screen.

17. Open the **labRepo/queryData.py** file.

18. **Copy/Paste:** **TODO 5** : Complete the code using the parameters passed to the function to query the table by using the correct choice:

- Choice A

```
response = ddbClient.query(
        TableName=tableName,
        KeyConditionExpression='UserId = :userId',
        ExpressionAttributeValues={
            ':userId': {"S": qUserId}
        },
        ProjectionExpression="NoteId, Note"
    )
```

- Choice B

```
response = ddbClient.query(
        TableName=tableName,
        KeyConditionExpression='UserId = :userId',
        ExpressionAttributeValues={
            ':userId': qUserId
        },
        ProjectionExpression="NoteId, Note"
    )
```

Answer: You can find the solution to this step in the TODO 5 solution section at the bottom of these instructions.

Note: You can ignore the error stating **An attribute defined in json.encoder line 158 hides this method** .

19. **Command:** Run the **queryData.py** script from the **~/environment** directory to display the items from the table with the command below:

```
python labRepo/queryData.py
```

Expected output:

```
*******************************
**** This is OUTPUT ONLY. ****
*******************************

************
Querying for notes that belong to user student...

{"Note": "DynamoDB is NoSQL", "NoteId": "1"}
{"Note": "A DynamoDB table is schemaless", "NoteId": "2"}
{"Note": "PartiQL is a SQL compatible language for DynamoDB", "NoteId": "3"}
{"Note": "I love DyDB", "NoteId": "4"}
{"Note": "Maximum size of an item is ____ KB ?", "NoteId": "5"}
```

Congratulations! You now know how to retrieve items from your table.

Task 4: Scan the table with a paginator

In this task, you read all data from the notes table. You will use Scan operation to read the entire table.

The Scan operation returns one or more items and item attributes by accessing every item in a table or a secondary index. To have DynamoDB return fewer items, you can provide a FilterExpression operation. DynamoDB paginates the results from Scan operations. With pagination, the Scan results are divided into "pages" of data that are 1 MB in size (or less). An application can process the first page of results, then any consecutive pages you need. You can also limit the maximum number of items to be retrieved in a single page.

The **labRepo/paginateData.py** file is processed in the following order by the **main** function:

- Read the config.ini file and use the settings for all static values.
- **queryAllNotesPaginator** performs a scan with a paginator, and then display all pages returned.

High-Level Instructions

- Finish **labRepo/paginateData.py** to find notes for a specific UserId.
- Complete **TODO 6** to scan with a paginator.
- Run the script and resolve any errors.

Detailed Instructions

TASK 4.1: DEVELOP CODE TO RETURN ALL NOTES, DISPLAY THEM IN PAGES, AND RUN YOUR CODE

This file uses a paginator to break up the results into multiple pages that you can process. Remember, this task could be accomplished with a query or scan. The example uses a scan so that there are more records returned.

20. Open the **labRepo/paginateData.py** file.

21. **Copy/Paste:** **TODO 6** : Select the correct choice that creates the paginator and returns the items from each page to the printNotes function.

- Choice A

```
paginator = ddbClient.get_paginator('scan')

    page_iterator = paginator.paginate(
      TableName=tableName,
      PaginationConfig={
        'PageSize': pageSize
      },
      ProjectionProcessor=printNotes
      )
```

- Choice B

```
paginator = ddbClient.get_paginator('scan')

    page_iterator = paginator.paginate(
      TableName=tableName,
      PaginationConfig={
        'PageSize': pageSize
      })

    pageNumber = 0
    for page in page_iterator:
      if page["Count"] > 0:
```

```
        pageNumber += 1
        print("Starting page " + str(pageNumber))
        printNotes(page['Items'])
        print("End of page " + str(pageNumber) + "\n")
```

Answer: You can find the solution to this step in the TODO 6 solution section at the bottom of these instructions.

22. **Command:** Run the **paginateData.py** script from the **~/environment** directory to display the items in the table using the following command:

```
python labRepo/paginateData.py
```

Expected output:

```
******************************
**** This is OUTPUT ONLY. ****
******************************
Scanning with pagination...

Starting page 1
{"Note": "hello", "UserId": "testuser", "NoteId": "1"}
{"Note": "this is my first note", "UserId": "testuser", "NoteId": "2"}
{"Note": "DynamoDB is NoSQL", "UserId": "student", "NoteId": "1"}
End of page 1

Starting page 2
{"Note": "A DynamoDB table is schemaless", "UserId": "student", "NoteId": "2"}
{"Note": "PartiQL is a SQL compatible language for DynamoDB", "UserId": "student", "NoteId": "3"}
{"Note": "I love DyDB", "UserId": "student", "NoteId": "4"}
End of page 2

Starting page 3
{"Note": "Maximum size of an item is ____ KB ?", "UserId": "student", "NoteId": "5"}
{"Note": "Free swag code: 1234", "UserId": "newbie", "NoteId": "1"}
{"Note": "I love DynamoDB", "UserId": "newbie", "NoteId": "2"}
End of page 3
```

Congratulations! You now know how to retrieve items from your table.

Task 5: Update an item in the Table

In this task, you will perform two update actions. The first action is to update an existing item by adding a new attribute named *Is_Incomplete* and assign a value *Yes*. The second action is to update the same item only if a specific condition is met, that is, if *Is_Incomplete = Yes*. This will prevent items from unintended changes.

The **labRepo/updateItem.py** file is processed in the following order by the **main** function:

- Read the config.ini file and use the settings for all static values.
- **updateNewAttribute** creates a new attribute on a specific note to mark it as incomplete.
- **updateExistingAttributeConditionally** will update that item, but only if the flag is marked as incomplete.

High-Level Instructions

- Finish **labRepo/updateItem.py** to update an item two different ways.
- Complete **TODO 7** to add an attribute to an existing item.
- Complete **TODO 8** to update an item conditionally based on the attribute you added.
- Run the script and resolve any errors.

Detailed Instructions

TASK 5.1: DEVELOP CODE TO UPDATE AN ITEM IN YOUR DYNAMODB TABLE

Updating an item is more efficient than replacing an item if you only need to change some of the attribute values.

23. Open the **labRepo/updateItem.py** file.

24. **Copy/Paste:** For **TODO 7** , select the correct choice and insert it into the file to add an **Is_Incomplete** attribute to the item.

- Choice A

```
response = ddbClient.put_item(
      TableName=tableName,
      Item={
        'UserId': {'S': qUserId},
        'NoteId': {'N': str(qNoteId)},
        'Is_Incomplete': {'S': 'Yes'}
      },
      ReturnValues='ALL_NEW'
    )
```

- Choice B

```
response = ddbClient.update_item(
      TableName=tableName,
      Key={
        'UserId': {'S': qUserId},
        'NoteId': {'N': str(qNoteId)}
      },
      ReturnValues='ALL_NEW',
      UpdateExpression='SET Is_Incomplete = :incomplete',
      ExpressionAttributeValues={
          ':incomplete': {'S': 'Yes'}
      }
    )
```

Answer: You can find the solution to this step in the TODO 7 solution section at the bottom of these instructions.

TASK 5.2: DEVELOP CODE TO UPDATE AN ITEM CONDITIONALLY IN YOUR DYNAMODB TABLE AND RUN YOUR CODE

There are times where you only want to update an item if something about that item is true. For example, if you have an application that has many users, you may want to lock a record if it is in use. If the record is locked, your application would know that it can or cannot be changed in that state.

25. Continue editing **labRepo/updateItem.py** .

26. **Copy/Paste:** For **TODO 8** , select the correct choice to conditionally update an item. In this TODO, the code is correct in both choices, choose the option that will update the item with a note that is accurate.

28. **Copy/Paste:** For **TODO 8** , select the correct choice to conditionally update an item. In this TODO, the code is correct in both choices, choose the option that will update the item with a note that is accurate.

Note: You can check the value of the **notePrefix** variable in the **config.ini** file.

- Choice A

```
notePrefix += ' 400 KB'
```

- Choice B

```
notePrefix += ' 100 KB'
```

Answer: You can find the solution to this step in the TODO 8 solution section at the bottom of these instructions.

27. Save **updateItem.py** .

28. **Command:** Run the **updateItem.py** script from the **~/environment** directory to create records in the table with the command below:

```
python labRepo/updateItem.py
```

Expected output:

```
*******************************
**** This is OUTPUT ONLY. ****
*******************************

Updating the note flag for remediation...

{'Note': {'S': 'Maximum size of an item is ____ KB ?'}, 'UserId': {'S': 'student'}, 'NoteId': {'N': '5'}, 'Is_Incomplete': {'S': 'Yes'}}

Remediating the marked note...

{'Note': {'S': 'The maximum item size in DynamoDB is 400 KB'}, 'UserId': {'S': 'student'}, 'NoteId': {'N': '5'}, 'Is_Incomplete': {'S': 'No'}}
```

Congratulations! You can now update specific attributes in your table.

TASK 5.3: TEST UPDATING AN ITEM WHERE THE CONDITION IS NOT TRUE

So far, the script sets the Is_Incomplete flag and then updates the item while confirming the value of that flag. In this task, you will try to update the item without first setting the Is_Incomplete flag.

29. Delete line 16 from the **updateItem.py** file which displays the following code.

```
print(updateNewAttribute(ddbClient, tableName, qUserId, qNoteId))
```

30. Save and close **updateItem.py** .

31. **Command:** Run the **updateItem.py** script from the **~/environment** directory to try to update the item without first setting the flag with the following command:

```
python labRepo/updateItem.py
```

Expected output:

```
*******************************
**** This is OUTPUT ONLY. ****
*******************************

Updating the note flag for remediation...


Remediating the marked note...

Sorry, your update is invalid mate!
```

Congratulations! You can now update specific attributes in your table. You have also verified that you can conditionally update items as well.

Optional Task 6: Using PartiQL for DynamoDB

This lab has used traditional API methods for your database operations. One thing that you will notice is that it is completely different than traditional SQL based database engines. Amazon DynamoDB also supports **PartiQL** , a SQL-compatible query language, to select, insert, update, and delete data in your tables.

In this optional task, you will use **PartiQL** to query the same note that you first retrieved in task 3.

The **labRepo/partiQL.py** file is processed in the following order by the **main** function:

- Read the config.ini file and use the settings for all static values.
- **querySpecificNote** uses partiQL to query DynamoDB for a specific note.

High-Level Instructions

- Finish **labRepo/partiQL.py** to retrieve items from your table.
- Complete **TODO 9** to use PartiQL to query your table.
- Run the script and resolve any errors.

Detailed Instructions

TASK 6.1: DEVELOP CODE TO QUERY DYNAMODB WITH PARTIQL AND RUN YOUR CODE

In this task, complete the final TODO section of this lab.

32. Open the **labRepo/partiQL.py** file.

33. **Copy/Paste:** For **TODO 9** , choose and insert the correct code snippet in the **querySpecificNote** function.

- Choice A

```
response = ddbClient.query(
      KeyConditionExpression="SELECT * FROM " + tableName + " WHERE UserId = ? AND NoteId = ?",
```

```
        ExpressionAttributeValues=[
          {"S": qUserId},
          {"N": str(qNoteId)}
        ]
      )
```

- Choice B

```
response = ddbClient.execute_statement(
      Statement="SELECT * FROM " + tableName + " WHERE UserId = ? AND NoteId = ?",
      Parameters=[
        {"S": qUserId},
        {"N": str(qNoteId)}
      ]
    )
```

Answer: You can find the solution to this step in the TODO 9 solution section at the bottom of these instructions.

Note: You can ignore the error stating **An attribute defined in json.encoder line 158 hides this method** .

34. **Command:** Run the **partiQL.py** script from the **~/environment** directory to create records in the table using the command below:

```
python labRepo/partiQL.py
```

Expected output:

```
*******************************
**** This is OUTPUT ONLY. ****
*******************************

************
Querying for note 5 that belongs to user student...

{"Note": "The maximum item size in DynamoDB is 400 KB", "UserId": "student", "NoteId": "5", "Is_Incomplete": "No"}
```

Congratulations! You now know how to use PartiQL to interact with DynamoDB tables.

Summary

Congratulations on completing the python version of the lab! You can now:

- Interact with DynamoDB programmatically using low-level, document, and high-level APIs in your programs.
- Create a table using waiters with a partition key, sort key, and desired provisioned throughput.
- Load a table by reading JSON objects from a file.
- Retrieve items from a table using key attributes, filters, expressions, and paginations.
- Update items by adding new attributes and changing data conditionally.
- Access DynamoDB data using PartiQL.

End lab

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

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

Additional Resources

- Amazon DynamoDB Developer Guide
- AWS SDK for Python (Boto3) - DynamoDB Service

For more information about AWS Training and Certification, see *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* .

Code Challenge Solutions

TODO 1 SOLUTION

- **Choice A is the correct code snippet** .

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

- Choice B is incorrect because DynamoDB is not a relational database and has its own client.

Return to instructions

TODO 2 SOLUTION

- Choice A is incorrect, because the dictionary passed in the tableDefinition variable is not in the correct format to create a DynamoDB table.
- **Choice B is the correct code snippet** .

```
response = ddbClient.create_table(
        AttributeDefinitions=[
            {
                'AttributeName': tableDefinition["partitionKey"],
                'AttributeType': 'S',
            },
            {
                'AttributeName': tableDefinition["sortKey"],
                'AttributeType': 'N',
```

```
            },
        ],
        KeySchema=[
            {
                'AttributeName': tableDefinition["partitionKey"],
                'KeyType': 'HASH',
            },
            {
                'AttributeName': tableDefinition["sortKey"],
                'KeyType': 'RANGE',
            },
        ],
        ProvisionedThroughput={
            'ReadCapacityUnits': int(tableDefinition["readCapacity"]),
            'WriteCapacityUnits': int(tableDefinition["writeCapacity"]),
        },
        TableName=tableDefinition["tableName"]
    )
```

Return to instructions

TODO 3 SOLUTION

- **Choice A is the correct code snippet** . You want use the *table_exists* waiter to pause until the table is available.

```
waiter = ddbClient.get_waiter('table_exists')
    waiter.wait(TableName=tableName)
```

- Choice B is incorrect because the parameter is not set correctly.

Return to instructions

TODO 4 SOLUTION

- Choice A is incorrect because it does not include the *NoteId* attribute. For a composite primary key, you must provide both values for both the partition key and the sort key.
- **Choice B is the correct code snippet** .

```
table.put_item(
        Item={
            'UserId': note["UserId"],
            'NoteId': int(note["NoteId"]),
            'Note': note["Note"]
        }
    )
```

Return to instructions

TODO 5 SOLUTION

- **Choice A is the correct code snippet** .

```
response = ddbClient.query(
        TableName=tableName,
        KeyConditionExpression='UserId = :userId',
        ExpressionAttributeValues={
            ':userId': {"S": qUserId}
        },
        ProjectionExpression="NoteId, Note"
    )
```

- Choice B is incorrect because it does not have the ExpressionsAttributeValues formatted correctly. The values must specify the data type of the value.

Return to instructions

TODO 6 SOLUTION

- Choice A is incorrect because you have to iterate through the results of the paginate call to process them.
- **Choice B is the correct code snippet.**

```
paginator = ddbClient.get_paginator('scan')

    page_iterator = paginator.paginate(
        TableName=tableName,
        PaginationConfig={
            'PageSize': pageSize
        })

    pageNumber = 0
    for page in page_iterator:
        if page["Count"] > 0:
            pageNumber += 1
            print("Starting page " + str(pageNumber))
            printNotes(page['Items'])
            print("End of page " + str(pageNumber) + "\n")
```

Return to instructions

TODO 7 SOLUTION

- Choice A is incorrect because even though it is valid syntax, it would replace the entire note with the item passed. In this case, removing the actual Note attribute.
- **Choice B is the correct code snippet** .

```
response = ddbClient.update_item(
        TableName=tableName,
        Key={
            'UserId': {'S': qUserId}
```

```
            'NoteId': {'N': str(qNoteId)}
        },
        ReturnValues='ALL_NEW',
        UpdateExpression='SET Is_Incomplete = :incomplete',
        ExpressionAttributeValues={
            ':incomplete': {'S': 'Yes'}
        }
    )
```

Return to instructions

TODO 8 SOLUTION

- **Choice A is the correct code snippet** . The maximum item size in DynamoDB is 400KB.

```
notePrefix += ' 400 KB'
```

- Choice B is incorrect because 100KB is not the maximum item size in DynamoDb.

Return to instructions

TODO 9 SOLUTION

- Choice A is incorrect because the parameters are not set properly.
- **Choice B is the correct code snippet** To use PartiQL you must use the execute_statement method.

```
response = ddbClient.execute_statement(
        Statement="SELECT * FROM " + tableName + " WHERE UserId = ? AND NoteId = ?",
        Parameters=[
            {"S": qUserId},
            {"N": str(qNoteId)}
        ]
    )
```

Return to instructions

Appendix

AWS SERVICES NOT USED IN THIS LAB

AWS services that are not used in this lab are deactivated in the lab environment. In addition, the capabilities of the services used in this lab are limited to what the lab requires. Expect errors when accessing other services or performing actions beyond those provided in this lab guide.

ICON KEY

Various icons are used throughout this lab to call attention to different types of instructions and notes. While not all of the icons will be used, the following list explains the purpose for each icon:

- **Command:** A command that you must run.
- **Expected output:** A sample output that you can use to verify the output of a command or edited file.
- **Note:** A note, tip, or important guidance.
- **Additional information:** Where to find more information.
- **Caution:** Information of special interest or importance (not so important to cause problems with the equipment or data if you miss it, but it could result in the need to repeat certain steps).
- **WARNING:** An action that is irreversible and could potentially impact the failure of a command or process (including warnings about configurations that cannot be changed after they are made).
- **Consider:** A moment to pause to consider how you might apply a concept in your own environment or to initiate a conversation about the topic at hand.
- **Copy/Paste:** A code block that displays the contents of a script or file you need to copy and paste that has been pre-created for you. When you need to copy only a certain part of a code block, there will be numbered **TODO** comments in the code.
- **Knowledge check:** An opportunity to check your knowledge and test what you have learned.
- **Security:** An opportunity to incorporate security best practices.
- **Refresh:** A time when you might need to refresh a web browser page or list to show new information.
- **Copy command:** A time when copying a command, script, or other text to a text editor (to edit specific variables within it) might be easier than editing directly in the command line or terminal.
- **Hint:** A hint to a question or challenge.
- **Answer:** An answer to a question or challenge.
- **Group effort:** A time when you must work together with another student to complete a task.

Return to the instructions