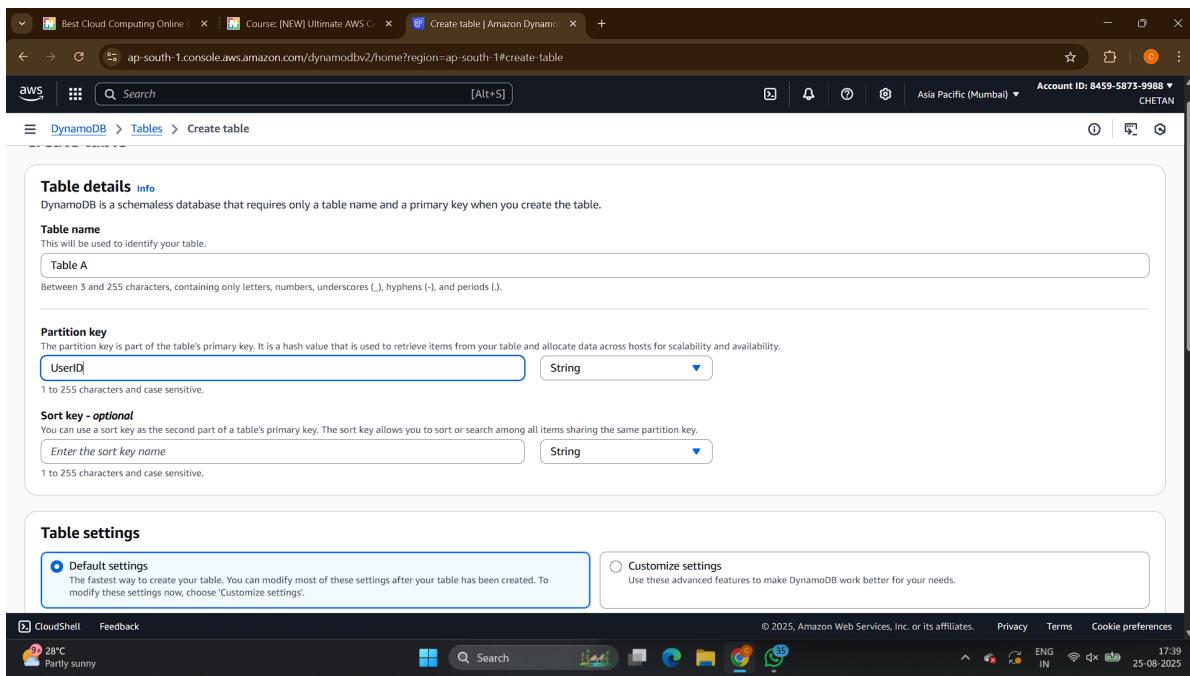


# Cycle 08 AWS Homework

## Task 1: Experiment with Partition and Sort Keys

- Create Table A with a simple primary key:
  - Table: **Users**
  - Partition Key: **UserID** (e.g., string or number)



The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. The current step is 'Table details'. It includes fields for 'Table name' (set to 'Table-A'), 'Partition key' (set to 'UserID' of type 'String'), and 'Sort key - optional' (left empty). Below these, there are sections for 'Table settings' with 'Default settings' selected, and 'Customize settings' which is described as using advanced features to make DynamoDB work better. The status bar at the bottom shows it's 25-08-2025.

The screenshot shows the 'Create table' wizard continuing through the 'Table settings' step. Under 'Provisioned', the 'Provisioned' option is selected, allowing users to manage costs by allocating read/write capacity in advance. The 'On-demand' option is also shown. Below this, sections for 'Read capacity' and 'Write capacity' are displayed, each with 'Auto scaling' and 'Off' options. The status bar at the bottom shows it's 25-08-2025.

The screenshot shows the AWS DynamoDB console. On the left, there's a navigation sidebar with 'DynamoDB' selected, followed by 'Tables', 'Explore items', 'PartiQL editor', 'Backups', 'Exports to S3', 'Imports from S3', 'Integrations', 'Reserved capacity', and 'Settings'. Below that is a section for 'DAX' with 'Clusters', 'Subnet groups', 'Parameter groups', and 'Events'. The main area is titled 'Tables (1) Info' and shows a table with one item. The table has columns: Name, Status, Partition key, Sort key, Indexes, Replication Regions, Deletion protection, Favorite, and Read capacity n. The single item is 'Table-A', which is Active, with a Partition key of 'UserID (\$)' and a Sort key of '-'. It has 0 indexes, 0 replication regions, Off deletion protection, and is not a favorite. The provisioned read capacity is 1. At the top right, there are 'Actions', 'Delete', and 'Create table' buttons.

The screenshot shows the 'Create item' dialog for 'Table-A'. A green success message at the top says 'The Table-B table was created successfully.' Below it, the heading 'Create item' is followed by a sub-instruction: 'You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep. [Learn more](#)'. The 'Form' view is selected. The 'Attributes' section contains two rows. The first row has 'Attribute name' 'UserID - Partition key', 'Value' '101', and 'Type' 'String'. The second row has 'Attribute name' 'Username', 'Value' 'bob', and 'Type' 'String'. There is a 'Remove' button next to the second row. At the bottom right are 'Cancel' and 'Create item' buttons. The status bar at the bottom shows 'CloudShell Feedback' and system icons like battery level, signal strength, and date/time.

**Create item**

You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep. [Learn more](#)

**Attributes**

Attribute name	Value	Type
UserID - Partition key	102	String
User Name	Aliza	String

[Add new attribute](#) | [Cancel](#) | [Create item](#)

**DynamoDB**

- Dashboard
- Tables
- Explore items**
- PartiQL editor
- Backups
- Exports to S3
- Imports from S3
- Integrations
- Reserved capacity
- Settings

**DAX**

- Clusters
- Subnet groups
- Parameter groups
- Events

**Items | Amazon DynamoDB**

**Table-A**

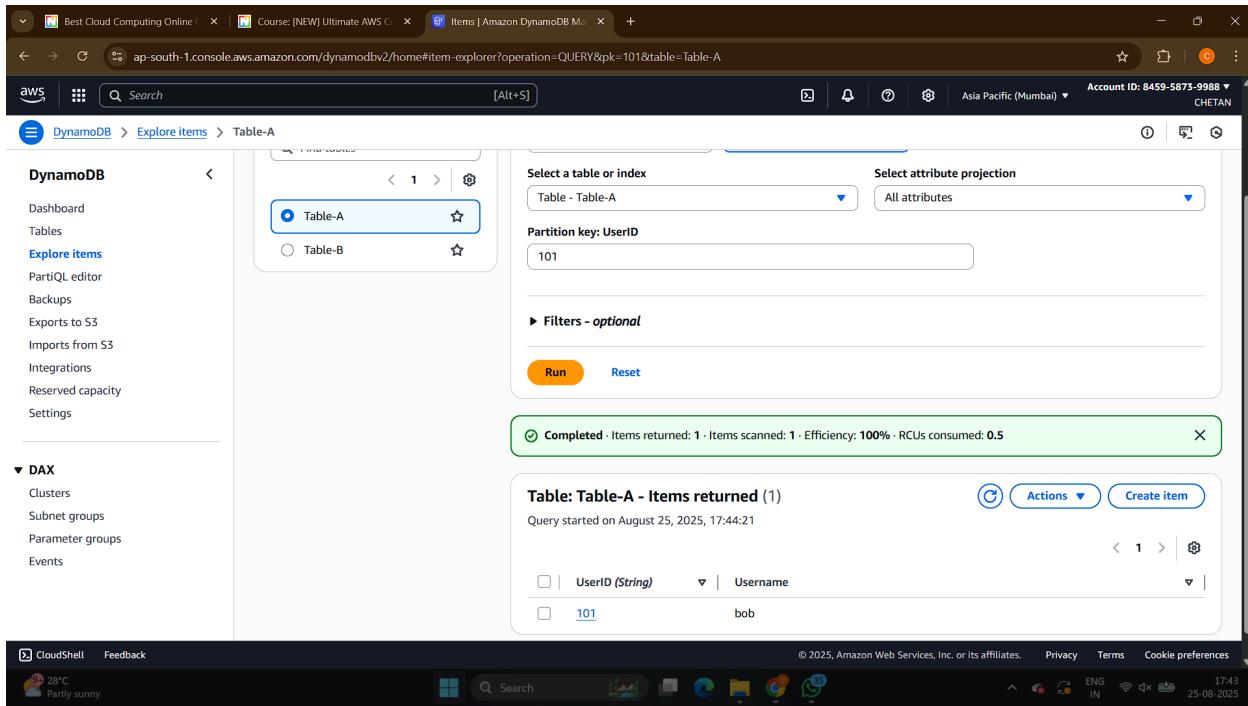
**Select a table or index**: Table - Table-A | **Select attribute projection**: All attributes

**Completed** · Items returned: 0 · Items scanned: 0 · Efficiency: 100% · RCU consumed: 0.5

**Table: Table-A - Items returned (2)**

Scan started on August 25, 2025, 17:42:38

User ID	User Name
102	Aliza
101	bob



- Create Table B with a composite primary key:
    - Table: **Orders**
    - Partition Key: **CustomerID**
    - Sort Key: **OrderDate** (allows for sorting and range querying)
  - Load sample data into both tables.

**Table details** Info  
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

**Table name**  
This will be used to identify your table.

**Partition key**  
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

1 to 255 characters and case sensitive.

**Sort key - optional**  
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

1 to 255 characters and case sensitive.

**Table settings**

**Default settings**  
The fastest way to create your table. You can modify most of these settings after your table has been created. To

**Customize settings**  
Use these advanced features to make DynamoDB work better for your needs.

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

**Read capacity**

**Auto scaling** Info  
Dynamically adjusts provisioned throughput capacity on your behalf in response to actual traffic patterns.

On  
 Off

**Provisioned capacity units**

**Write capacity**

**Auto scaling** Info  
Dynamically adjusts provisioned throughput capacity on your behalf in response to actual traffic patterns.

On  
 Off

**Provisioned capacity units**

**Warm throughput** Info  
Increasing the warm throughput value pre-warms your table to handle planned peak events without throttling or scaling delays. By default, warm throughput values are visible for all tables and global secondary indexes. These values automatically adjust as you increase your provisioned throughput or on-demand consumption without extra charges, but if you choose to change them manually, additional charges apply. Learn more about [Amazon](#)

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

The screenshot shows the AWS DynamoDB console interface. On the left, there's a navigation sidebar with options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Below that is a section for DAX with Clusters, Subnet groups, Parameter groups, and Events. The main content area is titled "Tables (2) Info". It lists two tables: "Table-A" and "Table-B". Table-A has a Partition key of "UserID (\$)" and a Sort key of "-". Table-B has a Partition key of "CustomerID (\$)" and a Sort key of "OrderDate (\$)". Both tables are active and have "Provisioned" read and write capacity. At the top right, there are buttons for Actions, Delete, and Create table.

The screenshot shows the "Create item" dialog for Table-B. The title is "Create item" and it says "You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep." There are three attributes defined: "CustomerID - Partition key" with value "101" and type "String"; "OrderDate - Sort key" with value "24022004" and type "String"; and "CustomerName" with value "chetan" and type "String". At the bottom right are "Cancel" and "Create item" buttons. The status bar at the bottom indicates it's 25-08-2025.

The item has been saved successfully.

**Create item**

You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep. [Learn more](#)

Attribute name	Value	Type
CustomerID - Partition key	102	String
OrderDate - Sort key	24052004	String
CustomerName	Rajesh	String

Add new attribute ▾

Cancel Create item

CloudShell Feedback

28°C Partly sunny

Search [Alt+S]

DynamoDB > Explore items > Table-B

Find tables Table-B

Select a table or index Table - Table-B Select attribute projection All attributes

Filters - optional Run Reset

Completed · Items returned: 0 · Items scanned: 0 · Efficiency: 100% · RCU consumed: 0.5

Table: Table-B - Items returned (2)

Scan started on August 25, 2025, 17:44:55

CustomerID (String)	OrderDate (String)	CustomerName
102	24032004	Rajesh
101	24022004	chetan

Actions ▾ Create item

CloudShell Feedback

28°C Partly sunny

Best Cloud Computing Online | Course: [NEW] Ultimate AWS | Create item | Amazon DynamoDB

ap-south-1.console.aws.amazon.com/dynamodbv2/home#edit-item?itemMode=1&route=ROUTE\_ITEM\_EXPLORER&table=Table-B

aws Search [Alt+S]

DynamoDB > Explore items: Table-B > Create item

### Create item

You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep. [Learn more](#)

**Attributes**

Attribute name	Value	Type	Add new attribute ▾
CustomerID - Partition key	103	String	
OrderDate - Sort key	24042004	String	
CustomerName	Mukesh	String	<a href="#">Remove</a>

[Cancel](#) [Create item](#)

CloudShell Feedback

28°C Partly sunny

Best Cloud Computing Online | Course: [NEW] Ultimate AWS | Items | Amazon DynamoDB

ap-south-1.console.aws.amazon.com/dynamodbv2/home#item-explorer?table=Table-B

aws Search [Alt+S]

DynamoDB > Explore items > Table-B

**DynamoDB**

- Dashboard
- Tables
- Explore items**
- PartiQL editor
- Backups
- Exports to S3
- Imports from S3
- Integrations
- Reserved capacity
- Settings

**DAX**

- Clusters
- Subnet groups
- Parameter groups
- Events

Select a table or index

Table - Table-B

Select attribute projection

All attributes

Filters - optional

[Run](#) [Reset](#)

Completed - Items returned: 0 - Items scanned: 0 - Efficiency: 100% - RCU consumed: 0.5

**Table: Table-B - Items returned (3)**

Scan started on August 25, 2025, 17:44:55

	CustomerID (String)	OrderDate (String)	CustomerName
<input type="checkbox"/>	103	24042004	Mukesh
<input type="checkbox"/>	102	24032004	Rajesh
<input type="checkbox"/>	101	24022004	chetan

The screenshot shows the AWS DynamoDB Item Explorer interface. On the left, there's a navigation sidebar with options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Under the 'Explore items' section, 'Table-B' is selected. The main area is titled 'Scan or query items' with a 'Query' button selected. It shows the configuration for a query: 'Table - Table-B', 'Partition key: CustomerID' set to '102', 'Sort key: OrderDate' set to 'Greater than or equal to 24022004', and a 'Sort descending' checkbox that is unchecked. Below these fields are 'Filters - optional' and 'Run' and 'Reset' buttons. A green status bar at the bottom indicates 'Completed - Items returned: 1 - Items scanned: 1 - Efficiency: 100% - RCU consumed: 0.5'. At the bottom of the page, there's a footer with links for CloudShell, Feedback, and various system icons.

This screenshot shows the results of the query from the previous screen. The top part of the interface is identical to the first screenshot. The main results area is titled 'Table: Table-B - Items returned (1)' and shows a single item with the following details:

CustomerID (String)	OrderDate (String)	CustomerName
102	24022004	Rajesh

Below the table, it says 'Query started on August 25, 2025, 17:48:12'. The footer at the bottom includes links for CloudShell, Feedback, and various system icons.

- Use AWS CLI for:
  - **GetItem** on **Users** table (retrieves single item by **UserID**).

- **Query** on **Orders** table (retrieves all orders for a **CustomerID** and can filter by date range).
- Understanding:
  - Simple key allows unique retrieval by partition key only.
  - Composite key offers more querying flexibility by partition + sort key.

Example AWS CLI for creating tables with provisioned throughput:

```
bashaws dynamodb create-table \
--table-name Users \
--attribute-definitions AttributeName=UserID,AttributeType=S \
--key-schema AttributeName=UserID,KeyType=HASH \
--provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5

aws dynamodb create-table \
--table-name Orders \
--attribute-definitions AttributeName=CustomerID,AttributeType=S AttributeName=OrderDate,AttributeType=S \
--key-schema AttributeName=CustomerID,KeyType=HASH AttributeName=OrderDate,KeyType=RANGE \
--provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
```

---

## Task 2: Compare Consistency Models

- Write an item to a DynamoDB table using AWS CLI.
- Read the item immediately:
  - Eventually consistent read: **-consistent-read false** (default)
  - Strongly consistent read: **-consistent-read true**
- Result:
  - Eventually consistent may show stale data if read too soon after write.
  - Strongly consistent read guarantees the latest committed data.
- Use cases for eventual consistency favor lower cost and latency, whereas strong consistency guarantees fresh data at a higher cost.

```

Administrator: Command Prompt
Microsoft Windows [Version 10.0.26120.5751]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>aws dynamodb create-table --table-name table-c --attribute-definitions AttributeName=Id,AttributeType=S --key-schema AttributeName=Id,KeyType=HASH --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Id",
                "AttributeType": "S"
            }
        ],
        "TableName": "table-c",
        "KeySchema": [
            {
                "AttributeName": "Id",
                "KeyType": "HASH"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2025-08-25T18:04:59.886000+05:30",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:ap-south-1:845958739988:table/table-c",
        "TableId": "e81db670-97de-4ac0-b516-e734766a6f56",
        "DeletionProtectionEnabled": false
    }
}

C:\Windows\System32>

```

The screenshot shows the AWS DynamoDB console interface. On the left, there's a navigation sidebar with 'DynamoDB' selected, followed by 'Tables', 'Explore items', 'PartQL editor', 'Backups', 'Exports to S3', 'Imports from S3', 'Integrations', 'Reserved capacity', and 'Settings'. Below this is a 'DAX' section with 'Clusters', 'Subnet groups', 'Parameter groups', and 'Events'. The main content area has a green success message: 'The request to delete the "table-c" table has been submitted successfully.' Below this, the 'Tables (1) Info' section displays a table with one row for 'table-c'. The table columns include Name (table-c), Status (Active), Partition key (Id (\$)), Sort key (-), Indexes (0), Replication Regions (0), Deletion protection (Off), Favorite (star icon), and Read capacity units (Provisioned (5)). At the top of the main content area, there are buttons for Actions, Delete, and Create table. The browser's address bar shows the URL: ap-south-1.console.aws.amazon.com/dynamodbv2/home#tables. The status bar at the bottom indicates it's 18:05 on 25-08-2025.

```

Administrator: Command Prompt
        "WriteCapacityUnits": 5
    },
    "TableSizeBytes": 0,
    "ItemCount": 0,
    "TableArn": "arn:aws:dynamodb:ap-south-1:845958739988:table/table-c",
    "TableId": "e81db670-97de-44c0-b516-e734766a6f5b",
    "DeletionProtectionEnabled": false
}
}

C:\Windows\System32>aws dynamodb describe-table --table-name table-c
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Id",
                "AttributeType": "S"
            }
        ],
        "TableName": "table-c",
        "KeySchema": [
            {
                "AttributeName": "Id",
                "KeyType": "HASH"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": "2025-08-25T18:04:59.886000+05:30",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:ap-south-1:845958739988:table/table-c",
        "TableId": "e81db670-97de-44c0-b516-e734766a6f5b",
        "DeletionProtectionEnabled": false,
        "WarmThroughput": {
            "ReadUnitsPerSecond": 5,
            "WriteUnitsPerSecond": 5,
            "Status": "ACTIVE"
        }
    }
}

C:\Windows\System32>

```

28°C Partly sunny ENG IN 18:05 25-08-2025

```

Administrator: Command Prompt
C:\Windows\System32>aws dynamodb put-item --table-name table-c --item "{\"Id\": {\"S\": \"123\"}, \"Attribute1\": {\"S\": \"Value1\"}}"
C:\Windows\System32>
C:\Windows\System32>


```

28°C Partly sunny ENG IN 18:07 25-08-2025

The screenshot shows the AWS DynamoDB console interface. On the left, there's a sidebar with options like Dashboard, Tables, Explore items (which is selected), PartQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Below that is a section for DAX with Clusters, Subnet groups, Parameter groups, and Events. The main area has a search bar at the top with 'Find tables' and a dropdown showing 'table-c'. There are two tabs: 'Scan' (selected) and 'Query'. Under 'Scan', it says 'Select a table or index' (Table - table-c) and 'Select attribute projection' (All attributes). Below that is a 'Filters - optional' section with 'Run' and 'Reset' buttons. A green message box says 'Completed · Items returned: 1 · Items scanned: 1 · Efficiency: 100% · RCU consumed: 0.5'. The table below has columns for Id (String) and Attribute1. One row is shown with Id '123' and Attribute1 'Value1'. At the bottom of the page, there's a CloudShell section with a weather icon (28°C, Partly sunny), a search bar, and a toolbar with icons for File, Home, Recent, and others. The status bar at the bottom right shows 'CloudShell' and the date '25-08-2025'.

```

Administrator: Command Prompt
C:\Windows\System32>aws dynamodb put-item --table-name table-c --item "{\"Id\": {\"S\": \"123\"}, \"Attribute1\": {\"S\": \"Value1\"}}"
C:\Windows\System32>aws dynamodb scan --table-name table-c
{
    "Items": [
        {
            "Id": {
                "S": "123"
            },
            "Attribute1": {
                "S": "Value1"
            }
        }
    ],
    "Count": 1,
    "ScannedCount": 1,
    "ConsumedCapacity": null
}

C:\Windows\System32>

```

The screenshot shows a Windows Command Prompt window with a black background. It displays the output of two AWS CLI commands. The first command, 'aws dynamodb put-item', adds a new item to the 'table-c' table with an Id of '123' and an Attribute1 of 'Value1'. The second command, 'aws dynamodb scan', retrieves all items from the table, returning a single item with the same key-value pair. The prompt ends with 'C:\Windows\System32>'. The taskbar at the bottom shows various pinned icons and the date '25-08-2025'.

```
Administrator: Command Prompt
Unknown options: true

C:\Windows\System32>aws dynamodb get-item --table-name table-c --key "{\"Id\": {\"S\": \"123\"}}"
{
    "Item": {
        "Id": {
            "S": "123"
        },
        "Attribute1": {
            "S": "Value1"
        }
    }
}

C:\Windows\System32>
C:\Windows\System32>
```



```
Administrator: Command Prompt
)

C:\Windows\System32>aws dynamodb get-item --table-name table-c --key "{\"Id\": {\"S\": \"123\"}}" --consistent-read
{
    "Item": {
        "Id": {
            "S": "123"
        },
        "Attribute1": {
            "S": "Value1"
        }
    }
}

C:\Windows\System32>
C:\Windows\System32>
```



## Task 3: Experience Throttling in Provisioned Mode

- Create a table with low provisioned throughput, e.g., 1 RCU and 1 WCU.

- Use a script (e.g., Python Boto3) to attempt rapid write operations (e.g., 10 writes in a loop).
- Observe `ProvisionedThroughputExceededException` errors indicating throttling.
- Learn importance of:
  - Proper capacity provisioning based on load.
  - Implementing retry and backoff logic.
  - Using Auto Scaling or On-Demand mode for variable workloads.

## CODE:

```
import boto3
import time

# Initialize DynamoDB client with your preferred region
client = boto3.client('dynamodb', region_name='us-east-1')

# Change region if needed

table_name = 'table-low-capacity'

# Step 1: Create table with 1 RCU and 1 WCU

def create_table():
    response = client.create_table(
        TableName=table_name,
        AttributeDefinitions=[
            {'AttributeName': 'Id', 'AttributeType': 'S'},
        ],
        KeySchema=[
            {'AttributeName': 'Id', 'KeyType': 'HASH'},
        ],
        ProvisionedThroughput={
            'ReadCapacityUnits': 1,
            'WriteCapacityUnits': 1
        }
    )

```

```

)
print("Creating table...")

# Wait for the table to become active

waiter = client.get_waiter('table_exists')
waiter.wait(TableName=table_name)
print("Table created and active.")

# Step 2: Write 10 items rapidly to the table

def write_items():
    for i in range(10):
        item = {
            'Id': {'S': str(i)},
            'Attribute': {'S': f'Value{i}'}
        }
        client.put_item(TableName=table_name, Item=item)
        print(f'Wrote item {i}')

# Optional: slight delay to modulate speed

# time.sleep(0.1)

if __name__ == '__main__':
    create_table()
    write_items()

```

## Screenshots:

The screenshot shows the Visual Studio Code interface with the Python extension installed. The code editor displays a script named `boto3.py` which contains Python code for creating a DynamoDB table. The terminal below shows the command-line output of running the script, indicating the table was created successfully.

```
import boto3
# Initialize DynamoDB client with your preferred region
client = boto3.client('dynamodb', region_name='us-east-1') # Change region if needed
table_name = 'table-low-capacity'
# Step 1: Create table with 1 RCU and 1 WCU
def create_table():
    response = client.create_table(
        TableName=table_name,
        AttributeDefinitions=[
            {'AttributeName': 'Id', 'AttributeType': 'S'},
        ],
        KeySchema=[
            {'AttributeName': 'Id', 'KeyType': 'HASH'},
        ],
        ProvisionedThroughput={
            'ReadCapacityUnits': 1,
            'WriteCapacityUnits': 1
        }
)
Creating table...
Table created and active.
Wrote item 0
Wrote item 1
Wrote item 2
```

This screenshot shows the same `boto3.py` script as the first one, but it includes a `waiter` object to wait for the table to become active before proceeding. The terminal output shows the table was created and became active.

```
# Step 1: Create table with 1 RCU and 1 WCU
def create_table():
    response = client.create_table(
        TableName=table_name,
        AttributeDefinitions=[
            {'AttributeName': 'Id', 'AttributeType': 'S'},
        ],
        KeySchema=[
            {'AttributeName': 'Id', 'KeyType': 'HASH'},
        ],
        ProvisionedThroughput={
            'ReadCapacityUnits': 1,
            'WriteCapacityUnits': 1
        }
)
print("Creating table...")
# Wait for the table to become active
waiter = client.get_waiter('table_exists')
waiter.wait(TableName=table_name)
Creating table...
Table created and active.
Wrote item 0
Wrote item 1
Wrote item 2
```

The screenshot shows the VS Code interface with a Python file named `boto3.py` open. The code creates a table and writes 10 items to it. The terminal output shows the command run and the resulting table creation and item writes.

```
import boto3.py

C: > Users > cheta > Desktop > DE Homework > dynamodb > sdk - rru 1 > import boto3.py > ...
8 def create_table():
16     'ReadCapacityUnits': 1,
17     'WriteCapacityUnits': 1
18
19     }
20
21     )
22     print("Creating table...")
23     # Wait for the table to become active
24     waiter = client.get_waiter('table_exists')
25     waiter.wait(TableName=table_name)
26     print("Table created and active.")
27
28     # Step 2: Write 10 items rapidly to the table
29     def write_items():
30         for i in range(10):
31             item = {
32                 'Id': {'S': str(i)},
33                 'Attribute': {'S': f'Value{i}'}
34             }
35             client.put_item(TableName=table_name, Item=item)
36             print(f'Wrote item {i}')
37             # Optional: slight delay to modulate speed
38             # time.sleep(0.1)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\cheta\Desktop\DE Homework\dynamodb\ sdk - rru 1> c:; cd 'c:\Users\cheta\Desktop\DE Homework\dynamodb\ sdk - rru 1'; & 'c:\Users\cheta\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\cheta\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '61162' '--' 'c:\Users\cheta\Desktop\DE Homework\dynamodb\ sdk - rru 1\import boto3.py'
Creating table...
Table created and active.
Wrote item 0
Wrote item 1
Wrote item 2

28C Partly sunny
```

This screenshot is identical to the one above, showing the same code execution and terminal output. It demonstrates the creation of a DynamoDB table and the writing of 10 items to it.

```
import boto3.py

C: > Users > cheta > Desktop > DE Homework > dynamodb > sdk - rru 1 > import boto3.py > ...
29     def write_items():
30         item = {
31             'Id': {'S': str(i)},
32             'Attribute': {'S': f'Value{i}'}
33         }
34         client.put_item(TableName=table_name, Item=item)
35         print(f'Wrote item {i}')
36         # Optional: slight delay to modulate speed
37         # time.sleep(0.1)
38
39
40     if __name__ == '__main__':
41         create_table()
42         write_items()
43

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\cheta\Desktop\DE Homework\dynamodb\ sdk - rru 1> c:; cd 'c:\Users\cheta\Desktop\DE Homework\dynamodb\ sdk - rru 1'; & 'c:\Users\cheta\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\cheta\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '61162' '--' 'c:\Users\cheta\Desktop\DE Homework\dynamodb\ sdk - rru 1\import boto3.py'
Creating table...
Table created and active.
Wrote item 0
Wrote item 1
Wrote item 2

28C Partly sunny
```

```

import boto3
import time

def write_items():
    item = {
        'Id': {'S': str(i)},
        'Attribute': {'S': f'Value{i}'}
    }
    client.put_item(TableName=table_name, Item=item)
    print(f'Wrote item {i}')
    # Optional: slight delay to modulate speed
    # time.sleep(0.1)

if __name__ == '__main__':
    create_table()
    write_items()

```

TERMINAL

```

\dynamodb\launcher' '61162' '--' 'c:\Users\cheta\Desktop\DE Homework\dynamodb\ sdk - rcu 1 wru 1\import boto3.py'
Creating table...
Table created and active.
Wrote item 0
Wrote item 1
Wrote item 2
Wrote item 3
Wrote item 4

```

DynamoDB > Tables

**Tables (1) Info**

Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read
table-low-capacity	Active	Id (\$)	-	0	0	Off		

Actions ▾ Delete Create table

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Id (String)	Value	Attribute	Actions	Create item
2	Value2			
8	Value8			
9	Value9			
1	Value1			
6	Value6			
0	Value0			
5	Value5			
4	Value4			
7	Value7			
3	Value3			

## Task 4: Cost Calculation Exercise for Provisioned Throughput

Given workload:

- Writes: 500/sec, item size 2.3 KB
- Reads: 3000/sec (eventual consistency), item size 5.1 KB

Calculations:

- **Write Capacity Units (WCUs):**
  - 1 WCU = 1 write per second for up to 1 KB
  - Each write is 2.3 KB ≈ 3 WCUs per write
  - Total WCU = 500 writes/sec \* 3 = 1500 WCUs
- **Read Capacity Units (RCUs):**
  - 1 RCU = 1 strongly consistent or 2 eventually consistent reads per second for up to 4 KB
  - 5.1 KB item > 4 KB, use  $\text{ceil}(5.1/4) = 2$  RCUs per read

- Eventual consistency halves RCU usage (1 RCU supports 2 reads)
- Total RCU = (3000 reads / 2) \* 2 = 3000 RCUs

The screenshot shows the 'Create table' wizard in the AWS DynamoDB console. In the 'Choose table class' step, 'DynamoDB Standard' is selected. The 'Capacity calculator' section shows inputs: Average item size (1 KB), Item read/second (3000), Item write/second (500). Read consistency is set to 'Eventually consistent' and Write consistency to 'Standard'. The resulting configuration is: Read capacity units (1,500), Write capacity units (500), Region (ap-south-1), and Estimated cost (\$550.56 / month).

## Task 5: Explore Autoscaling / On-Demand Mode

- Create a new DynamoDB table with **On-Demand** capacity mode (no manual RCU/WCU settings).
- Run rapid write tests as in Task 3 script.
- Observe no throttling occurs unless hitting overall account limits.
- Understand that On-Demand mode automatically adjusts capacity instantly to match load.
- Consider pricing implications:
  - On-Demand is more expensive but offers flexibility.
  - Provisioned mode with Auto Scaling offers cost savings but slower capacity adjustment.

**CODE:**

```
import boto3
# Initialize DynamoDB client with your preferred region
client = boto3.client('dynamodb', region_name='ap-south-1') # Change region if needed

table_name = 'table-on-demand'

# Step 2: Write 10 items rapidly to the table
def write_items():
    for i in range(10):
        item = {
            'Id': {'S': str(i)},
            'Attribute': {'S': f'Value{i}'}
        }
        client.put_item(TableName=table_name, Item=item)
        print(f'Wrote item {i}')
    # Optional: slight delay to modulate speed
    # time.sleep(0.1)

if __name__ == '__main__':
    write_items()
```

## SCREENSHOTS:

```
C:\Windows\System32>aws dynamodb create-table --table-name table-on-demand --attribute-definitions AttributeName=Id,AttributeType=S --key-schema AttributeName=Id,KeyType=HASH --billing-mode PAY_PER_REQUEST
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Id",
                "AttributeType": "S"
            }
        ],
        "TableName": "table-on-demand",
        "KeySchema": [
            {
                "AttributeName": "Id",
                "KeyType": "HASH"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2025-08-25T18:48:56.831000+05:30",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 0,
            "WriteCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:ap-south-1:845958739988:table/table-on-demand",
        "TableId": "725d183b-49c5-489c-9452-2a5c60e41a48",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        },
        "DeletionProtectionEnabled": false
    }
}

C:\Windows\System32>
```

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows a file named "import boto3.py" and an untitled document.
- Search Bar:** Contains the text "Search".
- Code Editor:** Displays the content of "import boto3.py". The code initializes a DynamoDB client, defines a function to write 10 items to a table, and includes a loop to print each item's ID and attribute.
- Terminal:** Shows command-line output for running the script. It includes environment variables like PS, PATH, and PYTHONHOME, and the command "python import boto3.py". The terminal also displays the printed output of the script, which shows the IDs and attributes of the 10 written items.
- Bottom Status Bar:** Shows the current file is "import boto3.py", line 24, column 5, with 3.1.4 Python selected as the language.

The screenshot shows a code editor window in VS Code with a Python file named `boto3.py`. The code defines a function `write_items` that inserts 10 items into a table. It uses the `boto3` library's `client.put_item` method. A terminal tab below shows the command to run the script and its output, which shows three items inserted successfully.

```

import boto3.py
C: > Users > cheta > Desktop > DE Homework > dynamodb > sdk - rru 1 wru 1 > import boto3.py > ...
7
8
9
10
11  # Step 2: Write 10 items rapidly to the table
12 def write_items():
13     for i in range(10):
14         item = {
15             'Id': str(i),
16             'Attribute': {'S': f'Value{i}'}
17         }
18         client.put_item(TableName=table_name, Item=item)
19         print(f'Wrote item {i}')
20         # Optional: slight delay to modulate speed
21         # time.sleep(0.1)
22
23 if __name__ == '__main__':
24     write_items()
25
26
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\cheta\Desktop\DE Homework\dynamodb\sdk - rru 1 wru 1>
PS C:\Users\cheta\Desktop\DE Homework\dynamodb\sdk - rru 1 wru 1> cd 'c:\Users\cheta\Desktop\DE Homework\dynamodb\sdk - rru 1 wru 1'; & 'c:\Users\cheta\AppData\Local\Programs\Python313\python.exe' 'c:\Users\cheta\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '54256' '--' 'c:\Users\cheta\Desktop\DE Homework\dynamodb\sdk - rru 1 wru 1\import boto3.py'
Wrote item 0
Wrote item 1
Wrote item 2
Wrote item 3
0 0 28°C Partly sunny
25-08-2025

```

The screenshot shows the AWS DynamoDB console. On the left, a navigation sidebar includes links for Dashboard, Tables, DAX, and CloudShell. The main area displays a table titled "Tables (1) Info" with one item: "table-on-demand". The table has columns for Name, Status, Partition key, Sort key, Indexes, Replication Regions, Deletion protection, Favorite, and Read capacity.

	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacity
	table-on-demand	Active	Id (\$)	-	0	0	Off		On-demand

The screenshot shows the AWS DynamoDB console interface. On the left, there's a navigation sidebar with options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Below that is a section for DAX with Clusters, Subnet groups, Parameter groups, and Events. The main content area is titled "Table: table-on-demand - Items returned (10)". It shows a table of items with two columns: "Id (String)" and "Attribute". The items are listed as follows:

	Id (String)	Attribute
2	Value2	
8	Value8	
9	Value9	
1	Value1	
6	Value6	
0	Value0	
5	Value5	
4	Value4	
7	Value7	
3	Value3	

At the bottom of the main window, there are buttons for Actions and Create item. The status bar at the bottom right shows the date (25-08-2025), time (18:55), and location (India). The taskbar at the very bottom includes icons for CloudShell, Feedback, Search, and various system applications.