Deploying and querying a DynamoDB table

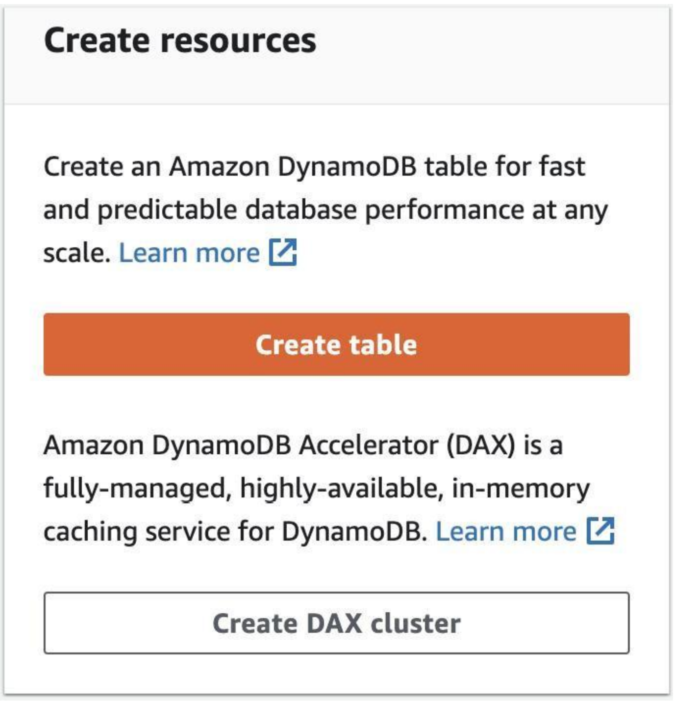
Now that we've learned about DynamoDB and its features, let's implement our own DynamoDB table to practice using the console and API. We'll use the GameScores table to build a simple ranking database. We'll use both the AWS console and the AWS CLI for these steps.

Provisioning a DynamoDB Table

We'll start by provisioning a DynamoDB table. We'll be using the **Oregon region:**

Open the AWS console in an Internet browser and sign in with an account that has privileges to create and modify a DynamoDB table.

Navigate to the DynamoDB section. Click the orange **Create Table** button on the right side of the screen:



Create resources

Create an Amazon DynamoDB table for fast and predictable database performance at any scale. Learn more

Amazon DynamoDB Accelerator (DAX) is a fully-managed, highly-available, in-memory caching service for DynamoDB. Learn more

This will open the **Create Table** page, allowing us to enter our table details. Choose the following options. Any options not mentioned here, leave them as default:

- **Table name**: GameScores.
- **Partition key**: PlayerID.
- **Sort key**: GameName.
- **Settings**: Customize your settings.
- **Read/write capacity settings**: on-demand.
- **Secondary indexes**: Leave blank.
- **Encryption at rest**: Owned by Amazon DynamoDB.
- Click **Create table**.

Note that we don't add any attributes except partition and sort keys, as they are only added when new records are created in the table due to the capability of DynamoDB. to handle unstructured data.

The table is created immediately. We can use the AWS CLI to check the status of the table:

```
aws dynamodb describe-table –table-name GameScore
```

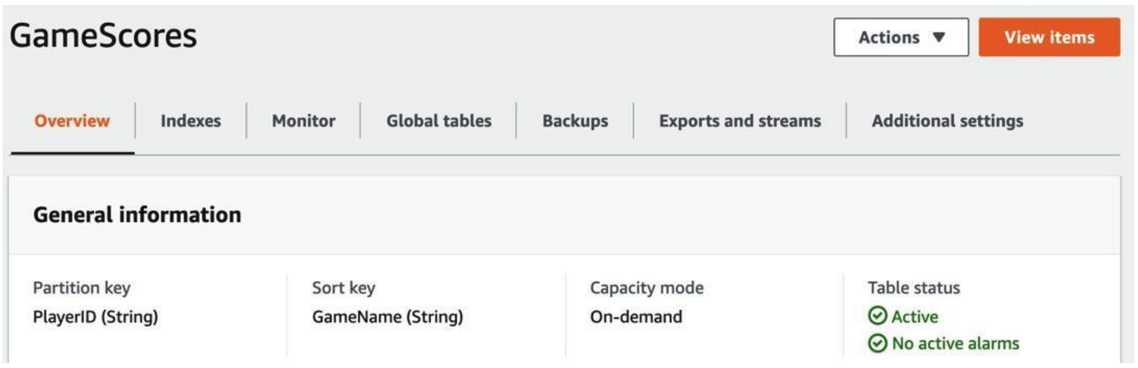
Now we've created our table called GameScores and we're ready to add some elements.

Add Items

Now we have our table empty, we can add some elements. We can use the API to do this or we can use the console. Let's start with the console:

From the DynamoDB dashboard, click **Tables** on the left side and click the GameScores table.

Click **View items** at the top left of the screen:



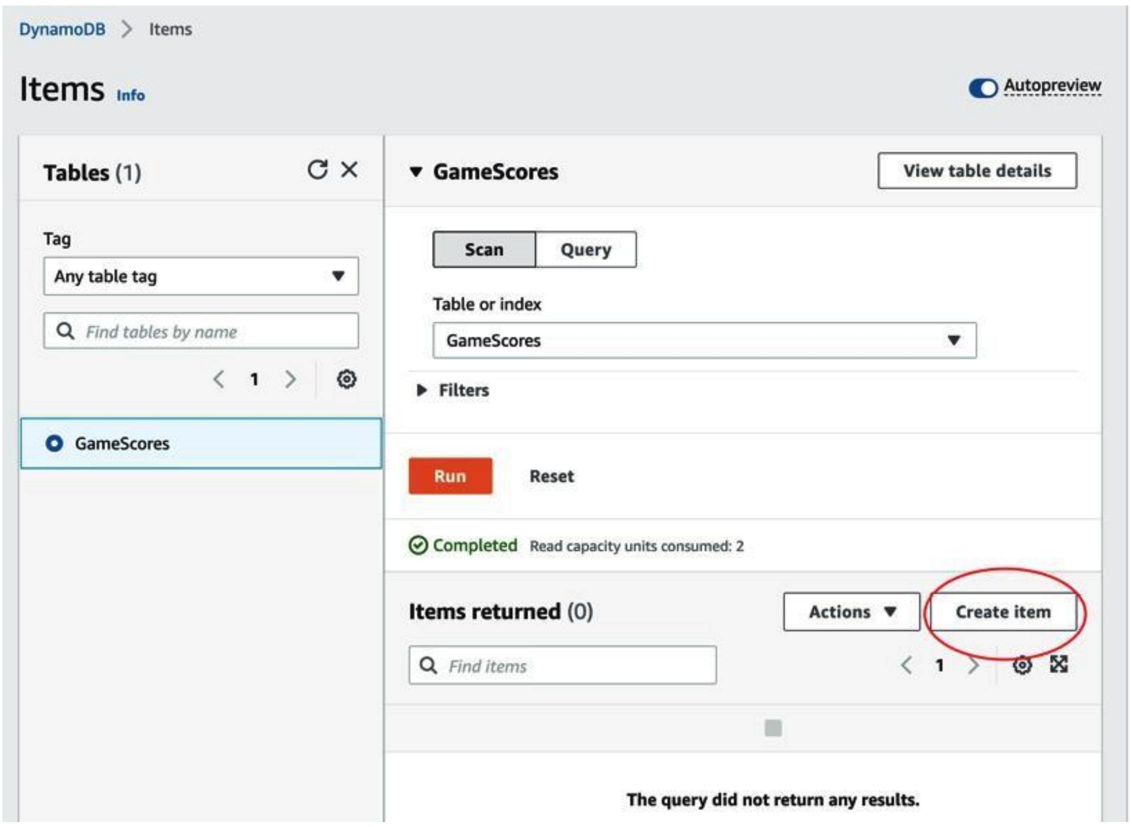
Partition key
PlayerID (String)

Sort key
GameName (String)

Capacity mode
On-demand

Table status
Active
No active alarms

This opens a page that allows you to run queries on the DynamoDB table, but we need to add a few items first. Click **Create Item** as shown in the following screenshot:



Completed Read capacity units consumed: 2

The query did not return any results.

Add a few items to the table; we'll need to add both PlayerID and GameName. We'll also add an attribute at this point. Click on the menu Drop-down **Add New attribute and select Number**. We'll call this HighScore. Complete all three fields, and then click **Add Item** . If we wish, we can repeat this step to add more records.

Once this is done, we can see the elements created in the table. Note the message in the middle of the page that reads as follows:

Consult and scan the table

Now that we have a couple of records in our table, we can start running a query and scanning to see how they behave.

We're going to use the API to run the query. Here, we are going to check the JohnS score in the game *Bug Hunt* change it based on the records that have been added in the previous step:

```
AWS Dynamodb
   --table-name GameScores \
   --key-condition-expression  "PlayerID  =  :playerid  AND  GameName      =
:gamename"\
   --expression-attribute-values '{
      ":playerid": { "S": "JohnS" },
      ":gamename": { "S": "Bug Hunt" }
   }'
   --projection-expression "PlayerID, GameName, HighScore"
```

Note that we can only query items that are part of the key, but we can filter by other attributes if they are returned in the original dataset. So, for example, we can add an additional filter to get only the scores greater than 1,000:

```
AWS Dynamodb
Query\
   --table-name GameScores
   --key-condition-expression  "PlayerID  =  :playerid  AND  GameName      =
:gamename"
   --expression-attribute-values '{
      ":playerid": { "S": "JohnS" }
```

```
        ":gamename": { "S": "Bug Hunt" }
    }'
    --filter-expression "HighScore > :score"\
    --projection-expression "PlayerID, GameName, HighScore"
```

Now we've learned how to use queries to quickly retrieve records using the partition key, so now let's learn how to use scan commands to retrieve records using columns other than the partition key.

We want to get to know all the players. who got more than 100, but since the HighScore attribute is not part of the key, we can't get it through a query, so we need to use a scan. Remember that scans involve checking every item in the table, so it can be time-consuming and expensive, and should be avoided if possible:

```
AWS Dynamodb
Scan \
     --table-name GameScores
     --filter-expression "HighScore > :score"
     --projection-expression "PlayerID, GameName" \
     --expression-attribute-values '{":score":{"N":"1000"}}'
```

A better way to make this type of query would be to add a GSI and use it instead of relying on a scan. Let's do that now.

In the previous section, we had to use a scan to get the data we wanted since HighScore wasn't part of our key. If we add a new GSI to the table, we can convert it to a key.

First, think about how we want the data to be returned. Since we're storing data for multiple games, it's unlikely that we want our query to return all the scores, but rather the scores for a specific game, so we need to create a GSI with the GameName partition key and HighScore as the sort key.

The benefit of doing this is that the scores are stored in order, so if we want to get an instance of HighScore, we can simply get the first key:

Use the API (or console if you prefer) to create a new GSI:

```
AWS Dynamodb update-
        --table-name GameScore \
        --attribute-definitions '[
{AttributeName=GameName,AttributeType=S},
{AttributeName=HighScore,AttributeType=N}
]'\
        --global-secondary-index-updates '[
{"Create": {"IndexName": "GameTitleScores","KeySchema":[
{"AttributeName":"GameName","KeyType":"HASH"},":{"AttributeName":"Hi
ghScore","KeyType":"RANGE"}],
"Projection":{"ProjectionType":"ALL"}}
        ]'
```

Now that we have the new index, we can run a query to get the highest score for our game; unlike the case with a SQL database, we need to specifically tell DynamoDB which index to use. Remember that the HighScore attribute is sorted, so we can only get the first value returned using the no-scan-index-forward setting to reverse the order:

```
AWS Dynamodb
    --table-name GameScores \
    --index-name GameTitleScores
    --key-condition-expression "GameName = :game\
    --expression-attribute-values '{
        ":gamename": { "S": "Bug Hunt" }
    --projection-expression "PlayerID, HighScore" \
    --no-scan-index-forward
    --max-items 1
                    }'
```

Now we've learned how to create a DynamoDB table, insert items, query them, and use a scan.