June-22-2025

Unity - Rendering a Global LeaderBoard Using AWS DynamoDB Integration

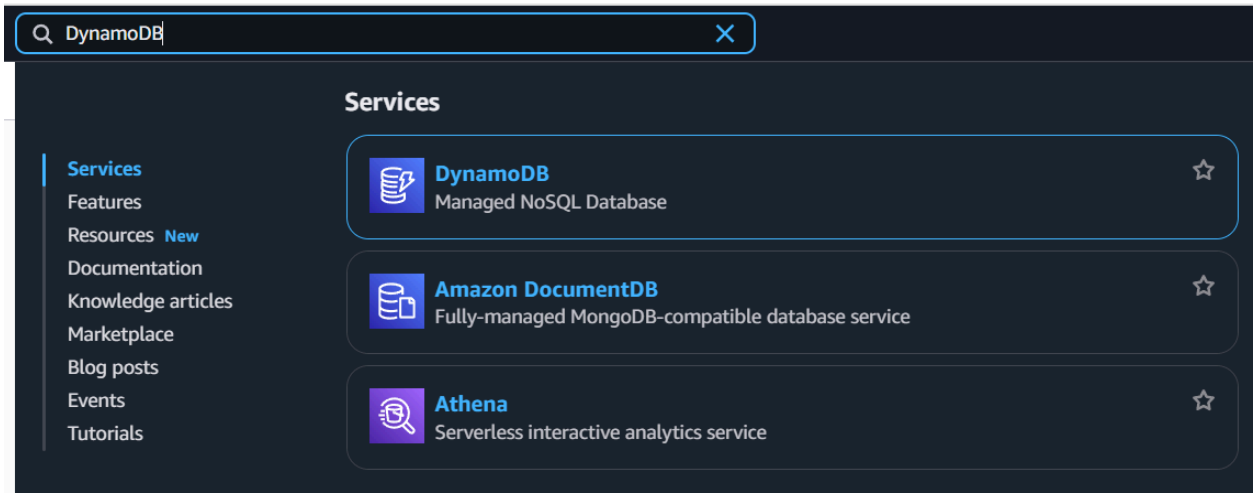
By: Adi Barda (scenemax3d@gmail.com)

AWS (Amazon Web Services) is giving 25GB of free for life storage on its DynamoDB database with ~20,000,000 free API calls per month.
In this manual I will give instructions on how to make use of that storage in Unity game engine.

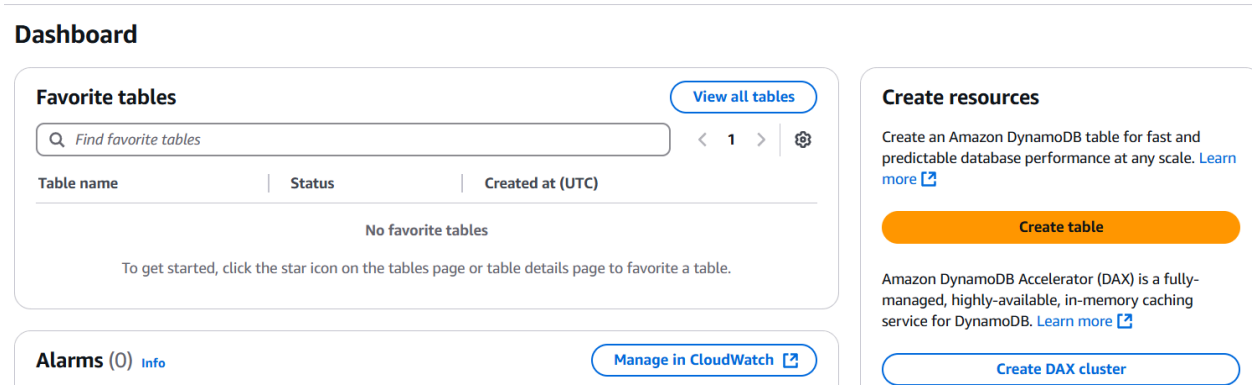
Step 1 - Create The DynamoDB table

1. Create an AWS account if you don't already have one
   Navigate to: https://aws.amazon.com/

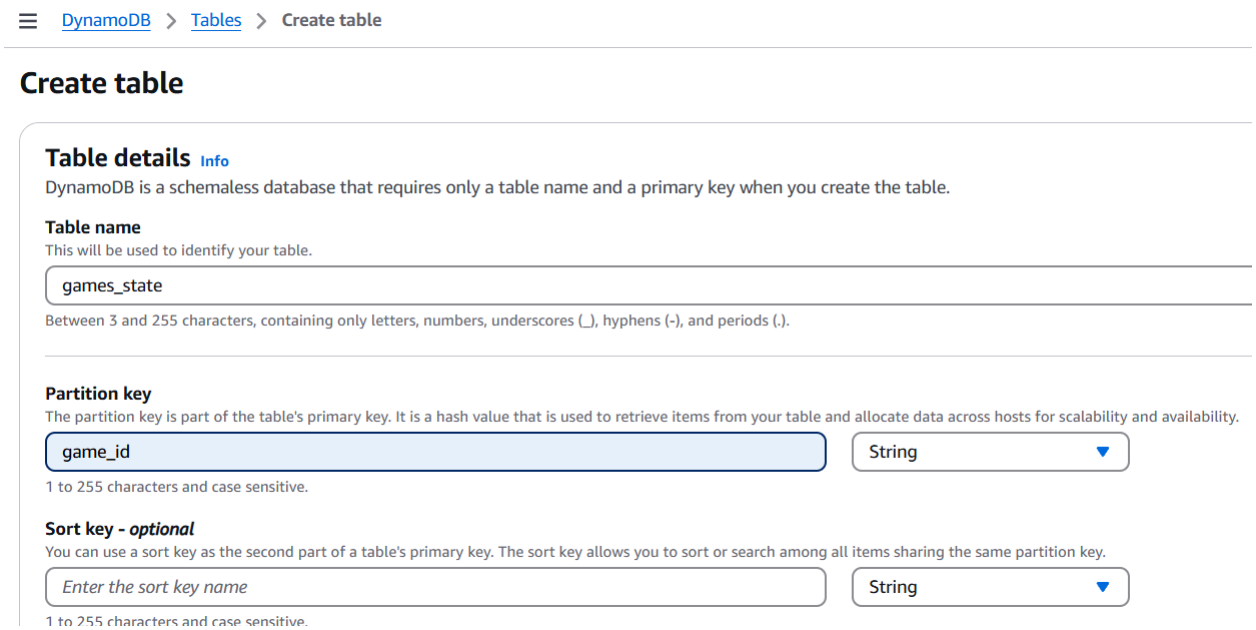
2. In your AWS account
   Search for **DynamoDB** in the search box and click on "DynamoDB" service



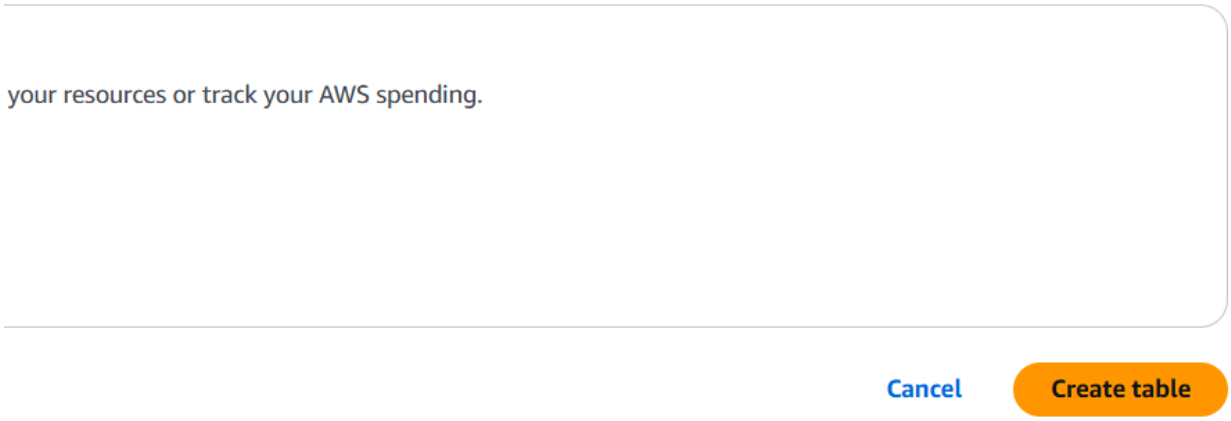
3. Click on "**Create Table**" button in the right side of the dashboard



4. In the Create Table form, enter a table name and a partition key.
   Use table name: games_state and partition key: game_id
   (you can use different names for the table and partition key but those values will make it easier for you to use the Unity code later on in this manual)



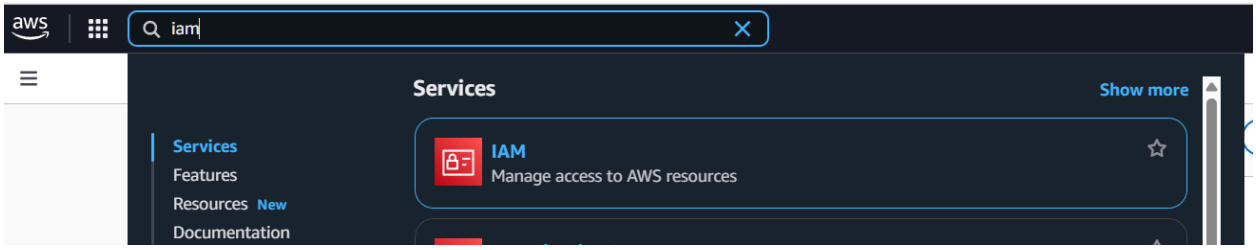
5. Scroll down to the bottom and press on "**Create Table**" button:



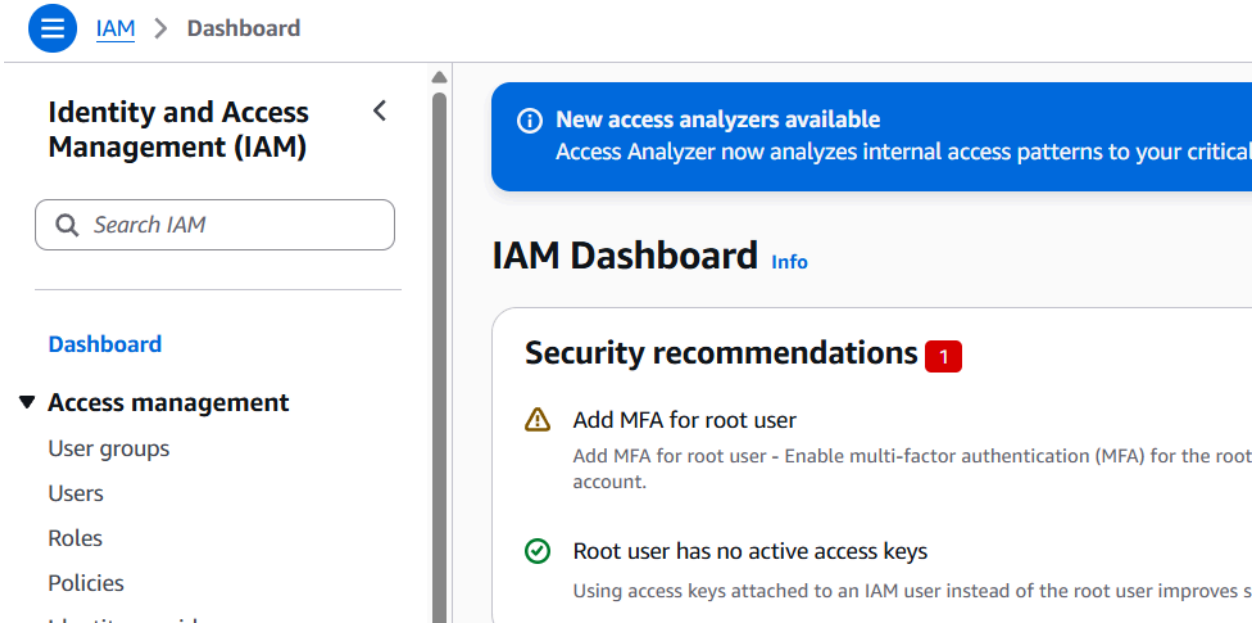
Congratulations! you just created your free DynamoDB storage

Step 2 - Create a user for accessing the DynamoDB storage

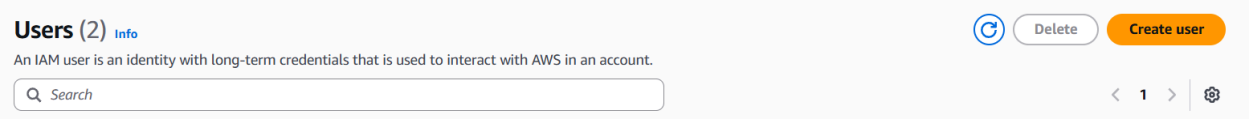
1. Type “iam” in the services search box and click on the “IAM” service



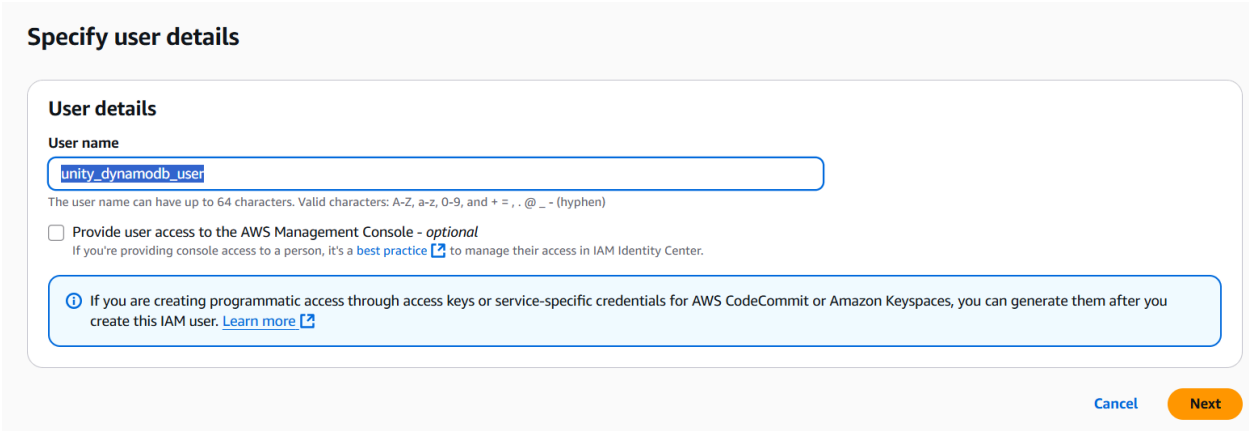
2. In the IAM Dashboard window, click on “**Users**” in the left navigation menu under the “**Access management**” section



3. In the “users” window, click on “Create user” button:

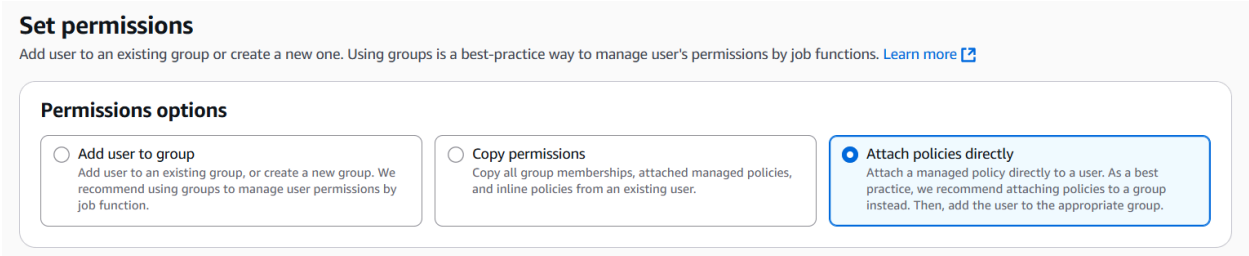


4. Give your user a name. It’s going to access your game’s data so it’s best give it a meaningful name for example “unity_dynamodb_user” but you can give it any name you like.

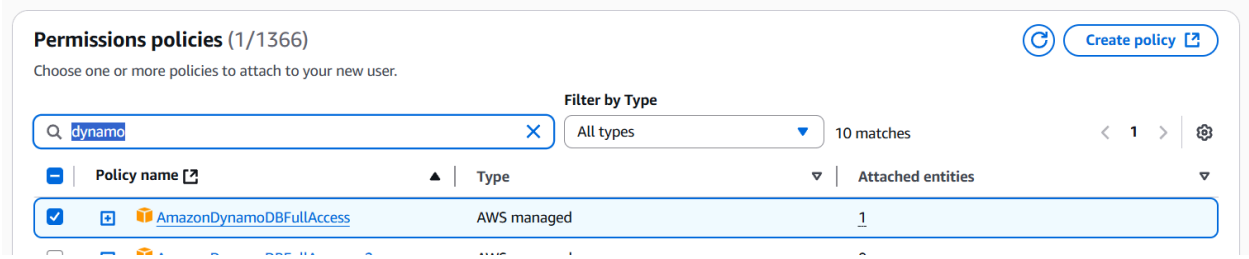


Click “Next”

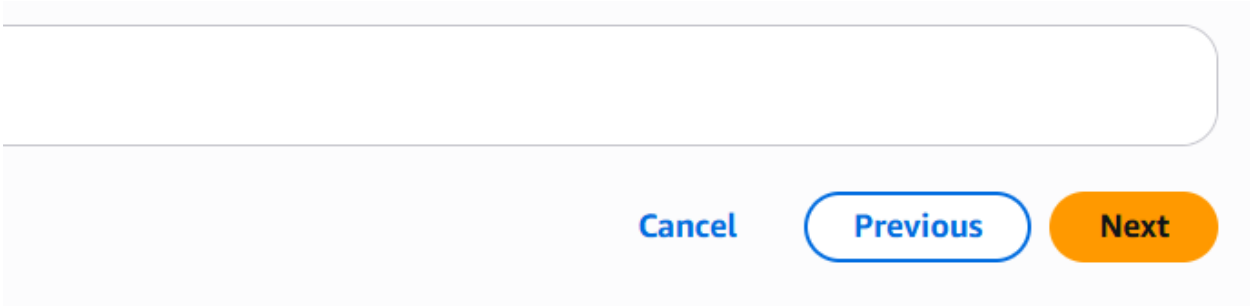
5. In the “Set permissions” window, select the “**Attach policies directly**” option



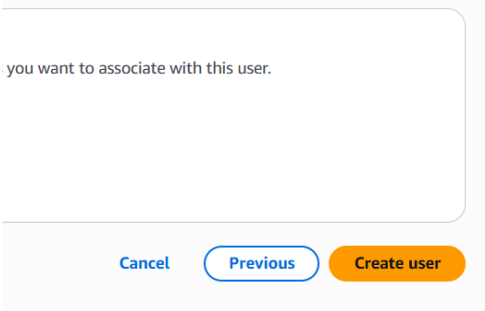
Now, in the **Permissions policies** section, type “**dynamo**” in the search box and select the “**AmazonDynamoDBFullAccess**” policy.



Scroll down to the bottom of the page and click “Next”



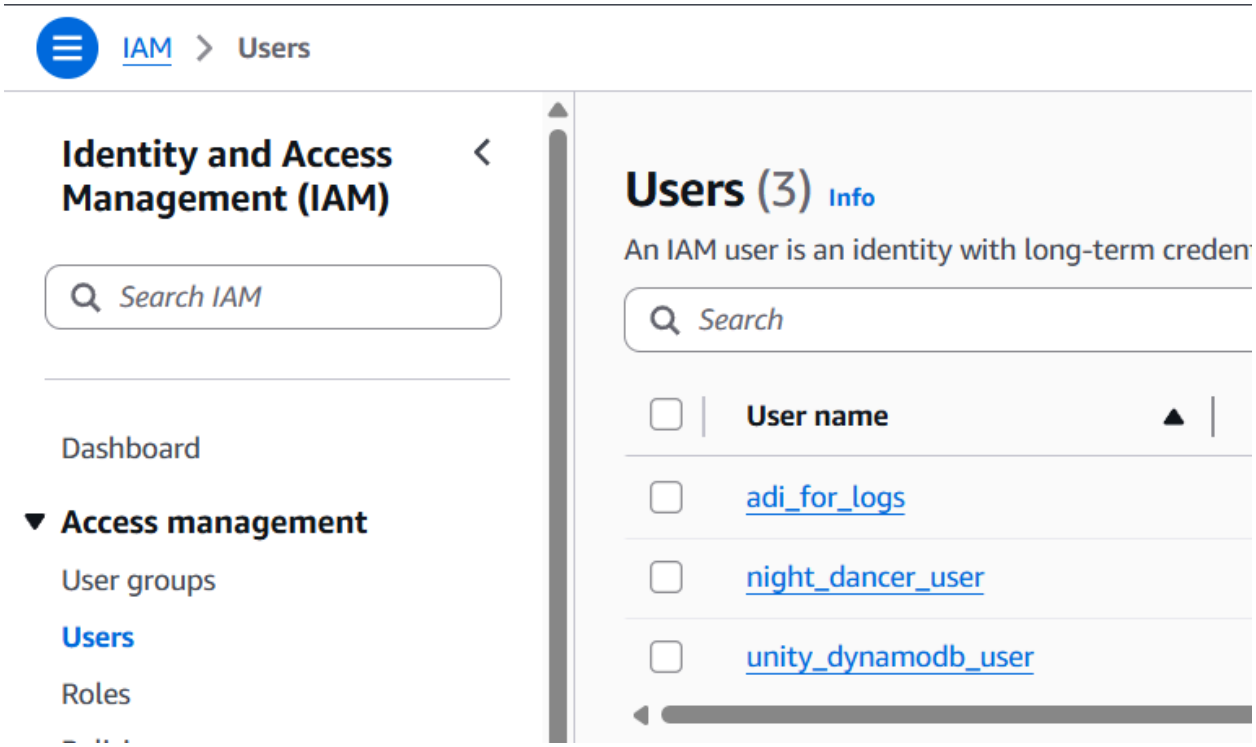
6. In the final “**Review and create**” page, click on “**Create user**” button



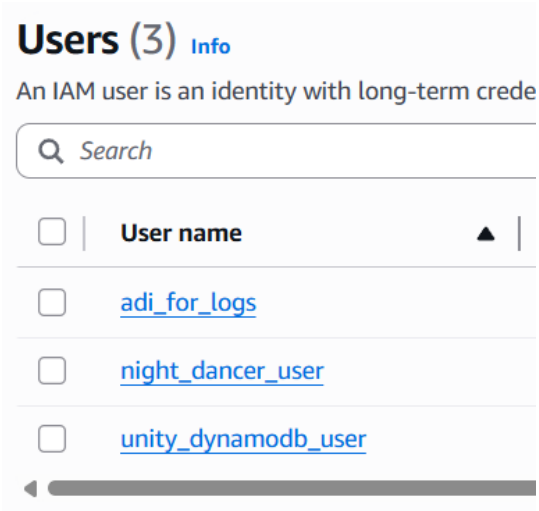
Step 3 - Create Access Keys

we will need to create access keys for your newly created user for authenticating and accessing your database from within Unity so let’s create them.

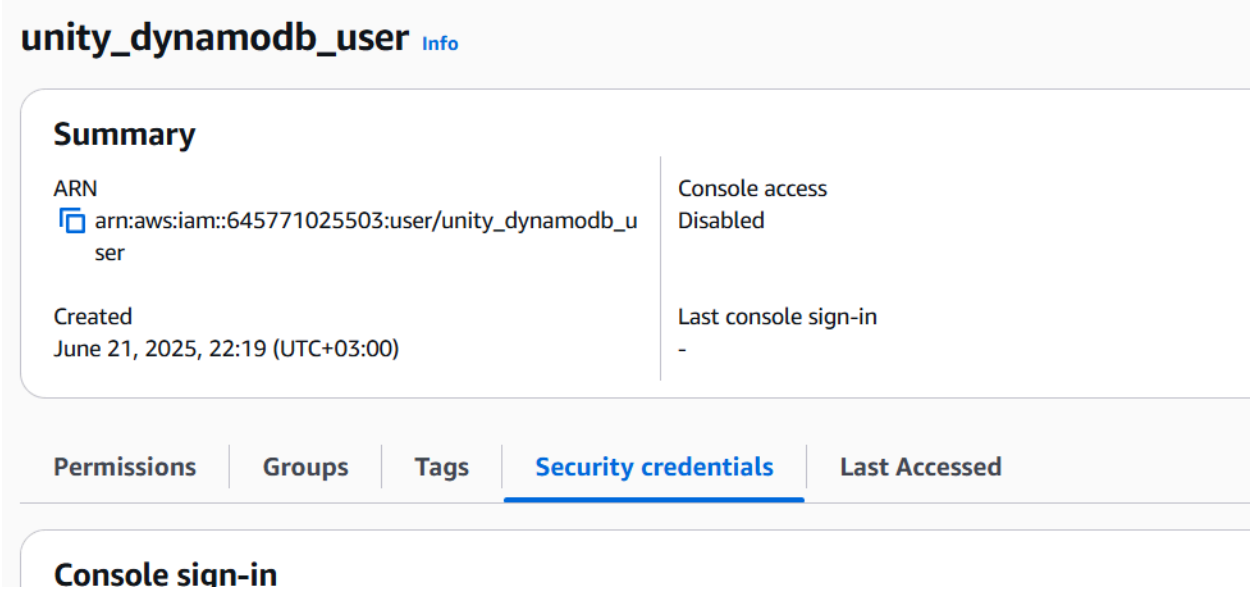
1. Return to the “IAM” service and click on “Users” - you will see your new “unity_dynamodb_users” in the users list



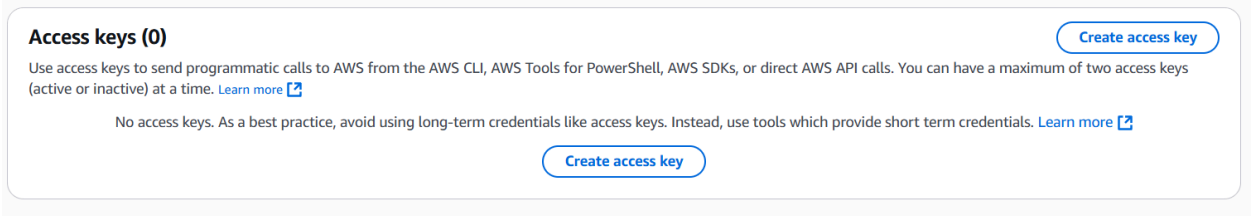
2. Click on the “**unity_dynamodb_user**”



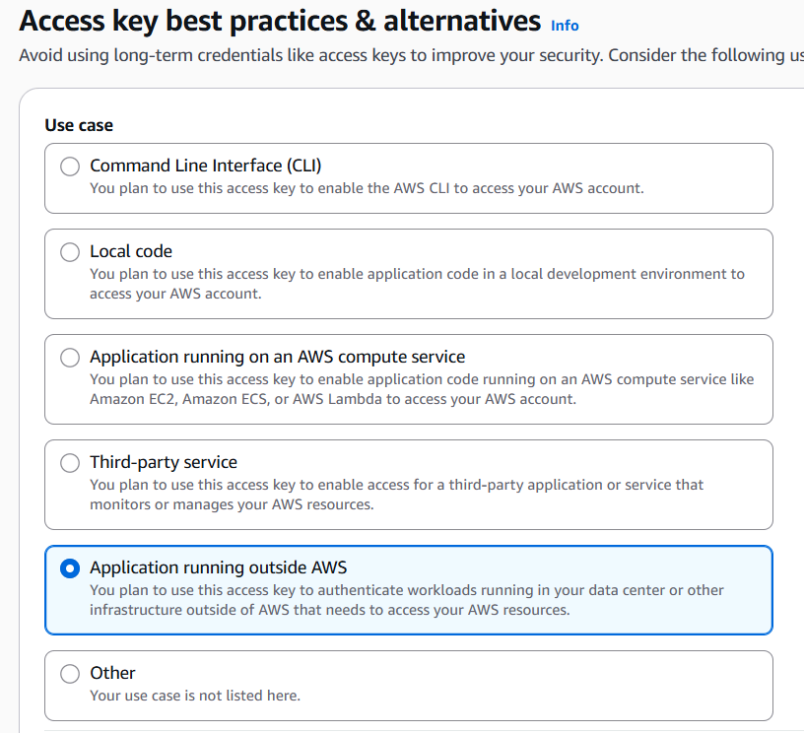
3. In the unity_dynamodb_user page, click on “**Security credentials**” tab



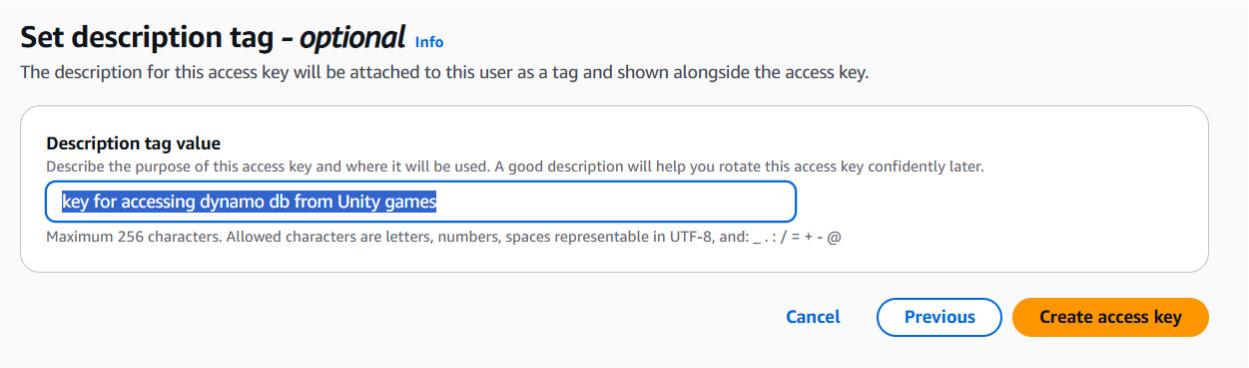
4. Scroll down to the “Access keys” section and click on “**Create access key**” button



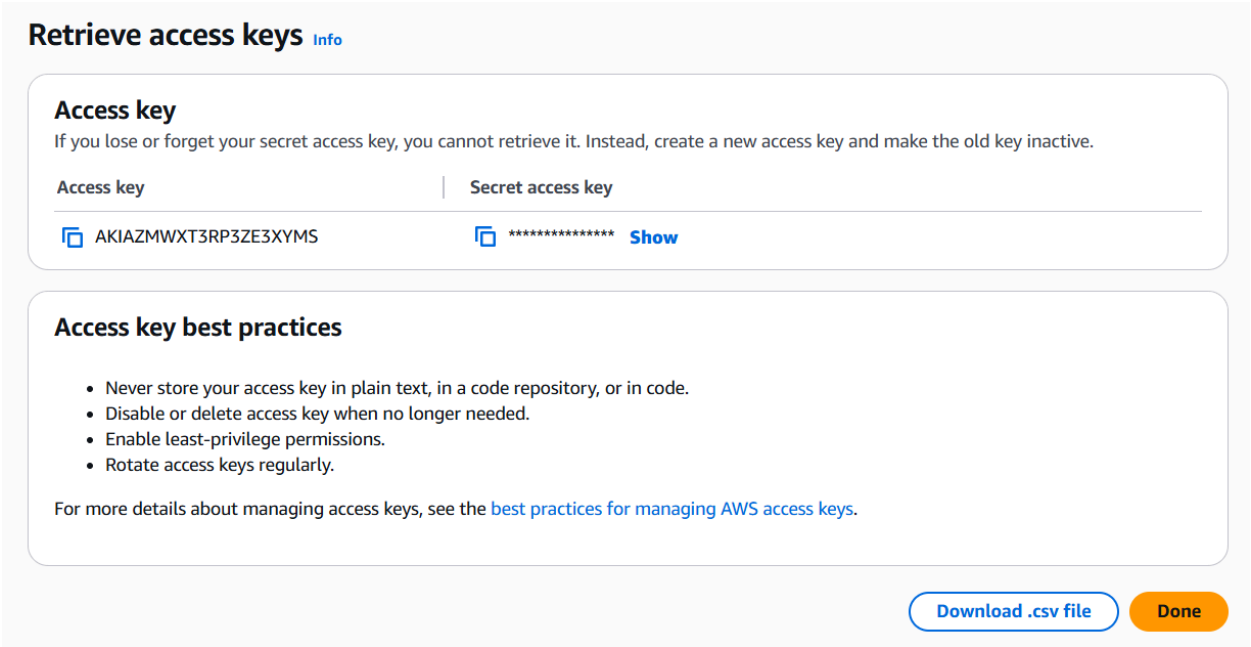
5. Select “**Application running outside AWS**” option and click the “**Next**” button



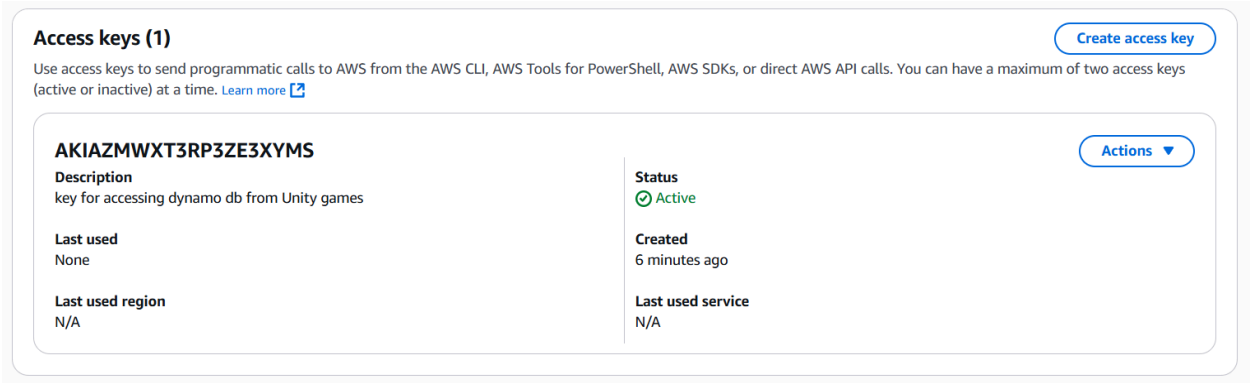
6. Type a meaningful description for your key and click “Create access key”



7. Access and secret keys created for you, copy - paste them to a safe place.
   <u>This is your only chance to copy them</u>. you will not be able to copy them after leaving this page so do it now.
   If you lose those Access & secret keys you will need to create a new pair of access-secret keys.



After copying the access & secret keys click on the "**Done**" button
You can see now your newly created access key in the access keys section of your user:



Now for the fun part 😀let's use our database in Unity 💪

Step 4 - Connect your Unity game to your DynamoDB database

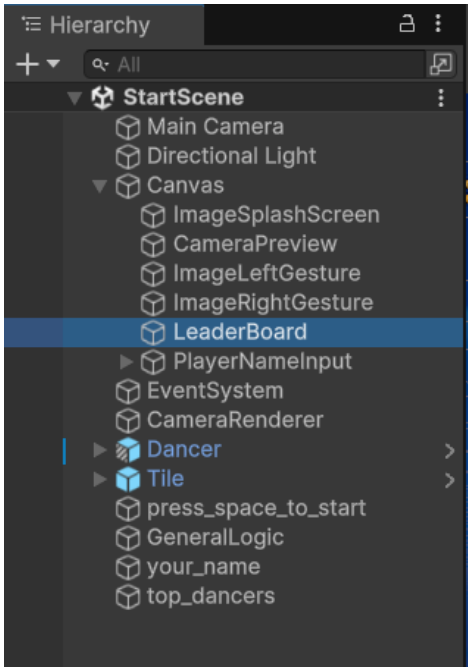
Download and add the AwsGameStatePersistor.cs C# script to your Unity game. Easiest way is to simply drag the file to the assets section in Unity.
The file can be found here:
https://github.com/scenemax3d/unity-aws-dynamodb-integration/blob/main/AwsGameStatePersistor.cs

Open the file in your code editor (e.g. Visual Studio)
Find the **accessKey** and **secretKey** variable definitions and replace the place-holder texts with your actual access key and secret key that you created in the previous section.

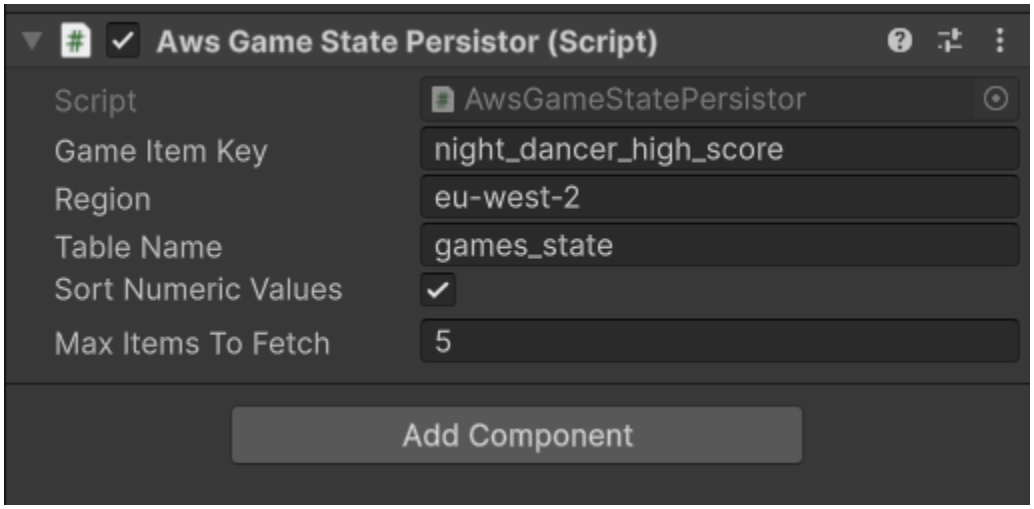
```
private string accessKey = "your aws access key goes here";
private string secretKey = "your aws secret key goes here";
```

Example Of Rendering A Leader Board Table In Unity

Assuming you have a Canvas for your game's UI (if not, add one)
Add an empty game object to the canvas - call it "**LeaderBoard**"



Add an **AwsGameStatePersistor** C# script to the **LeaderBoard** game object.



In the "**Game Item Key"** field enter the purpose of this game state. For example I called it "**night_dancer_high_score**" because this item will be used for storing the leader board (global high score) for my Night Dancer game. You should put a meaningful item name for your game.

In the "**Region**" field, enter the AWS region where your DynamoDB table was created. If you followed the instructions and created it in the **London** region then you can leave the default

“eu-west-2” value otherwise you will need to copy the AWS region code name from your AWS console. See table below for a list of AWS regions:

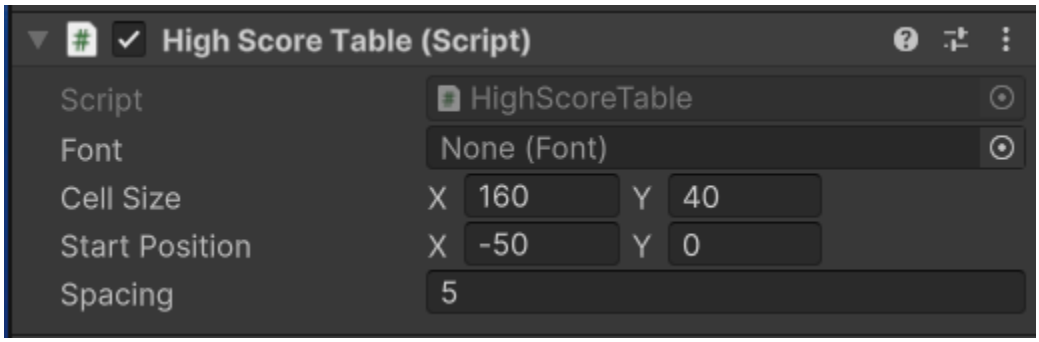
United States	
N. Virginia	us-east-1
Ohio	us-east-2
N. California	us-west-1
Oregon	us-west-2
Asia Pacific	
Mumbai	ap-south-1
Osaka	ap-northeast-3
Seoul	ap-northeast-2
Singapore	ap-southeast-1
Sydney	ap-southeast-2
Tokyo	ap-northeast-1
Canada	
Central	ca-central-1
Europe	
Frankfurt	eu-central-1
Ireland	eu-west-1
London	**eu-west-2**
Paris	eu-west-3
Stockholm	eu-north-1

In the “**Table Name**” field, you can leave the default value “**games_state**” if that’s the name of your DynamoDB table; otherwise, simply change it to your table name.

Check the “**Sort numeric Values**” field if you store numbers in the table (e.g. High score) and want it to be automatically sorted in descending order.

Set the “**Max Items To Fetch**” field to the number of items that you would like to get from the table. usually you will show only a few top items for example Top 10 high scores.

Download a sample High Score Table rendering C# script and add it as well.



That's it. Your leaderboard high score table will be rendered when you start your game.

How To Save The Player’s High Score In Your Global Table

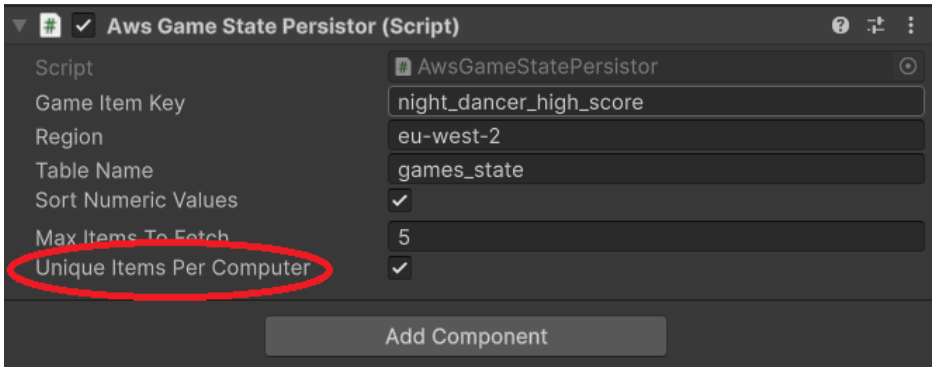
At some point, usually when the game is over, you will like to save the player’s highest score in the global table.
Here is a proposed function to do just that. I assume that you have the player’s name and the current score and you would like to update his high score if needed.

```
    private void SavePlayerHighScore(string playerName, int score)
    {
        AwsGameStatePersistor gp = this.GetComponent<AwsGameStatePersistor>();
        gp.FetchState(onSuccess: data =>
        {
            var playerKey = gp.CreateStateItem(playerName);
            if (!data.ContainsKey(playerKey))
            {
                data[playerKey] = "0";
            }

            var currHighScore = int.Parse(data[playerKey]);
            if (score > currHighScore)
            {
                Debug.Log("Set high score " + Globals.score + " for player: " +
playerName);
                data[playerKey] = score.ToString();
            }

            gp.SaveState(data); // save high score table to cloud storage

        });
    }
```

Unique Items Per Computer

AwsGameStatePersistor supports storing data that is unique to each computer. For example, suppose you're managing a leaderboard—a high-score table of names and scores—and three different users on three different computers around the world all happen to be named Fernando. By default, because they share the same name, only one "Fernando" entry appears in the high-score list, which is fine for many use cases.

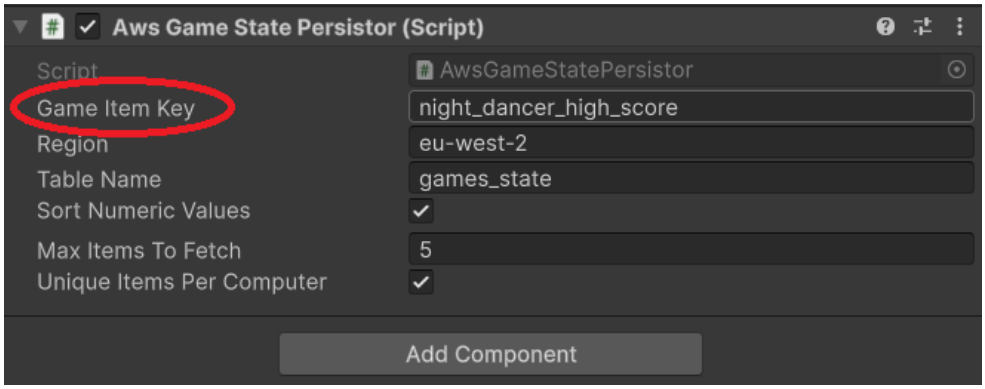
However, if you want to keep keys unique per computer and prevent one computer's data from overwriting another's, you can enable the **Unique Items Per Computer** flag.



System Limitations

AwsGameStatePersistor is fetching and saving data against AWS DynamoDB NoSQL database. As mentioned there is a free for life tier of 25GB storage and ~20 million API requests per month. Those are the limitations of the system. For extremely popular games with a large gamers community these limitations might break and usage charge will be applied.

The AwsGameStatePersistor can handle unlimited game item keys, each serves for a different purpose like leaderboard, game settings etc. and for as many games that you like up to using your free 25GB of storage.



However, each game item key can store up to 440 KB of data so for example your high score table can store about 8000 - 10000 items.