UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Entrega 7

Taller - 2022-1

Introducción a Cloud Computing con AWS

13 de mayo de 2022 - v1.0

Rodrigo Cayazaya Marín | 201773538-4
rodrigo.cayazaya@sansano.usm.cl

Índice

Índice de figuras

Instrucciones

El presente documento corresponde a una plantilla que incluye las informaciones que deben ser proveídas para evaluar la entrega.

La entrega se basa mayormente en capturas de pantalla de la consola, la mayoría de ellas vistas en clase más algunas acciones adicionales que deben ser descubiertas por cada estudiante.

Todos los textos en rojo a lo largo de la plantilla, junto con esta página de instrucciones, deben ser eliminadas antes de la compilación final que debe ser entregada por Moodle.

1. Creación de tabla

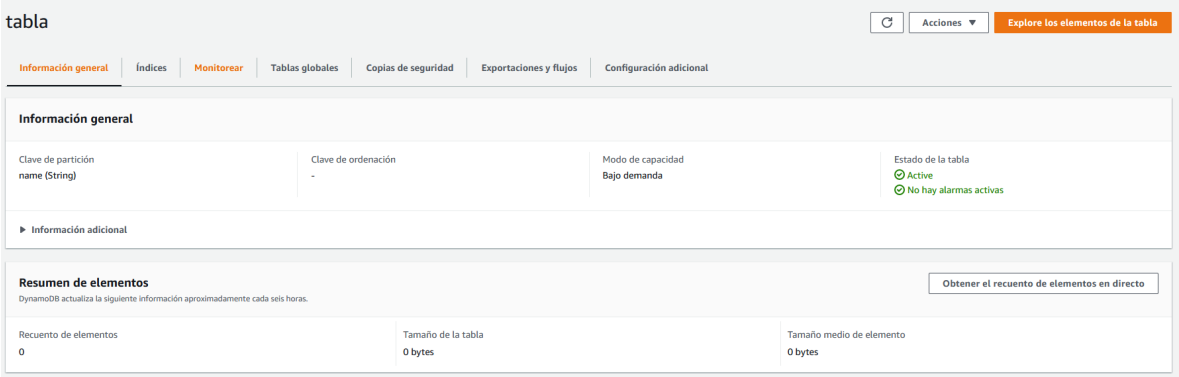

Figura 1: Captura de pantalla de la consola de DynamoDB con la vista Overview de la tabla creada

2. Creación de item

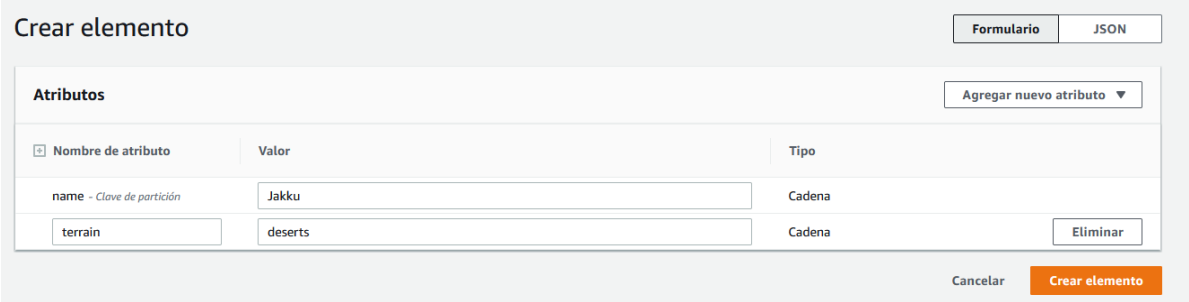

Figura 2: Captura de pantalla mostrando el item a crear en el formulario

La representación del item a ingresar en DynamoDB JSON corresponde a:

```
{
  {
  "name": {
    "S": "Jakku"
  },
  "terrain": {
    "S": "deserts"
  }
}
}
```

Figura 3: Captura de pantalla mostrando la tabla con el nuevo item creado

3. Carga de items

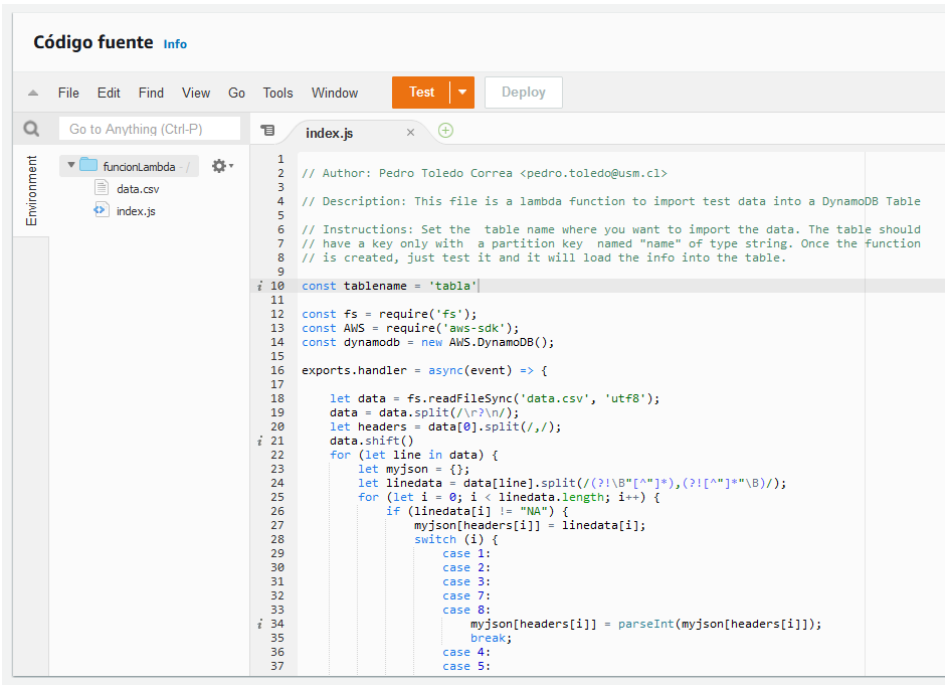

Figura 4: Captura de pantalla mostrando el código de la función de carga de datos

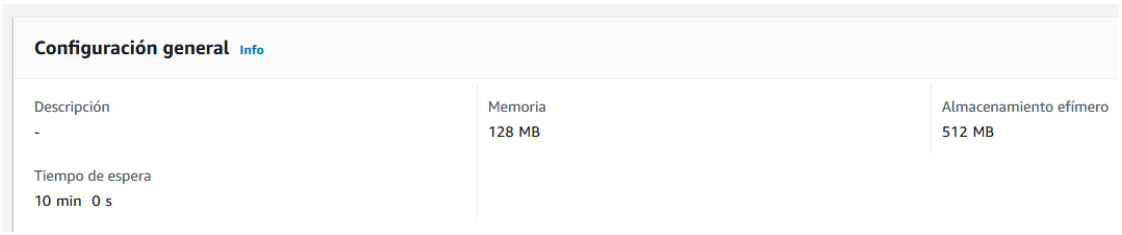

Figura 5: Captura de pantalla mostrando “General Configuration” de la función de carga de datos

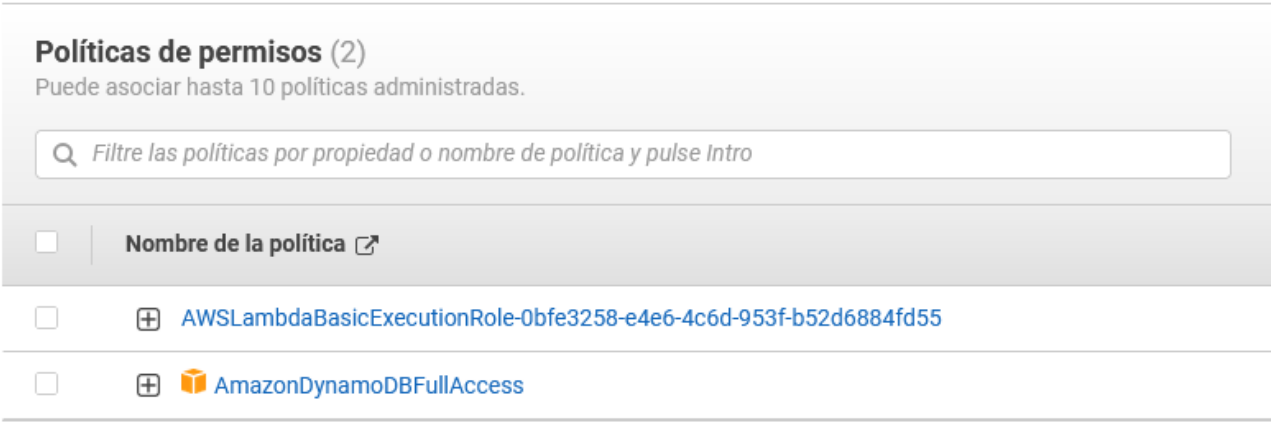

Figura 6: Captura de pantalla mostrando los permisos de ejecución del rol de la función lambda

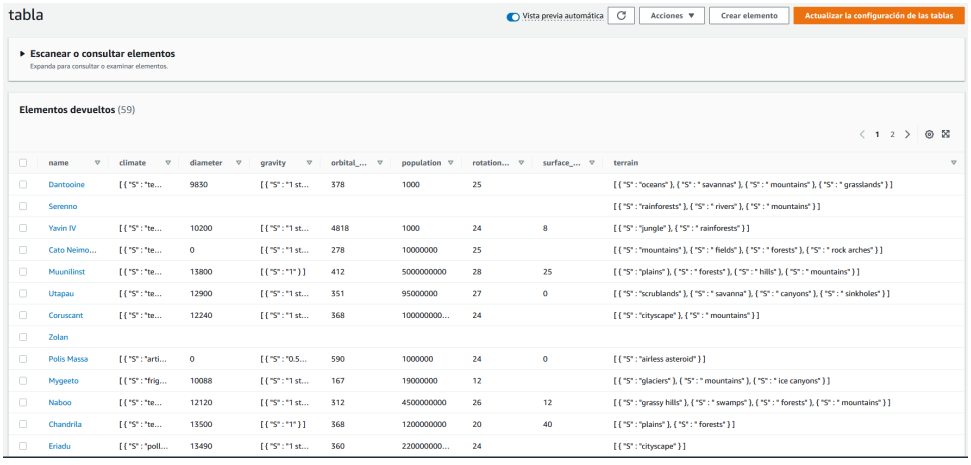

Figura 7: Captura de pantalla mostrando la lista de items contenidos en la tabla luego de la ejecución del test

4. Operaciones sobre tabla

El siguiente es el código y resultado de una Query para obtener el item con llave de partición “Coruscant”

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'sa-east-1'});

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB();

exports.handler = async ( event ) => {
    var params = {
      TableName: 'tabla',
      Key: {
        'name': {S: 'Coruscant'}
      }
    };

    // Call DynamoDB to read the item from the table
    await ddb.getItem(params, function(err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data.Item);
      }
    }). promise();
};
```

```
2022-05-10T21:03:14.161Z  489fc4fe-f085-4f5e-9221-7b734ad88b70  INFO
     Success {
  diameter: { N: '12240' },
  rotation_period: { N: '24' },
  gravity: { L: [ [Object] ] },
  climate: { L: [ [Object] ] },
  orbital_period: { N: '368' },
  population: { N: '1000000000000' },
  name: { S: 'Coruscant' },
  terrain: { L: [ [Object], [Object] ] }
}
```

El siguiente código es el utilizado para agregar un nuevo item con llave de partición “Hoth”:

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'sa-east-1'});

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB();
```

```
8
9  exports.handler = async ( event ) => {
10     var params = {
11       TableName: 'tabla',
12       Item: {
13         'name' : {S: 'Hoth'},
14         'diameter' : {N: '7200'},
15         'rotation_period' : {N: '23'},
16         'gravity' : {L: [{
17                     S: "1.1 standard"
18                     }]},
19         'climate' : {L: [{
20                     S: "frozen"
21                     }]},
22         'surface_water' : {N: '100'},
23         'orbital_period' : {N: '549'},
24         'terrain' : {L: [
25                     {S: "tundra"},
26                     {S: "ice caves"},
27                     {S: "mountain ranges"}
28                     ]},
29     }
30     };
31
32     // Call DynamoDB to read the item from the table
33     await ddb.putItem(params, function(err, data) {
34     if (err) {
35         console.log("Error", err);
36     } else {
37         console.log("Success", data);
38     }
39     }). promise();
40 };
```

El siguiente código y resultado muestra un Query para obtener el nombre y población de todos los planetas en la lista que tienen periodo orbital entre 500 y 1000 días:

```
1  // Load the AWS SDK for Node.js
2  var AWS = require('aws-sdk');
3  // Set the region
4  AWS.config.update({region: 'sa-east-1'});
5
6  // Create the DynamoDB service object
7  var ddb = new AWS.DynamoDB();
8
9  exports.handler = async ( event ) => {
10     var params = {
11         TableName: 'tabla',
12         FilterExpression: "orbital_period <= :b and orbital_period
               >= :a",
13         ExpressionAttributeValues: {
14         ':a': {N: '500'},
15         ':b': {N: '1000'}
16         },
```

```
        ExpressionAttributeNames : {
        "#c": "name",
        "#d": "population"
        },
        ProjectionExpression: "#c, #d"
    };


    // Call DynamoDB to read the item from the table
    await ddb.scan(params, function(err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data.Items);
      }
    }).promise();
};
```

```
2022-05-11T23:27:58.862Z  f111298e-75ea-4d45-b9fd-5a27e310243c  INFO
     Success [
  { name: { S: 'Polis Massa' }, population: { N: '1000000' } },
  { name: { S: 'Bestine IV' }, population: { N: '62000000' } },
  { name: { S: 'Hoth' } }
]
```

El siguiente código re-escribe los items agregando una columna “string” que clasifica a los planetas por clase según el orden de magnitud de su población y una columna “number” que clasifica los planetas según su periodo orbital:

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'sa-east-1'});

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB();

exports.handler = async ( event ) => {
    let tableContents;
    try{
        //get items from dynamo
        const params = {
            TableName: `${'tabla'}`,
        };
        tableContents = await scanDB(params);
    }catch(err){
        console.log(err);
        return err;
    }

    let calls = [];
    tableContents.forEach(function(value){
      //population
      let params;
```

```
26        if (value["population"] == undefined){
27          params = {
28              ExpressionAttributeValues: {
29                  ":newAttribute": {S: ''}
30              },
31              Key: {
32                  "name": value.name
33              },
34              TableName: `${'tabla'}`,
35              UpdateExpression: "SET class_population = :newAttribute"
                    ,
36              };
37        }
38        else if (Number(value["population"].N) <= 1000){
39          params = {
40              ExpressionAttributeValues: {
41                  ":newAttribute": {S: 'clase A'}
42              },
43              Key: {
44                  "name": value.name
45              },
46              TableName: `${'tabla'}`,
47              UpdateExpression: "SET class_population = :newAttribute"
                    ,
48              };
49        }
50        else if(Number(value["population"].N) <= 1000000){
51            params = {
52                ExpressionAttributeValues: {
53                    ":newAttribute": {S: 'clase B'}
54                },
55                Key: {
56                    "name": value.name
57                },
58                TableName: `${'tabla'}`,
59                UpdateExpression: "SET class_population = :
                      newAttribute",
60                };
61        }
62        else if(Number(value["population"].N) <= 1000000000){
63            params = {
64                ExpressionAttributeValues: {
65                    ":newAttribute": {S: 'clase C'}
66                },
67                Key: {
68                    "name": value.name
69                },
70                TableName: `${'tabla'}`,
71                UpdateExpression: "SET class_population = :
                      newAttribute",
72                };
73        }
74        else{
75          params = {
```

```
76            ExpressionAttributeValues: {
77                ":newAttribute": {S: 'clase D'}
78            },
79            Key: {
80                "name": value.name
81            },
82            TableName: `${'tabla'}`,
83            UpdateExpression: "SET class_population = :newAttribute",
84            };
85        }
86        calls.push(ddb.updateItem(params).promise());
87
88        //orbital_period
89        let params2;
90        if (value["orbital_period"] == undefined){
91          params2 = {
92              ExpressionAttributeValues: {
93                  ":newAttribute": {S: ''}
94              },
95              Key: {
96                  "name": value.name
97              },
98              TableName: `${'tabla'}`,
99              UpdateExpression: "SET class_orbital = :newAttribute",
100             };
101       }
102       else if (Number(value["orbital_period"].N) <= 250){
103         params2 = {
104             ExpressionAttributeValues: {
105                 ":newAttribute": {S: 'clase A'}
106             },
107             Key: {
108                 "name": value.name
109             },
110             TableName: `${'tabla'}`,
111             UpdateExpression: "SET class_orbital = :newAttribute",
112             };
113       }
114       else if(Number(value["orbital_period"].N) <= 500){
115           params2 = {
116               ExpressionAttributeValues: {
117                   ":newAttribute": {S: 'clase B'}
118               },
119               Key: {
120                   "name": value.name
121               },
122               TableName: `${'tabla'}`,
123               UpdateExpression: "SET class_orbital = :newAttribute",
124               };
125       }
126       else if(Number(value["orbital_period"].N) <= 750){
127           params2 = {
128               ExpressionAttributeValues: {
129                   ":newAttribute": {S: 'clase C'}
```

```
130                 },
131                 Key: {
132                     "name": value.name
133                 },
134                 TableName: `${'tabla'}`,
135                 UpdateExpression: "SET class_orbital = :newAttribute",
136                 };
137         }
138         else if(Number(value["orbital_period"].N) <= 1000){
139             params2 = {
140                 ExpressionAttributeValues: {
141                     ":newAttribute": {S: 'clase D'}
142                 },
143                 Key: {
144                     "name": value.name
145                 },
146                 TableName: `${'tabla'}`,
147                 UpdateExpression: "SET class_orbital = :newAttribute",
148                 };
149         }
150         else{
151           params2 = {
152             ExpressionAttributeValues: {
153                 ":newAttribute": {S: 'clase E'}
154             },
155             Key: {
156                 "name": value.name
157             },
158             TableName: `${'tabla'}`,
159             UpdateExpression: "SET class_orbital = :newAttribute",
160             };
161         }
162         calls.push(ddb.updateItem(params2).promise());
163     });
164 
165     let response;
166     try{
167         response = await Promise.all(calls);
168     }catch(err){
169         console.log(err);
170     }
171     return response;
172 };
173 
174 async function scanDB(params) {
175     let dynamoContents = [];
176     let items;
177     do{
178         items =  await ddb.scan(params).promise();
179         items.Items.forEach((item) => dynamoContents.push(item));
180         params.ExclusiveStartKey  = items.LastEvaluatedKey;
181     }while(typeof items.LastEvaluatedKey != "undefined");
182     return dynamoContents;
183 }
```

El siguiente código y resultado muestra un Scan para obtener el nombre y diámetro de todos los planetas en la lista que son de clase B o C:

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'sa-east-1'});

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB();

exports.handler = async ( event ) => {
    var params = {
            TableName: 'tabla',
            FilterExpression: 'contains(class_orbital, :B) OR
               contains(class_orbital, :C) OR contains(
               class_population, :B) OR contains(class_population, :
               C)',
            ExpressionAttributeValues: {
                ':B': {
                    S: 'clase B'
                },
                ':C': {
                    S: 'clase C'
                }
            },
            ExpressionAttributeNames : {
                "#n": "name",
                "#d": "diameter"
            },
            ProjectionExpression: "#n, #d"
        };

        await ddb.scan(params, function(err, data) {
            if (err) console.log(err); // an error occurred
            else console.log(data.Items); // successful response
        }).promise();
};
```

```
2022-05-13T00:09:57.147Z  4ec8b4d0-f087-43b6-b755-4bd73bd950a7  INFO
     [
  { diameter: { N: '9830' }, name: { S: 'Dantooine' } },
  { diameter: { N: '0' }, name: { S: 'Cato Neimoidia' } },
  { diameter: { N: '13800' }, name: { S: 'Muunilinst' } },
  { diameter: { N: '12900' }, name: { S: 'Utapau' } },
  { diameter: { N: '12240' }, name: { S: 'Coruscant' } },
  { diameter: { N: '0' }, name: { S: 'Polis Massa' } },
  { diameter: { N: '10088' }, name: { S: 'Mygeeto' } },
  { diameter: { N: '12120' }, name: { S: 'Naboo' } },
  { diameter: { N: '13500' }, name: { S: 'Chandrila' } },
  { diameter: { N: '13490' }, name: { S: 'Eriadu' } },
  { diameter: { N: '19720' }, name: { S: 'Kamino' } },
  { diameter: { N: '8900' }, name: { S: 'Dagobah' } },
  { diameter: { N: '11030' }, name: { S: 'Mon Cala' } },
```

```
  { name: { S: 'Champala' } },
  { diameter: { N: '10120' }, name: { S: 'Haruun Kal' } },
  { diameter: { N: '7549' }, name: { S: 'Rodia' } },
  { diameter: { N: '12780' }, name: { S: 'Sullust' } },
  { diameter: { N: '13850' }, name: { S: 'Kalee' } },
  { diameter: { N: '12765' }, name: { S: 'Kashyyyk' } },
  { diameter: { N: '13400' }, name: { S: 'Dorin' } },
  { diameter: { N: '11370' }, name: { S: 'Geonosis' } },
  { diameter: { N: '14900' }, name: { S: 'Vulpter' } },
  { name: { S: 'Iktotch' } },
  { name: { S: 'Ojom' } },
  { diameter: { N: '6400' }, name: { S: 'Bestine IV' } },
  { diameter: { N: '4200' }, name: { S: 'Mustafar' } },
  { diameter: { N: '14920' }, name: { S: 'Saleucami' } },
  { diameter: { N: '4900' }, name: { S: 'Endor' } },
  { diameter: { N: '0' }, name: { S: 'Trandosha' } },
  { diameter: { N: '10465' }, name: { S: 'Tatooine' } },
  { diameter: { N: '12150' }, name: { S: 'Nal Hutta' } },
  { diameter: { N: '118000' }, name: { S: 'Bespin' } },
  { diameter: { N: '12500' }, name: { S: 'Alderaan' } },
  { diameter: { N: '0' }, name: { S: 'Socorro' } },
  { diameter: { N: '10600' }, name: { S: 'Ryloth' } },
  { name: { S: 'Cerea' } },
  { name: { S: 'Skako' } },
  { diameter: { N: '11000' }, name: { S: 'Corellia' } },
  { name: { S: 'Iridonia' } },
  { diameter: { N: '7200' }, name: { S: 'Hoth' } },
  { diameter: { N: '15600' }, name: { S: 'Glee Anselm' } },
  { diameter: { N: '9100' }, name: { S: 'Felucia' } },
  { diameter: { N: '7900' }, name: { S: 'Toydaria' } },
  { diameter: { N: '14050' }, name: { S: 'Ord Mantell' } },
  { diameter: { N: '10480' }, name: { S: 'Dathomir' } }
]
```

5. Operaciones sobre indice

El siguiente código y resultado muestra un Query para obtener todos los planetas población clase C y periodo orbital clase B:

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'sa-east-1'});

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB();

exports.handler = async ( event ) => {
    var params = {
            TableName: 'tabla',
```

```
            IndexName: "class_population-class_orbital-index",
            FilterExpression: 'contains(class_orbital, :B) AND
               contains(class_population, :C)',
            ExpressionAttributeValues: {
                ':B': {
                    S: 'clase B'
                },
                ':C': {
                    S: 'clase C'
                }
            }
        };

        await ddb.scan(params, function(err, data) {
            if (err) console.log(err); // an error occurred
            else console.log(data.Items); // successful response
        }).promise();
};
```

```
2022-05-13T23:17:02.248Z  08b6d988-1570-4950-95b6-ce67b12e7f1c  INFO
    [
  {
    diameter: { N: '0' },
    population: { N: '10000000' },
    class_population: { S: 'clase C' },
    name: { S: 'Cato Neimoidia' },
    class_orbital: { S: 'clase B' }
  },
  {
    diameter: { N: '12900' },
    population: { N: '95000000' },
    class_population: { S: 'clase C' },
    name: { S: 'Utapau' },
    class_orbital: { S: 'clase B' }
  },
  {
    diameter: { N: '19720' },
    population: { N: '1000000000' },
    class_population: { S: 'clase C' },
    name: { S: 'Kamino' },
    class_orbital: { S: 'clase B' }
  },
  {
    diameter: { N: '12765' },
    population: { N: '45000000' },
    class_population: { S: 'clase C' },
    name: { S: 'Kashyyyk' },
    class_orbital: { S: 'clase B' }
  },
  {
    diameter: { N: '14900' },
    population: { N: '421000000' },
    class_population: { S: 'clase C' },
    name: { S: 'Vulpter' },
```

```
    class_orbital: { S: 'clase B' }
  },
  {
    diameter: { N: '4900' },
    population: { N: '30000000' },
    class_population: { S: 'clase C' },
    name: { S: 'Endor' },
    class_orbital: { S: 'clase B' }
  },
  {
    diameter: { N: '0' },
    population: { N: '42000000' },
    class_population: { S: 'clase C' },
    name: { S: 'Trandosha' },
    class_orbital: { S: 'clase B' }
  },
  {
    diameter: { N: '0' },
    population: { N: '300000000' },
    class_population: { S: 'clase C' },
    name: { S: 'Socorro' },
    class_orbital: { S: 'clase B' }
  },
  {
    population: { N: '450000000' },
    class_population: { S: 'clase C' },
    name: { S: 'Cerea' },
    class_orbital: { S: 'clase B' }
  }
]
```

El siguiente código y resultado muestra las operaciones necesarias para obtener todos los datos del planeta con menor densidad de población cuya población es de clase D y periodo orbital clase B:

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');
// Set the region
AWS.config.update({region: 'sa-east-1'});

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB();

exports.handler = async ( event ) => {
    var params = {
            TableName: 'tabla',
            IndexName: "class_population-class_orbital-index",
            FilterExpression: 'contains(class_orbital, :B) AND
               contains(class_population, :D)',
            ExpressionAttributeValues: {
                ':B': {
                    S: 'clase B'
                },
```

```
                ':D': {
                    S: 'clase D'
                }
            }
        };
    let items;
    let poblacion = Number.POSITIVE_INFINITY;
    let item_final;
    items = await ddb.scan(params).promise();
    items.Items.forEach(item => {
        if(Number(item['population'].N) < poblacion){
            poblacion = Number(item['population'].N);
            item_final = item;
        }
    });
    return item_final;
};
```

```
{
  "diameter": {
    "N": "13500"
  },
  "population": {
    "N": "1200000000"
  },
  "class_population": {
    "S": "clase D"
  },
  "name": {
    "S": "Chandrila"
  },
  "class_orbital": {
    "S": "clase B"
  }
}
```