EBook Gratis

APRENDIZAJE amazon-dynamodb

Free unaffiliated eBook created from
Stack Overflow contributors.

#amazon-dynamodb

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: amazon-dynamodb

It is an unofficial and free amazon-dynamodb ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official amazon-dynamodb.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con amazon-dynamodb

Observaciones

Esta sección proporciona una descripción general de qué es amazon-dynamodb y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de amazon-dynamodb, y vincular a los temas relacionados. Dado que la Documentación para amazon-dynamodb es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

Examples

Instalación o configuración

DynamoDB es un servicio totalmente gestionado proporcionado por AWS. No necesita ser instalado o configurado. AWS es responsable de todas las cargas administrativas de operación, escalonamiento y copia de seguridad / restauración de la base de datos distribuida.

Introducción

DynamoDB es una base de datos distribuida NoSQL, basada en una arquitectura de clave-valor, totalmente gestionada por Amazon Web Services. Fue diseñado para proporcionar escalabilidad, redundancia y conmutación por error en un rendimiento predecible.

Lea Empezando con amazon-dynamodb en línea: https://riptutorial.com/es/amazon-dynamodb/topic/2085/empezando-con-amazon-dynamodb

Capítulo 2: ¿Cómo insertar datos en la tabla usando DynamoDb?

Examples

Importe un archivo CSV en una tabla de DynamoDB usando boto (paquete Python)

La función de Python `import_csv_to_dynamodb(table_name, csv_file_name, colunm_names, column_types)` debajo importa un archivo CSV en una tabla de DynamoDB. Los nombres de columna y columna deben ser especificados. Utiliza boto . A continuación se muestra la función, así como una demostración (`main()`) y el archivo CSV utilizado.

```
import boto

MY_ACCESS_KEY_ID = 'copy your access key ID here'
MY_SECRET_ACCESS_KEY = 'copy your secrete access key here'


def do_batch_write(items, table_name, dynamodb_table, dynamodb_conn):
    '''
    From https://gist.github.com/griggheo/2698152#file-gistfile1-py-L31
    '''
    batch_list = dynamodb_conn.new_batch_write_list()
    batch_list.add_batch(dynamodb_table, puts=items)
    while True:
        response = dynamodb_conn.batch_write_item(batch_list)
        unprocessed = response.get('UnprocessedItems', None)
        if not unprocessed:
            break
        batch_list = dynamodb_conn.new_batch_write_list()
        unprocessed_list = unprocessed[table_name]
        items = []
        for u in unprocessed_list:
            item_attr = u['PutRequest']['Item']
            item = dynamodb_table.new_item(
                    attrs=item_attr
            )
            items.append(item)
        batch_list.add_batch(dynamodb_table, puts=items)


def import_csv_to_dynamodb(table_name, csv_file_name, colunm_names, column_types):
    '''
    Import a CSV file to a DynamoDB table
    '''
    dynamodb_conn = boto.connect_dynamodb(aws_access_key_id=MY_ACCESS_KEY_ID,
aws_secret_access_key=MY_SECRET_ACCESS_KEY)
    dynamodb_table = dynamodb_conn.get_table(table_name)
    BATCH_COUNT = 2 # 25 is the maximum batch size for Amazon DynamoDB

    items = []
```

```
    count = 0
    csv_file = open(csv_file_name, 'r')
    for cur_line in csv_file:
        count += 1
        cur_line = cur_line.strip().split(',')

        row = {}
        for colunm_number, colunm_name in enumerate(colunm_names):
            row[colunm_name] = column_types[colunm_number](cur_line[colunm_number])

        item = dynamodb_table.new_item(
                    attrs=row
            )
        items.append(item)

        if count % BATCH_COUNT == 0:
            print 'batch write start ... ',
            do_batch_write(items, table_name, dynamodb_table, dynamodb_conn)
            items = []
            print 'batch done! (row number: ' + str(count) + ')'

    # flush remaining items, if any
    if len(items) > 0:
        do_batch_write(items, table_name, dynamodb_table, dynamodb_conn)


    csv_file.close()


def main():
    '''
    Demonstration of the use of import_csv_to_dynamodb()
    We assume the existence of a table named `test_persons`, with
    - Last_name as primary hash key (type: string)
    - First_name as primary range key (type: string)
    '''
    colunm_names = 'Last_name First_name'.split()
    table_name = 'test_persons'
    csv_file_name = 'test.csv'
    column_types = [str, str]
    import_csv_to_dynamodb(table_name, csv_file_name, colunm_names, column_types)


if __name__ == "__main__":
    main()
    #cProfile.run('main()') # if you want to do some profiling
```

El contenido de `test.csv` (debe estar ubicado en la misma carpeta que el script de Python):

```
John,Doe
Bob,Smith
Alice,Lee
Foo,Bar
a,b
c,d
e,f
g,h
i,j
j,l
```

Lea ¿Cómo insertar datos en la tabla usando DynamoDb? en línea: https://riptutorial.com/es/amazon-dynamodb/topic/6354/-como-insertar-datos-en-la-tabla-usando-dynamodb-

Capítulo 3: Cómo crear una tabla de DynamoDB

Observaciones

Al crear tablas, asegúrese de prestar atención a la elección de atributos para la partición y las claves de clasificación. Vea las pautas publicadas para trabajar con tablas .

Examples

Crear tabla en Java utilizando Document API

En el siguiente ejemplo, crearemos una tabla llamada `Membership` utilizando el SDK Java de AWS para DynamoDB. La tabla constará de elementos que representan las tareas del equipo. La tabla será particionada por TeamID. Cada equipo tendrá varios miembros, identificados por el MemberID (como una clave de clasificación).

```
AWSCredentials credentials = new BasicAWSCredentials("access_key", "secret_key");
DynamoDB dynamoDB = new DynamoDB(new AmazonDynamoDBClient(credentials));

try {
      // every DynamoDB table must have basic schema that determines
      //   which attributes are to be used as partition key (and optionally sort key)
  List<KeySchemaElement> keySchema = new ArrayList<~>();
      // required: specify the partition key (also called hash key)
  keySchema.add(new KeySchemaElement()
            .withAttributeName("TeamID")
            .withKeyType(KeyType.HASH));
      // optionally: add a sort key (also called a range key)
  keySchema.add(new KeySchemaElement()
            .withAttributeName("MemberID")
            .withKeyType(KeyType.RANGE));

      // after defining the key schema - the attributes that will be used as partition and
range key
      //  we need to specify these attributes' type
  List<AttributeDefinition> attrDefinitions = new ArrayList<~>();
      //  we must specify the type for the TeamID attribute; suppose it's a string
  attrDefinitions.add(new AttributeDefinition()
                .withAttributeName("TeamID")
                .withAttributeType("S"));
      //  if using a sort key we need to specify its type; suppose that it's numeric
  attrDefinitions.add(new AttributeDefinition()
                .withAttributeName("MemberID")
                .withAttributeType("N"));


      // after defining the attributes making up the schema and their type
      // we build a request, specifying the table name and the provisioned capacity
  CreateTableRequest request = new CreateTableRequest()
      .withTableName("Membership")
```

```
      .withKeySchema(keySchema)
      .withAttributeDefinitions(attrDefinitions)
      .withProvisionedThroughput(new ProvisionedThroughput()
           .withReadCapacityUnits(30L)
           .withWriteCapacityUnits(10L));

      // now submit the request to create the table in DynamoDB
  Table table = dynamoDB.createTable(request);

      // creating a table is an asynchronous operation
      // we can wait for the table to be created
  table.waitForActive();

      // the table is now ready and can be used

} catch (Exception e) {
      // the table creation failed.
      //   the reason for failure may be determined from the exception
}
```

Crea una tabla en Ruby usando AWS SDK v2

En el siguiente ejemplo crearemos `movies` tabla con AWS Ruby SDK v2. Aquí, cada película como una clave de partición única como `id` , y `year` clave de rango. Aparte de esto queremos ser capaz de consultar las películas con su nombre, por lo tanto, vamos a crear un índice secundario Global (GSI) `name-year-index` con el `name` como clave hash y `year` como Key Range. La película puede tener otros atributos como `released` , `created_at` , `actor` y `actress` . El esquema para la tabla se muestra a continuación:

Nombre de la tabla : películas

Clave de partición	Clave de rango	Índice secundario global	Atributos
carné de identidad	año	nombre	lanzado, created_at, actor, actriz

```
# it's better to initialize client as global variable in initializer
$ddb ||= Aws::DynamoDB::Client.new({
  access_key_id: ENV["AWS_ACCESS_KEY_ID"],
  secret_access_key: ENV["AWS_SECRET_ACCESS_KEY"]
})

# create table API
$ddb.create_table({

  # array of attributes name and their type that describe schema for the Table and Indexes
  attribute_definitions: [
    {
      attribute_name: "id",
      attribute_type: "N"
    }
    {
      attribute_name: "name",
      attribute_type: "S",
```

```
    },
    {
      attribute_name: "year",
      attribute_type: "N",
    }
  ],

  # key_schema specifies the attributes that make up the primary key for a table
  # HASH - specifies Partition Key
  # RANGE - specifies Range Key
  # key_type can be either HASH or RANGE
  key_schema: [
    {
      attribute_name: "id",
      key_type: "HASH",
    },
    {
      attribute_name: "year",
      key_type: "RANGE",
    }
  ],

  # global_secondary_indexes array specifies one or more keys that makes up index, with name
of index and provisioned throughput for global secondary indexes
  global_secondary_indexes: [

    index_name: "name-year-index",
    key_schema: [
        {
            attribute_name: "name",
            key_type: "HASH"
        },
        {
            attribute_name: "year",
            key_type: "RANGE"
        }
    ],

    # Projection - Specifies attributes that are copied (projected) from the table into the
index.
    # Allowed values are - ALL, INCLUDE, KEYS_ONLY
    # KEYS_ONLY - only the index and primary keys are projected into the index.
    # ALL - All of the table attributes are projected into the index.
    # INCLUDE - Only the specified table attributes are projected into the index. The list of
projected attributes are then needs to be specified in non_key_attributes array
    projection: {
        projection_type: "ALL"
    },

    # Represents the provisioned throughput settings for specified index.
    provisioned_throughput: {
        read_capacity_units: 1,
        write_capacity_units: 1
    }
  ],

  # Represents the provisioned throughput settings for specified table.
  provisioned_throughput: {
    read_capacity_units: 1,
    write_capacity_units: 1,
  },
```

```
  table_name: "movies"
})

# wait till table is created
$ddb.wait_until(:table_exists, {table_name: "movies"})
```

Lea Cómo crear una tabla de DynamoDB en línea: https://riptutorial.com/es/amazon-dynamodb/topic/7686/como-crear-una-tabla-de-dynamodb

Capítulo 4: Dynamodb borrar datos a lo largo del tiempo

Introducción

Eliminar datos antiguos de dynamodb utilizando un atributo de fecha.

Observaciones

Mi caso de uso: eliminar datos antiguos de dynamodb usando un atributo de fecha.

Cosas importantes que debes saber:

- No puede consultar una tabla utilizando solo el atributo de clave de rango (fecha, por ejemplo).
- Solo puede consultar una tabla usando hash o hash + tecla de rango.
- No puede consultar una tabla usando una clave hash con operaciones '<' / '>', solo '='.

Soluciones posibles:

- Escanear toda la tabla - esto podría ser muy costoso
- Mi solución elegida: definir un índice con clave de rango para la fecha y con una clave de hash que sería bastante decente, como el día del año.

Finalmente eliminar por lotes el conjunto de resultados.

Notas: Construyendo la entidad estaba usando las anotaciones de dinamo amazon. Estaba usando DynamoDBQueryExpression para consultar, obteniendo la página de resultados con el objeto Class definido.

Examples

cleanUpOldData

```
public static void cleanUpOldData(AmazonDynamoDB amazonDynamoDB, String dynamoDBTablesPrefix,
String tableName,
                                  String dateRangeField, String dateHashKey, String dateIndex,
Class clazz) {
    log.info(String.format("Cleaning old data from table: %s", tableName));

    long cleanUpDateInMillis = (new Date()).getTime() - CLEAN_UP_TIME_MILLIS;
    SimpleDateFormat dateFormatter = new SimpleDateFormat(DYNAMO_DATE_FORMAT);
    final TimeZone utcTimeZone = TimeZone.getTimeZone("UTC");
    dateFormatter.setTimeZone(utcTimeZone);
    String cleanUpDate = dateFormatter.format(cleanUpDateInMillis);

    Calendar calendar = Calendar.getInstance(utcTimeZone);
```

```
    calendar.setTimeInMillis(cleanUpDateInMillis);

    final String dailyHashKey = String.format("%s_%s", calendar.get(Calendar.YEAR),
calendar.get(Calendar.DAY_OF_YEAR));
    final String pastDayHashKey = String.format("%s_%s", calendar.get(Calendar.YEAR),
calendar.get(Calendar.DAY_OF_YEAR)-1);

    final String fullTableName = dynamoDBTablesPrefix + "_" + tableName;
    final DynamoDBMapperConfig dbMapperConfig = new DynamoDBMapperConfig(new
DynamoDBMapperConfig.TableNameOverride(fullTableName));
    DynamoDBMapper mapper = new DynamoDBMapper(amazonDynamoDB, dbMapperConfig);
    DynamoDBTableMapper dbTableMapper = mapper.newTableMapper(clazz);

    final QueryResultPage dailyResultPage = getDailyQueryResultPage(dateRangeField,
dateHashKey, dateIndex, cleanUpDate, dailyHashKey, dbTableMapper);
    final QueryResultPage pastDayResultPage = getDailyQueryResultPage(dateRangeField,
dateHashKey, dateIndex, cleanUpDate, pastDayHashKey, dbTableMapper);

    deleteOldData(dbTableMapper, dailyResultPage, pastDayResultPage);

    log.info(String.format("Completed cleaning old data from table: %s, %s items were
deleted", tableName,
            dailyResultPage.getCount() + pastDayResultPage.getCount()));
}

private static QueryResultPage getDailyQueryResultPage(String dateRangeField, String
dateHashKey, String dateIndex,
                                                       String cleanUpDate, String dayHashKey,
DynamoDBTableMapper dbTableMapper) {
    HashMap<String, String > nameMap = new HashMap<>();
    nameMap.put("#date", dateRangeField);
    nameMap.put("#day", dateHashKey);
    HashMap<String, AttributeValue> valueMap = new HashMap<>();
    valueMap.put(":date", new AttributeValue().withS(cleanUpDate)) ;
    valueMap.put(":day", new AttributeValue().withS(dayHashKey));

    final DynamoDBQueryExpression dbQueryExpression = new DynamoDBQueryExpression()
            .withIndexName(dateIndex)
            .withConsistentRead(false)
            .withKeyConditionExpression("#day = :day and #date < :date")
            .withExpressionAttributeNames(nameMap)
            .withExpressionAttributeValues(valueMap);
    return dbTableMapper.query(dbQueryExpression);
}

private static void deleteOldData(DynamoDBTableMapper dbTableMapper, QueryResultPage
dailyResultPage, QueryResultPage pastDayResultPage) {
    if (dailyResultPage.getCount() > 0) {
        dbTableMapper.batchDelete(dailyResultPage.getResults());
    }
    if (pastDayResultPage.getCount() > 0) {
        dbTableMapper.batchDelete(pastDayResultPage.getResults());
    }
}
```

Lea Dynamodb borrar datos a lo largo del tiempo en línea: https://riptutorial.com/es/amazon-dynamodb/topic/10889/dynamodb-borrar-datos-a-lo-largo-del-tiempo

Capítulo 5: Operaciones por lotes: cosas que debe saber

Introducción

La base de datos es una parte integral de cualquier aplicación y el rendimiento y la persistencia son desafíos reales que enfrenta cualquier aplicación web. Las bases de datos NoSql no son diferentes en este asunto y deben tratarse con cuidado. DynamoDB es una de las bases de datos NoSQL que proporciona Amazon Web Services, que admite operaciones por lotes además de las operaciones CRUD. Vamos a empezar con Batch Operations. En este ejemplo, aprenderemos cómo podemos utilizar el SDK de JAVA de Dynamo DB para realizar inserciones por lotes.

Observaciones

Es bueno saber acerca de las operaciones por lotes

1. La operación por lotes no minimiza el recuento de solicitudes HTTP, sino que tiene más funciones para manejar cuando recibimos un error de regulación. Para ponerlo simple, cada registro que insertemos en dynamo db consumirá una solicitud http.
2. Una buena implementación de la operación por lotes es primero almacenar en caché los datos temporalmente y una vez que tengamos el número de umbral, es decir, 25, es el momento correcto para iniciar una solicitud por lotes.
3. Cualquier solicitud que falle debido a la limitación se reintentará entre (500-999) milisegundos, según lo recomendado por las mejores prácticas de Amazon.

Examples

Cómo codificar BatchWriteItemRequest y guardar datos

```
private static void saveItem(List<EventTracker> items) {
    List<WriteRequest> wrList = new ArrayList<>();
    try {

        for (EventTracker item : items) {
            WriteRequest wreqItem;
            wreqItem = getWriteRequest(item);
            wrList.add(wreqItem);
        }

        try {

            BatchWriteItemResult batchWriteItemResult = new BatchWriteItemResult();
            do {
                BatchWriteItemRequest batchWriteItemRequest = new BatchWriteItemRequest();
                batchWriteItemRequest.addRequestItemsEntry(forumTableName, wrList);//
setRequestItems(writeRequestitems);
                batchWriteItemResult = amazonDynamoDB.batchWriteItem(batchWriteItemRequest);
```

```
                // Check for unprocessed keys which could happen if you
                // exceed
                // provisioned throughput
                Map<String, List<WriteRequest>> unprocessedItems =
batchWriteItemResult.getUnprocessedItems();
                if (unprocessedItems.size() == 0) {
                    System.out.println("No unprocessed items found");
                } else {
                    System.out.println("Sleeping for: " +
ThreadLocalRandom.current().nextInt(500, 999 + 1));
                    Thread.sleep(ThreadLocalRandom.current().nextInt(500, 999 + 1));
                    wrList = unprocessedItems.get(forumTableName);
                    System.out.println("Retrieving the unprocessed items");
                }

            } while (batchWriteItemResult.getUnprocessedItems().size() > 0);

        } catch (Exception e) {
            System.err.println("Failed to retrieve items: ");
            e.printStackTrace(System.err);
        }

    } catch (Exception e) {

    }
}
```

Esto es lo que necesitamos saber si queremos hacer uso de las operaciones por lotes para colocar datos en una tabla de db dynamo. Veamos los pasos que deben seguirse para lograr esto. En este ejemplo, estamos tratando de conservar una lista de datos de EventTracker, que es mi POJO.

1. Para que se inserte cada registro, debemos crear una solicitud PUT.
2. Cada solicitud PUT se envuelve en una solicitud de escritura.
3. Todas las solicitudes de escritura se incluyen en una lista.
4. La lista WriteRequest se agrega a BatchWriteItemRequest y se ejecuta.

El siguiente código mostrará cómo creamos solicitudes de escritura.

Cómo crear WriteRequest

```
private static WriteRequest getWriteRequest(EventTracker event) {

    WriteRequest wreq = null;// = new WriteRequest();

    if (event != null) {
        Map<String, AttributeValue> attributeMap = new HashMap<String, AttributeValue>();

        addAttribute(attributeMap, "event_id", event.getEventId());
        addAttribute(attributeMap, "created_datetime", event.getCreatedDatetime());
        addAttribute(attributeMap, "event", event.getEvent());
        addAttribute(attributeMap, "event_type", event.getEventType());
        addAttribute(attributeMap, "response_id", "NULL");

        wreq = new WriteRequest(new PutRequest(attributeMap));
    }
```

```
    return wreq;
}

private static void addAttribute(Map<String, AttributeValue> item, String attributeName,
String value) {
    AttributeValue attributeValue = new AttributeValue(value);
    item.put(attributeName, attributeValue);
}
```

Lea Operaciones por lotes: cosas que debe saber en línea: https://riptutorial.com/es/amazon-dynamodb/topic/8675/operaciones-por-lotes--cosas-que-debe-saber

Capítulo 6: Uso de DynamoDb de AWS con el SDK de .NET de AWS

Observaciones

Amazon DynamoDB es un servicio rápido de base de datos NoSQL ofrecido por Amazon Web Services (AWS). Se puede invocar DynamoDB desde aplicaciones .NET utilizando AWS SDK para .NET . El SDK proporciona tres modelos diferentes para comunicarse con DynamoDB. Este tema presenta las distintas API de cada modelo.

Los modelos

El SDK proporciona tres formas de comunicarse con DynamoDB. Cada uno ofrece compensaciones entre el control y la facilidad de uso. Consulte la referencia de AWS .NET SDK para obtener detalles sobre las API a continuación.

- **Bajo nivel** : espacio de nombres `Amazon.DynamoDBv2` : se trata de una envoltura delgada sobre las llamadas de servicio de DynamoDB. Coincide con todas las características del servicio. Puede consultar la documentación del servicio para obtener más información sobre cada operación individual.

- **Modelo de documento** : espacio de nombres `Amazon.DynamoDBv2.DocumentModel` : este es un modelo que proporciona una interfaz más sencilla para tratar con los datos. Las tablas de DynamoDB están representadas por objetos de `Table` , mientras que las filas de datos individuales están representadas por objetos de `Document` . La conversión de objetos .NET a datos de DynamoDB es automática para los tipos básicos.

- **Modelo de persistencia de objetos** : espacio de nombres `Amazon.DynamoDBv2.DataModel` : este conjunto de API le permite almacenar y cargar objetos .NET en DynamoDB. Los objetos deben estar marcados para configurar la tabla de destino y las teclas hash / range. `DynamoDBContext` actúa sobre los objetos marcados. Se utiliza para almacenar y cargar datos de DynamoDB, o para recuperar objetos .NET de una consulta o operación de escaneo. Los tipos de datos básicos se convierten automáticamente a datos de DynamoDB y los convertidores permiten que se almacenen tipos arbitrarios en DynamoDB.

Los tres modelos proporcionan diferentes enfoques para trabajar con el servicio. Si bien el enfoque de bajo nivel requiere más código del lado del cliente (el usuario debe convertir los tipos .NET, como números y fechas, a cadenas compatibles con DynamoDB), brinda acceso a todas las funciones del servicio. En comparación, el enfoque del Modelo de Persistencia del Objeto facilita el uso del servicio, ya que el usuario trabaja en su mayoría con objetos .NET conocidos, pero no proporciona toda la funcionalidad. Por ejemplo, no es posible realizar llamadas condicionales con el modelo de persistencia de objetos.

Obtenga más información sobre cómo trabajar con AWS utilizando .NET SDK en la Guía del

desarrollador de .NET SDK .

Nota: este tema se adaptó con el permiso de una publicación de blog publicada originalmente en el blog AWS .NET SDK .

Examples

Ejemplo de API de bajo nivel

```
var client = new AmazonDynamoDBClient();

// Store item
client.PutItem(new PutItemRequest
{
    TableName = "Books",
    Item = new Dictionary<string, AttributeValue>
    {
        { "Title", new AttributeValue { S = "Cryptonomicon" } },
        { "Id", new AttributeValue { N = "42" } },
        { "Authors", new AttributeValue {
            SS = new List<string> { "Neal Stephenson" } } },
        { "Price", new AttributeValue { N = "12.95" } }
    }
});

// Get item
Dictionary<string, AttributeValue> book = client.GetItem(new GetItemRequest
{
    TableName = "Books",
    Key = new Dictionary<string, AttributeValue>
    {
        { "Id", new AttributeValue { N = "42" } }
    }
}).Item;

Console.WriteLine("Id = {0}", book["Id"].S);
Console.WriteLine("Title = {0}", book["Title"].S);
Console.WriteLine("Authors = {0}",
    string.Join(", ", book["Authors"].SS));
```

Ejemplo de modelo de documento

```
var client = new AmazonDynamoDBClient();
Table booksTable = Table.LoadTable(client, "Books");

// Store item
Document book = new Document();
book["Title"] = "Cryptonomicon";
book["Id"] = 42;
book["Authors"] = new List<string> { "Neal Stephenson" };
book["Price"] = 12.95;
booksTable.PutItem(book);

// Get item
book = booksTable.GetItem(42);
Console.WriteLine("Id = {0}", book["Id"]);
```

```
Console.WriteLine("Title = {0}", book["Title"]);
Console.WriteLine("Authors = {0}",
    string.Join(", ", book["Authors"].AsListOfString()));
```

Ejemplo de modelo de persistencia de objetos

Este ejemplo consta de dos partes: primero, debemos definir nuestro tipo de `Book` ; En segundo lugar, lo usamos con `DynamoDBContext` .

```
[DynamoDBTable("Books")]
class Book
{
    [DynamoDBHashKey]
    public int Id { get; set; }
    public string Title { get; set; }
    public List<string> Authors { get; set; }
    public double Price { get; set; }
}
```

Ahora, `DynamoDBContext` con `DynamoDBContext` .

```
var client = new AmazonDynamoDBClient();
DynamoDBContext context = new DynamoDBContext(client);

// Store item
Book book = new Book
{
    Title = "Cryptonomicon",
    Id = 42,
    Authors = new List<string> { "Neal Stephenson" },
    Price = 12.95
};
context.Save(book);

// Get item
book = context.Load<Book>(42);
Console.WriteLine("Id = {0}", book.Id);
Console.WriteLine("Title = {0}", book.Title);
Console.WriteLine("Authors = {0}", string.Join(", ", book.Authors));
```

Lea Uso de DynamoDb de AWS con el SDK de .NET de AWS en línea: https://riptutorial.com/es/amazon-dynamodb/topic/5275/uso-de-dynamodb-de-aws-con-el-sdk-de--net-de-aws

Creditos

S. No	Capítulos	Contributors
1	Empezando con amazon-dynamodb	Community, Gustavo Tavares
2	¿Cómo insertar datos en la tabla usando DynamoDb?	Franck Dernoncourt
3	Cómo crear una tabla de DynamoDB	Mike Dinescu, Parth Modi
4	Dynamodb borrar datos a lo largo del tiempo	Lior
5	Operaciones por lotes: cosas que debe saber	SACHESH A C
6	Uso de DynamoDb de AWS con el SDK de .NET de AWS	David Hale