eBook Gratuit

APPRENEZ amazon-dynamodb

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#amazon-dynamodb

Table des matières

À propos

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

Chapitre 1: Démarrer avec amazon-dynamodb

Remarques

Cette section fournit une vue d'ensemble de ce qu'est amazon-dynamodb et pourquoi un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les grands sujets dans amazon-dynamodb, et établir un lien avec les sujets connexes. La documentation pour amazon-dynamodb étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Examples

Installation ou configuration

DynamoDB est un service entièrement géré fourni par AWS. Il n'a pas besoin d'être installé ou configuré. AWS est responsable de toutes les tâches administratives liées à l'exploitation, à l'écaillage et à la sauvegarde / restauration de la base de données distribuée.

introduction

DynamoDB est une base de données Distributed NoSQL, basée sur une architecture à valeur-clé, entièrement gérée par Amazon Web Services. Il a été conçu pour offrir une évolutivité, une redondance et un basculement dans des performances prévisibles.

Lire Démarrer avec amazon-dynamodb en ligne: https://riptutorial.com/fr/amazon-dynamodb/topic/2085/demarrer-avec-amazon-dynamodb

Chapitre 2: Comment créer une table DynamoDB

Remarques

Lors de la création de tables, assurez-vous de faire attention au choix des attributs pour la partition et les clés de tri. Voir les directives publiées pour travailler avec des tables .

Examples

Créer une table en Java à l'aide de l'API de document

Dans l'exemple suivant, nous allons créer une table appelée `Membership` à l'aide du kit SDK AWS Java pour DynamoDB. Le tableau sera composé d'éléments représentant les affectations d'équipe. La table sera partitionnée par TeamID. Chaque équipe aura plusieurs membres, identifiés par le MemberID (comme clé de tri).

```
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
}
```

Créer une table dans Ruby en utilisant AWS SDK v2

Dans l'exemple suivant, nous allons créer des `movies` table avec AWS Ruby SDK v2. Ici, chaque film représente une clé de partition unique en tant `id` et une `year` clé de plage. En dehors de cela, nous voulons être en mesure d'interroger des films avec leur nom, nous allons donc créer un index de `name-year-index` GSI (Global Secondary Index) avec un `name` comme clé de hachage et l' `year` comme clé de plage. Film peut avoir d' autres attributs tels que `released` , `created_at` , `actor` et `actress` . Le schéma de la table est illustré ci-dessous:

Nom de la table : films

Clé de partition	Clé de portée	Index secondaire global	Les attributs
id	an	prénom	libéré, created_at, acteur, actrice

```
# 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"})
```

Lire Comment créer une table DynamoDB en ligne: https://riptutorial.com/fr/amazon-dynamodb/topic/7686/comment-creer-une-table-dynamodb

Chapitre 3: Comment insérer des données dans une table à l'aide de DynamoDb?

Examples

Importer un fichier CSV dans une table DynamoDB en utilisant boto (package Python)

La fonction Python `import_csv_to_dynamodb(table_name, csv_file_name, colunm_names, column_types)` importe un fichier CSV dans une table DynamoDB. Les noms de colonnes et les colonnes doivent être spécifiés. Il utilise boto . Voici la fonction ainsi qu'une démo (`main()`) et le fichier CSV utilisé.

```
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
```

Le contenu de `test.csv` (doit être situé dans le même dossier que le script Python):

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

Lire Comment insérer des données dans une table à l'aide de DynamoDb? en ligne: https://riptutorial.com/fr/amazon-dynamodb/topic/6354/comment-inserer-des-donnees-dans-une-table-a-l-aide-de-dynamodb-

Chapitre 4: Dynamodb supprime les données au fil du temps

Introduction

Suppression d'anciennes données de dynamodb en utilisant un attribut de date.

Remarques

Mon cas d'utilisation: supprimer les anciennes données de dynamodb en utilisant un attribut de date.

Des choses importantes à savoir:

- Vous ne pouvez pas interroger une table en utilisant uniquement l'attribut clé de plage (date par exemple).
- Vous pouvez uniquement interroger une table en utilisant une clé de hachage ou de hachage +.
- Vous ne pouvez pas interroger une table en utilisant une clé de hachage avec les opérations '<' / '>', seulement '='.

Solutions possibles:

- Scanner la table entière - cela pourrait être très coûteux
- Ma solution choisie - Définir un index avec une clé de plage pour la date et avec une clé de hachage qui serait assez décente comme le jour de l'année.

Finalement, le lot supprime le jeu de résultats.

Notes: En construisant l'entité, j'utilisais les annotations de la dynamo amazonienne. J'utilisais DynamoDBQueryExpression pour interroger, obtenant la page de résultat avec l'objet Class défini.

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());
    }
}
```

Lire Dynamodb supprime les données au fil du temps en ligne: https://riptutorial.com/fr/amazon-dynamodb/topic/10889/dynamodb-supprime-les-donnees-au-fil-du-temps

Chapitre 5: Opérations par lots: choses à savoir

Introduction

La base de données fait partie intégrante de toute application et la performance et la persistance sont de véritables défis pour toute application Web. Les bases de données NoSql ne sont pas différentes en la matière et doivent être traitées avec soin. DynamoDB étant l'une des bases de données NoSQL fournies par Amazon Web Services pour prendre en charge les opérations par lots en plus des opérations CRUD. Commençons par les opérations par lots. Dans cet exemple, nous allons apprendre comment utiliser JAVA SDK de Dynamo DB pour effectuer des insertions par lots.

Remarques

Bon à savoir sur les opérations par lots

1. Le traitement par lots ne minimise pas le nombre de requêtes HTTP, mais il a plus de fonctionnalités à gérer lorsque nous recevons une erreur de limitation. Pour simplifier, chaque enregistrement inséré dans la dynamo db consomme une requête http.
2. L'implémentation sonore de l'opération de traitement par lots consiste d'abord à mettre temporairement les données en mémoire cache. Une fois que nous avons le numéro de seuil, à savoir 25, l'heure est correcte pour lancer une requête par lot.
3. Toute demande qui échoue en raison d'une limitation sera réessayée entre 500 et 999 millisecondes, comme recommandé par les meilleures pratiques d'Amazon.

Examples

Comment coder le BatchWriteItemRequest et enregistrer des données

```
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) {

    }
}
```

C'est ce que nous devons savoir si nous voulons utiliser des opérations par lots pour placer des données dans une table de dynamo db. Voyons les étapes à suivre pour y parvenir. Dans cet exemple, nous essayons de conserver une liste de données EventTracker, qui est mon POJO.

1. Pour chaque enregistrement à insérer, nous devons créer une demande PUT.
2. Chaque demande PUT est encapsulée dans une demande d'écriture.
3. Toutes les demandes d'écriture sont regroupées dans une liste.
4. La liste WriteRequest est ensuite ajoutée à BatchWriteItemRequest et exécutée.

Le code ci-dessous montre comment nous créons des demandes d'écriture.

Comment créer 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);
}
```

Lire Opérations par lots: choses à savoir en ligne: https://riptutorial.com/fr/amazon-dynamodb/topic/8675/operations-par-lots--choses-a-savoir

Chapitre 6: Utilisation d'AWS DynamoDb avec le SDK AWS .NET

Remarques

Amazon DynamoDB est un service de base de données NoSQL rapide proposé par Amazon Web Services (AWS). DynamoDB peut être appelé à partir d'applications .NET à l'aide du kit AWS SDK pour .NET . Le SDK fournit trois modèles différents pour communiquer avec DynamoDB. Cette rubrique présente les différentes API de chaque modèle.

Les modèles

Le SDK offre trois manières de communiquer avec DynamoDB. Chacune offre des compromis entre le contrôle et la facilité d'utilisation. Consultez la référence AWS .NET SDK pour plus de détails sur les API ci-dessous.

- **Bas niveau** : `Amazon.DynamoDBv2` noms `Amazon.DynamoDBv2` - Il s'agit d'une enveloppe mince pour les appels de service DynamoDB. Il correspond à toutes les fonctionnalités du service. Vous pouvez vous référer à la documentation du service pour en savoir plus sur chaque opération individuelle.

- **Modèle de document** : `Amazon.DynamoDBv2.DocumentModel` noms `Amazon.DynamoDBv2.DocumentModel` - Il s'agit d'un modèle qui fournit une interface plus simple pour gérer les données. Les tables DynamoDB sont représentées par des objets `Table` , tandis que les lignes de données individuelles sont représentées par les objets `Document` . La conversion d'objets .NET en données DynamoDB est automatique pour les types de base.

- **Modèle de persistance d'objet** : `Amazon.DynamoDBv2.DataModel` noms `Amazon.DynamoDBv2.DataModel` - Cet ensemble d'API vous permet de stocker et de charger des objets .NET dans DynamoDB. Les objets doivent être marqués pour configurer la table cible et les clés de hachage / plage. `DynamoDBContext` agit sur les objets balisés. Il est utilisé pour stocker et charger des données DynamoDB ou pour récupérer des objets .NET à partir d'une requête ou d'une opération d'analyse. Les types de données de base sont automatiquement convertis en données DynamoDB et les convertisseurs permettent de stocker des types arbitraires dans DynamoDB.

Les trois modèles proposent différentes approches pour travailler avec le service. Bien que l'approche de bas niveau nécessite davantage de code côté client - l'utilisateur doit convertir les types .NET tels que les nombres et les dates en chaînes supportées par DynamoDB - il donne accès à toutes les fonctionnalités du service. En comparaison, l'approche du modèle de persistance d'objets facilite l'utilisation du service, car la plupart du temps, l'utilisateur travaille avec des objets .NET familiers, mais ne fournit pas toutes les fonctionnalités. Par exemple, il n'est pas possible de passer des appels Put conditionnels avec le modèle de persistance d'objet.

En savoir plus sur le fonctionnement d'AWS à l'aide du SDK .NET dans le Guide du développeur du SDK .NET .

Remarque: cette rubrique a été adaptée avec l'autorisation d'un article de blog publié à l'origine sur le blog AWS .NET SDK .

Examples

Exemple d'API de bas niveau

```
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));
```

Exemple de modèle de document

```
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()));
```

Exemple de modèle de persistance d'objet

Cet exemple comprend deux parties: premièrement, nous devons définir notre type de `Book` ; Deuxièmement, nous l'utilisons avec `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; }
}
```

Maintenant, utilisez-le avec `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));
```

Lire Utilisation d'AWS DynamoDb avec le SDK AWS .NET en ligne: https://riptutorial.com/fr/amazon-dynamodb/topic/5275/utilisation-d-aws-dynamodb-avec-le-sdk-aws--net

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec amazon-dynamodb	Community, Gustavo Tavares
2	Comment créer une table DynamoDB	Mike Dinescu, Parth Modi
3	Comment insérer des données dans une table à l'aide de DynamoDb?	Franck Dernoncourt
4	Dynamodb supprime les données au fil du temps	Lior
5	Opérations par lots: choses à savoir	SACHESH A C
6	Utilisation d'AWS DynamoDb avec le SDK AWS .NET	David Hale