DynamoDB

Lady Laureen van Schausberger, Dr. Fabian Hurnaus

PAPER

Nr. Thesis number not specified!

eingereicht am
Fachhochschul-Bachelorstudiengang

Mobile Computing

in Hagenberg

im Juli 2017

This paper was created as part of the course

DAB

during

Spring Semester 2017

Advisor:

DI Dr. Erik Sonnleitner

Contents

Chapter 1

Introduction

DynamoDB is a fast and flexible NoSQL database service provided by Amazon Web Services[1] and therefore is non-relational. It is a fully managed cloud service and supports both document and key-value store models. [1]

Without the user having to care for the management of his database, DynamoDB scales throughput capacity automatically to meet workload demands and partitions (and re-partitions) the data when the size table is growing, while synchronously replicating data across three facilities in an AWS Region, to enable high availability and data-durability.

Amazon DynamoDB makes it easy and cost-effective to store and retrieve any amount of data, while also serving any level of request traffic. Its data items are stored on solid-state drives providing high I/O performance, and can therefore efficiently handle high-scale requests. AWS user can interact with the service by using the AWS Management Console or a DynamoDB API.

This NoSQL database service allows documents, graphs and columnar among its data models. Its users can interact with it via GET and PUT queries after having stored their data in DynamoDB tables. In Addition, DynamoDB also supports basic CRUD operations and conditional operations.

DynamoDB can be used for nearly everything. For example for mobile, web, gaming, ad tech, IoT, etc.

[1] AWS is a comprehensive, evolving clout computing platform provided by Amazon.com, for further informations have a look at www.aws.amazon.com

Chapter 2

Main Features

Amazon DynamoDB aims to be cost-effective and to simplify the usage of scaling databases by taking care of the database-software-management and by providing hardware needed to run it. Because of this improvements, users should be able to create and deploy their non-relational databases in just a few minutes.

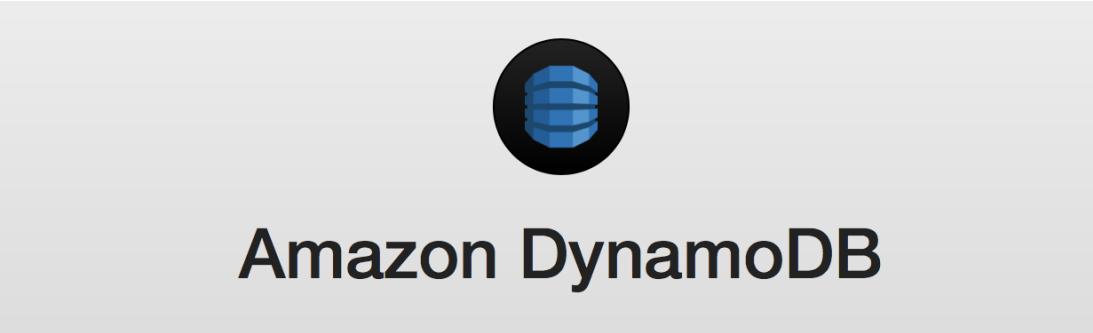

Figure 2.1: Amazon DyanamoDB logo

2.1 Performance

Average service-side latencies are typically single-digit milliseconds. When volumes increase further, DynamoDB will use automatic partitioning and SSD technologies to meet the upcoming requirements and will deliver low latencies at any scale.

2.2 Scalablility

By default, Auto Scaling is active for creating tables or global secondary indexes. Therefore, one will only need to specify the target utilization. Furthermore, the capacity is being scaled up or down automatically, depending on the application request volumes

status. CloudWatch alarms are always monitoring the throughput consumption, while the Users can see the scaling activities in real-time from the management console.

2.3 Flexibility

DynamoDB is known for supporting both document and key-value stored data structures. Therefore, users can choose how to design their architecture and work with what they prefer.

2.4 Management

DynamoDB is a fully managed cloud service. Users only have to create a database table, set the traget utilization for Auto Scaling and let the service do what it is best at: handling everything else.

2.5 Event driven programming

One can use AWS Lambda[1] to have the possibility to use Triggers, enabling architect applications that react automatically to changes in data.

2.6 Access control

Using AWS Identity and Access Management[2] with Amazon DynamoDB can allow fine-grained access control for users within one's organization.

[1] AWS Lambda is a serverless compute service, which runs one's code for virtually any type of application or backend service, while the user doesn't have to care for administration. It is provided by Amazon.com, for further informations have a look at www.aws.amazon.com/lambda

[2] IAM allows one to safely control their user-access to aws services and -resources. One can assign security credentials to each user and therefore control which access allowances are granted. It is provided by Amazon.com, for further informations have a look at www.aws.amazon.com/iam

Chapter 3

Usage

Figure 3.1: short selection companies using DyanamoDB

Figure 3.1 is listing some of the bigger companies, which are known for using DynamoDB for various projects.

As Amazon is promoting on its own website, DynamoDB can for example be used for **Ad Tech**. It is mentioned, that recommendation engines or realtime bidding platforms are easy to implement because of DynamoDB's consistent, single-digit millisecond latency, that is delivered at any scale.

When coming to Ad Tech, Amazon is referring to the company *VidRoll*, which is currently using this NoSQL database to help match hundreds of millions appropriate video ads (per month) with site visitors across 100.000 websites.[2]

Another example is **Gaming**: Responsive games for desktop, console or mobile devices can store and query Amazon Lumberyard[1] game data through Cloud Canvas[2].

[1] Amazon Lumberyard is a free cross-platform AAA game engine. It is provided by Amazon, for further information please have a look at www.aws.amazon.com/lumberyard

[2] Cloud Canvas is a suite of tools and solutions, that is designed to achieve an easy implementation of cloud-connected features using on-demand, global storage and compute provided by AWS. For further information please have a look at www.docs.aws.amazon.com/lumberyard/latest/developerguide/cloud-canvas-intro.html

DynamoDB can easily integrate with the AWS Mobile SDK[3] and other AWS services, for example for user authentication, social features or downloadable content.

A good example (and nice to mention) is *Zynga Poker*, which just recently moved a MySQL farm over to Amazon DynamoDB and therefore reduced their operational overhead drastically.[2]

Canary for example, is using AWS' scalability to support more than 150 million incoming videos daily within its home security systems. DynamoDB lets you connect your high-velocity, high-volume **IoT** data to a Amazon Redshift[4]data warehouse to enable BI analysis.[2]

Of course, Amazon likes to tell how amazing DynamoDB is in nearly every use case. But because one can not simply rely on one company's own marketing information, let's have a look at some real life use cases from multiple websites and different people:

3.1 Real Life Use Cases

Michael L.
works at The Internet
studied at Youtube
thinks he is funny..

Says, that instead of storing php sessions on the server's local file system, a good use case would be to use DynamoDB when having a php site running on multiple ec2 servers managed by an Elastic Load Balancer.[3]

Camille Emefa Acey
works at Clubhouse Software Inc.

Camille tells, that at Clubhouse (www.clubhouse.io), they use DynamoDB as a scalable backend for their Datomic database and DynamoDB Local in their local development environments.[3]

Ben Darfler
Systems Builder

Mentions, that at Localytics they decided to go with DynamoDB because they need to process data just once. And only once.[3]

[3]AWS Mobile SDK can help building app quickly and easily and has an easy access to many AWS services. It is available for iOS 8+, Android/Fire OS, Xamarin, Reactive Native, and many more. It is provided by Amazon, for further information please have a look at www.aws.amazon.com/mobile/sdk

[4]Amazon Redshift is a fast, simple, fully-managed and cost-effective data warehouse. t is provided by Amazon, for further information please have a look at www.aws.amazon.com/redshift

Chapter 4

How it works

All concepts described in this chapter are referred by [5].

4.1 Core Components

In DynamoDB, tables, items and attributes are the core components to work with. A table is a collection of items, and each item is a collection of attributes. These core components are very similar to other database systems.

Figure 4.1: This figure shows a table named *People* with some example items and attributes. [5]

4.1.1 Primary Key

When creating a table a primary key **must** be specified. It uniquely identifies an item of a table, therefore it is not allowed that multiple items have the same primary key. Each primary key attribute must be a scalar type. (See Section 4.1.2.)

Partition key

The **partition key** is a simple primary key composed of one attribute. DynamoDB will create a hash out of the key's value. This hash decides on which partition the item will be stored.

Partition key and sort key

In contrast to exclusive partition key (Section 4.1.1) it is possible that multiple items have the same primary key but must have a different sort key value. All items with the same partition key are stored together, in sorted order by the specified sort key value.

Secondary Indexes

A secondary index lets you query the data in the table using an alternate key, in addition to queries against the primary key. This can improve the performance of specific queries. Indexes are maintained automatically during add, update or delete of an item.

DynamoDB Streams

4.1.2 Data Types

DynamoDB offers a lot of different data types. These data types can be categorised as follows:

- Scalar Types
- Document Types
- Set Types

Scalar Types

A scalar type can represent exactly one value. Possible scalar types are: **number**, **string**, **binary**, **boolean** and **null**.

Note: DynamoDB does not offer a special data type for dates. To represent a date you either have to use a string (e.g. ISO 8601 string) or a number (e.g. epoch time).

Document Types

There are three document types: lists, maps and sets. These types can be nested within each other, to represent complex data structures up to 32 levels deep.

List A list type attribute can store an ordered collection of values. A list is very similar to a JSON array.

```
1  Calculation: ["sum", "*", 4, "/", "3.939"]
```

Listing 4.1: Example of a DynamoDB list

Map A map type attribute can store an unordered collection of key-value pairs. A map is very similar to a JSON object.

```
1  {
2    User: "fahu",
3    Date: "2017-06-28",
4    VisistedWebsites: ["https://medium.com", "https://derstandard.at", "http://delta
     -xi.net/"]
5  }
```

Listing 4.2: Example of a DynamoDB map

Sets DynamoDB supports types that represent **sets** of **numbers**, **strings** and **binary** values. All elements of a set must be of the same data type. Each value within a set must be unique.

```
1  ["black", "blue", "red, "green", "yellow"]
```

Listing 4.3: Example of a DynamoDB set

Chapter 5

Queries

5.0.1 ALTER

Description

Changes read/write throughput on table.
If not both but only one of the throughput values should be changed, "0" or "*" must be used.

Synopsis

```
ALTER TABLE tablename
    SET [INDEX index] THROUGHPUT throughput
ALTER TABLE tablename
    DROP INDEX index [IF EXISTS]
ALTER TABLE tablename
    CREATE GLOBAL [ALL|KEYS|INCLUDE] INDEX global_index [IF NOT EXISTS]
```

- **tablename**
  name of table
- **throughput**
  read/write throughput (read_throughput, write_throughput)
- **index**
  name of global index

Examples

```
ALTER TABLE foobars SET THROUGHPUT (4, 8);
ALTER TABLE foobars SET THROUGHPUT (7, *);
ALTER TABLE foobars SET INDEX ts-index THROUGHPUT (5, *);
ALTER TABLE foobars SET INDEX ts-index THROUGHPUT (5, *);
ALTER TABLE foobars DROP INDEX ts-index;
ALTER TABLE foobars DROP INDEX ts-index IF EXISTS;
ALTER TABLE foobars CREATE GLOBAL INDEX ('ts-index', ts NUMBER, THROUGHPUT (5, 5));
ALTER TABLE foobars CREATE GLOBAL INDEX ('ts-index', ts NUMBER) IF NOT EXISTS;
```

5.0.2 ANALYZE

Description

Can prefix any query reading or writing data.
Will also print out how much capacity was consumed at every part of query.

Synopsis

```
1 ANALYZE query
```

Examples

```
1 ANALYZE SELECT * FROM foobars WHERE id = 'a';
2 ANALYZE INSERT INTO foobars (id, name) VALUES (1, 'dsa');
3 ANALYZE DELETE FROM foobars KEYS IN ('foo', 'bar'), ('baz', 'qux');
```

5.0.3 CREATE

Description

Creates new table. Must have:

- exactly one hash key
- zero or one range keys (mandatory if having local indexes)
- up to five local indexes
- up to five global indexes

Synopsis

```
1 CREATE TABLE
2     [IF NOT EXISTS]
3     tablename
4     attributes
5     [GLOBAL [ALL|KEYS|INCLUDE] INDEX global_index]
```

- **IF NOT EXISTS**
  don't through an exception if table already exists
- **tablename**
  name of table you want to alter
- **attributes**
  attribute declaration like *name datatype* [*keytype*].
  Available data types: STRING, NUMBER, BINARY
  Available key types : $HASH\_KEY$, $RANGE\_KEY$, $[ALL|KEYS|INCLUDE]$ $INDEX(name)$
- **global_index**
  global index for table

Examples

```
CREATE TABLE foobars (id STRING HASH KEY);
CREATE TABLE IF NOT EXISTS foobars (id STRING HASH KEY);
CREATE TABLE foobars (id STRING HASH KEY, foo BINARY RANGE KEY,
                      THROUGHPUT (1, 1));
CREATE TABLE foobars (id STRING HASH KEY,
                      foo BINARY RANGE KEY,
                      ts NUMBER INDEX('ts-index'),
                      views NUMBER INDEX('views-index'));
CREATE TABLE foobars (id STRING HASH KEY, bar STRING) GLOBAL INDEX
                     ('bar-index', bar, THROUGHPUT (1, 1));
CREATE TABLE foobars (id STRING HASH KEY, baz NUMBER,
                      THROUGHPUT (2, 2))
                      GLOBAL INDEX ('bar-index', bar STRING, baz)
                      GLOBAL INCLUDE INDEX ('baz-index', baz, ['bar'], THROUGHPUT
    (4, 2));
```

5.0.4 DELETE

Description

Deletes items from table.

Synopsis

```
DELETE FROM
    tablename
    [ KEYS IN primary_keys ]
    [ WHERE expression ]
    [ USING index ]
    [ THROTTLE throughput ]
```

- **tablename**
  name of table
- **primary_keys**
  list of primary keys to the items to be deleted
- **expression**
  see SELECT for details about the WHERE clause, Chapter 5.0.11
- **index**
  when WHERE uses an indexed attribute, it allows you to specify an index name for the query
- **THROTTLE**
  limit the amount of throughput this query can consume
  $(read\_throughput, write\_throughput)$
  i.e. $(20\%, 50\%)$ or $(20, 50)$

Examples

```
1 DELETE FROM foobars; -- This will delete all items in the table!
2 DELETE FROM foobars WHERE foo != 'bar' AND baz >= 3;
3 DELETE FROM foobars KEYS IN 'hkey1', 'hkey2' WHERE attribute_exists(foo);
4 DELETE FROM foobars KEYS IN ('hkey1', 'rkey1'), ('hkey2', 'rkey2');
5 DELETE FROM foobars WHERE (foo = 'bar' AND baz >= 3) USING baz-index;
```

5.0.5 DROP

Description

Deletes table and all of its items.
Should be used carefully!!

Synopsis

```
1 DROP TABLE
2     [ IF EXISTS ]
3     tablename
```

- **IF EXISTS**
  don't raise an exception if table doesn't exist
- **tablename**
  name of table

Examples

```
1 DROP TABLE foobars;
2 DROP TABLE IF EXISTS foobars;
```

5.0.6 DUMP

Description

Prints out the matching CREATE statements for tables.

Synopsis

```
1 DUMP SCHEMA [ tablename [, ...] ]
```

- **tablename**
  name of table(s) whose schema should be dumped.
  will dump every table schemas by default if no tablename is given.

Examples

```
1 DUMP SCHEMA;
2 DUMP SCHEMA foobars, widgets;
```

5.0.7 EXPLAIN

Description

Meta-query, which is going to print out debug informations.

Synopsis

```
1 EXPLAIN query
```

Examples

```
1 EXPLAIN SELECT * FROM foobars WHERE id = 'a';
2 EXPLAIN INSERT INTO foobars (id, name) VALUES (1, 'dsa');
3 EXPLAIN DELETE FROM foobars KEYS IN ('foo', 'bar'), ('baz', 'qux');
```

5.0.8 INSERT

Description

Inserts data into table.

Synopsis

```
1 INSERT INTO tablename
2     attributes VALUES values
3     [ THROTTLE throughput ]
4 INSERT INTO tablename
5     items
6     [ THROTTLE throughput ]
```

- **tablename**
  name of table
- **attributes**
  comma-seperated list of attributes
- **values**
  comma-seperated list of data, that should be inserted
  must contain same number of items as the attributes param $var[, var]...$

- **items**
  comma-seperated key-value pairs, that should be inserted
- **THROTTLE**
  limit the amount of throughput this query can consume
  $(read\_throughput, write\_throughput)$
  i.e. $(20\%, 50\%)$ or $(20, 50)$

Examples

```
1 INSERT INTO foobars (id) VALUES (1);
2 INSERT INTO foobars (id, bar) VALUES (1, 'hi'), (2, 'yo');
3 INSERT INTO foobars (id='foo', bar=10);
4 INSERT INTO foobars (id='foo'), (id='bar', baz=(1, 2, 3));
```

5.0.9 LOAD

Description

Takes result of $SELECT...SAVE\ outfile$ and inserts all the records into table.

Synopsis

```
1 LOAD filename INTO tablename
2     [ THROTTLE throughput ]
```

- **filename**
  file containing records to upload
- **tablename**
  name of table
- **THROTTLE**
  limit the amount of throughput this query can consume
  $(read\_throughput, write\_throughput)$
  i.e. $(20\%, 50\%)$ or $(20, 50)$

Examples

```
1 LOAD archive.p INTO mytable;
2 LOAD dump.json.gz INTO mytable;
```

5.0.10 SCAN

Description

Is the exact same as a SELECT statement with the difference that it's allowed to perform table scans.

See Section 5.0.11

5.0.11 SELECT

Description

Query table for items.

Synopsis

```
SELECT
    [ CONSISTENT ]
    attributes
    FROM tablename
    [ KEYS IN primary_keys | WHERE expression ]
    [ USING index ]
    [ LIMIT limit ]
    [ SCAN LIMIT scan_limit ]
    [ ORDER BY field ]
    [ ASC | DESC ]
    [ THROTTLE throughput ]
    [ SAVE filename]
```

- **CONSISTENT**
  if present, perform strongly consistent read
- **attributes**
  Comma-separated list of attributes to fetch or expressions
  $TIMESTAMP$ and $DATE$ functions can be used, as well as arithmetic $(foo + (bar - 3)/100)$
  $SELECT*$ means 'all attributes'
  $SELECTcount(*)$ will return the number of results
- **tablename**
  name of table
- **index**
  When WHERE expression uses an indexed attribute, this allows you to manually specify which index name to use for the query
- **limit**
  maximum number of items to return
- **scan_limit**
  maximum number of items for DynamoDB to scan
- **ORDER BY**
  sorts results by field
- **ASC | DESC**
  sorts results in ASCending (default) or DESCending order
- **THROTTLE**
  limit the amount of throughput this query can consume
  $(read\_throughput, write\_throughput)$
  i.e. $(20\%, 50\%)$ or $(20, 50)$

- **SAVE**
  saves results to a file

Examples

```
SELECT * FROM foobars SAVE out.p;
SELECT * FROM foobars WHERE foo = 'bar';
SELECT count(*) FROM foobars WHERE foo = 'bar';
SELECT id, TIMESTAMP(updated) FROM foobars KEYS IN 'id1', 'id2';
SELECT * FROM foobars KEYS IN ('hkey', 'rkey1'), ('hkey', 'rkey2');
SELECT CONSISTENT * foobars WHERE foo = 'bar' AND baz >= 3;
SELECT * foobars WHERE foo = 'bar' AND attribute_exists(baz);
SELECT * foobars WHERE foo = 1 AND NOT (attribute_exists(bar) OR contains(baz, 'qux'
    ));
SELECT 10 * (foo - bar) FROM foobars WHERE id = 'a' AND ts < 100 USING ts-index;
SELECT * FROM foobars WHERE foo = 'bar' LIMIT 50 DESC;
SELECT * FROM foobars THROTTLE (50%, *);
```

WHERE Clause

if provided, the SELECT operation will use these constraints as the KeyConditionExpression if possible, and if not (or if there are constraints left over), the FilterExpression.

5.0.12 UPDATE

Description

Updates items in table.

Synopsis

```
UPDATE tablename
    update_expression
    [ KEYS IN primary_keys ]
    [ WHERE expression ]
    [ USING index ]
    [ RETURNS (NONE | ( ALL | UPDATED) (NEW | OLD)) ]
    [ THROTTLE throughput ]
```

- **tablename**
  name of table
- **RETURNS**
  return items that were operated on
- **THROTTLE**
  limit the amount of throughput this query can consume

$(read\_throughput, write\_throughput)$
i.e. $(20\%, 50\%)$ or $(20, 50)$

- **SAVE**
  saves results to a file

Examples

```
1 UPDATE foobars SET foo = 'a';
2 UPDATE foobars SET foo = 'a', bar = bar + 4 WHERE id = 1 AND foo = 'b';
3 UPDATE foobars SET foo = if_not_exists(foo, 'a') RETURNS ALL NEW;
4 UPDATE foobars SET foo = list_append(foo, 'a') WHERE size(foo) < 3;
5 UPDATE foobars ADD foo 1, bar 4;
6 UPDATE foobars ADD fooset (1, 2);
7 UPDATE foobars REMOVE old_attribute;
8 UPDATE foobars DELETE fooset (1, 2);
```

Chapter 6

Limitations

6.1 Data Types

When looking at the table below it is very clear to see that the maximum size for storing objects 400 KB is. Therefore it is not possible to save large objects (BLOB) to the database. If it is necessary to save large objects you have to use Amazon S3 storage which is comparatively slow. To increase the search speed it is possible to save the meta information of the object to DynamoDB.

Table 6.1: Data type limitations

Data Type	Maximum Size
String	400 KB
Number	38 digits
Binary	400 KB
Partition Key	2048 bytes
Sort Key	1024 bytes

6.2 Scalability

Scaling DynamoDB is very is easy possible via the AWS Console. But it's hard to find the right scale. Throughput should meet the needs of the users and your pocketbook. The process of finding the correct configuration for the right scale can be plodding and exhaustive.

6.3 Complex queries

DynamoDB is very good at high throughput reads and writes with 10ms response time on simple queries. But when it comes to more complex queries it reaches its limits very fast. To be able to execute more complicate queries it is necessary to add secondary indexes but these are limited to 10 GB on data under a single key. Reaching this boundary can bring the system to halt. [6]

Chapter 7

Installation

DynamoDB is intended to be run via the Amazon AWS console. Though it is also possible to it on a self hosted environment. In this chapter we will take a look on both possibilities.

7.1 Hosted by AWS

Hosting DynamoDB via Amazon AWS is for sure the most convenient way to go. There is nearly zero configuration required to set up a DynamoDB table. This also concludes that there is not a whole team of developers and devops required to setup and maintain the database which can save a company a lot of money.

7.1.1 Creating AWS Account

In order to host DynamoDB in AWS it is necessary to create an AWS account. To do so go to https://aws.amazon.com/. Click on register and follow the registration process.

After the registration you are done. No more steps are required to create the first DynamoDB table. You can go to the AWS console, select DynamoDB and click on create table.

7.2 Self-hosted

There is a downloadable version of DynamoDB which provides an executable .jar file. Which means that it will run on all platforms that are supported by Java.[4]

Note: Amazon does not mention if this version should be used in production. In their documentation they are only explaining that this version could be used local for offline development.

7.2.1 Downloading and Running DynamoDB

1. Download DynamoDB from docs.aws.amazon.com.
2. After the download is complete, extract the contents.

3. To start DynamoDB you need to open a command prompt window, navigate to its folder and execute the following command

```
1 java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -sharedDb
2
```

4. Congratulations! DynamoDB is successfully running on your machine.

Chapter 8

Demo Application

In order to get a feeling on how to use DynamoDB we will demonstrate it by an 'real-life application' which is attached to this project. In this section a useful use-case for DynamoDB will demonstrated. Additionally it will be demonstrated that how to read and write data.

8.1 Assumption

A small company in Upper Austria is producing and selling GPS tracker for pets. These GPS tracker send a so called 'PositionReport' containing the ID of the device, a timestamp, longitude, latitude and the altitude every second. The company wants to show its users the distance covered by the pet (and therefore the tracker) on a certain range. In order to display the data we need to implement a REST service to retrieve and save data. For the sake of convenience the REST service will be represented by a simple Java console application.

8.2 Project Setup

8.2.1 Create Table in AWS

Go to you AWS console and create a new DynamoDB table. The name of the table will be 'PositionReport'. The partition key of the table will be called 'TrackerId'. Additionally we add a sort key containing the timestamp of the report.

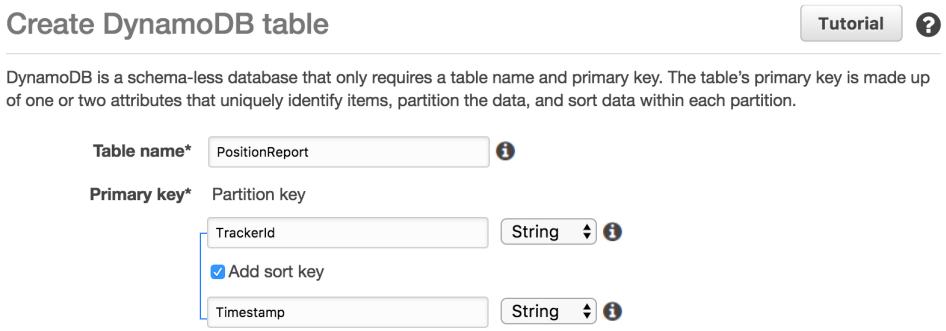

Figure 8.1: Creating a table in AWS.

After creating the table it takes some time until the table is ready to use.

8.2.2 Adding the Credentials

There are different ways to add the credentials for the DynamoDB. For more details on how to add the credentials go to http://docs.aws.amazon.com/sdk-for-java/v2/developer-guide/credentials.html.

8.2.3 Adding dependencies to project

In order to use the DynamoDB API it is necessary to to install the AWS SDK and the DynamoDB SDK. These dependencies can be loaded by using a dependency manager like Apache Maven (see the following listing).

```
<dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>com.amazonaws</groupId>
                <artifactId>aws-java-sdk-bom</artifactId>
                <version>1.11.106</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-dynamodb</artifactId>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

As soon as these dependencies are loaded you're ready to start developing.

8.3 Using the API

8.3.1 Connecting to AWS DynamoDB

Before trying to connect to DynamoDB make sure that the credentials are added correctly.

```
DynamoDB dynamoDB = new DynamoDB(new AmazonDynamoDBClient(
  new ProfileCredentialsProvider())
);
```

8.3.2 Read operation

To be able to read a certain item you must enter the primary key. If a table also contains a sort key the sort key must also be entered. By specifying a 'projection expression' only certain attributes of a table will be selected.

```
// select  table
Table table = dynamoDB.getTable(positionReportTableName);

// read item
Item item = table.getItem(
  "TrackerId",                    // primary key name
    "ABCDEFGH",                     // primary key value
    "Timestamp",                    // sort key name
    "2017-07-14T16:34:00.638Z",     // sort key value
    "TrackerId, Altitude",          // projection expression
    null
);

System.out.println("GetItem: printing results...");
System.out.println(item.toJSONPretty());
```

8.3.3 Write operation

The write operation is very straight forward. First of all it is necessary to select the table to write an item. Afterwards the item which should be written is created. With the method 'putItem()' the item is written to the database.

```
// select  table
Table table = dynamoDB.getTable("PositionReport");

try {
    // create item
    Item item = new Item()
        .withPrimaryKey("TrackerId", "ABCDEFGH")
        .withString("Timestamp", dateFormatter.format(new Date()))
        .withNumber("Longitude", 2)
      .withNumber("Latitude", 500)
      .withNumber("Altitude", 500);

  // save item
  table.putItem(item);

} catch (Exception e) {
```

```
17   System.err.println("Failed to create item in " + tableName);
18   System.err.println(e.getMessage());
19 }
```

8.3.4 DynamoDB Object Mapper

Since it is very inconvenient, error-prone and time-consuming to always create items by specifying its attribute names the DynamoDB SDK comes with an object mapper. This object mapper allows to add annotations to a POJO. In these annotations the attribute names can be specified.

```
1 @DynamoDBTable(tableName="ProductCatalog")
2 public class PositionReport {
3
4     @DynamoDBIndexHashKey(attributeName="Id")
5     public String getTrackerId() { return trackerId; }
6     public void setTrackerId(String _trackerId) { trackerId = _trackerId; }
7
8     @DynamoDBIndexRangeKey(attributeName = "Timestamp")
9     public String getTimestamp() { return timestamp; }
10    public void setTimestamp(String _timestamp) { timestamp = _timestamp; }
11
12    @DynamoDBAttribute(attributeName = "Altitude")
13    public double getAltitude() { return altitude; }
14    public void setAltitude(double _altitude) { altitude = _altitude; }
15
16 ...
17 }
```

References

Literature

[1] URL: http://searchaws.techtarget.com/definition/Amazon-Dynamo-Database-DDB (cit. on p. 1).

[2] URL: https://aws.amazon.com/dynamodb/?nc1=f_ls (cit. on pp. 4, 5).

[3] URL: https://www.quora.com/What-are-some-real-world-uses-of-Amazon-DynamoDB (cit. on p. 5).

[4] Amazon. *Setting Up DynamoDB Local (Downloadable Version)*. http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DynamoDBLocal.html. URL: http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DynamoDBLocal.html (visited on 06/28/2017) (cit. on p. 19).

[5] AmazonWebServices. *Amazon DynamoDB Developer Guide*. Amazon, 2013 (cit. on p. 6).

[6] Jono MacDougall. *Scaling a Startup Using DynamoDB*. https://syslog.ravelin.com/scaling-a-startup-using-dynamodb-4d97b0843350. URL: https://syslog.ravelin.com/scaling-a-startup-using-dynamodb-4d97b0843350 (visited on 06/28/2017) (cit. on p. 18).