DynamoDB

Daniel Charvát, Denisa Šrámková, Viliam Juríček, Šimon Berka

Amazon DynamoDB

Introduction to DynamoDB

- **Key-value** document database developed by Amazon
  - They discovered that 90 % of their operations query a single table
  - SQL database tables was thus mostly redundant
- Fully managed, multimaster, durable database with in-memory caching
- Partition through **consistent hashing** to spread data across instance nodes
- Size is defined through **read and write capacity units**
  - Allowed number of operations per second
  - Generally cheaper with less frequent usage

Companies that are using DynamoDB

Source: https://www.featuredcustomers.com/vendor/amazon-dynamodb/customers

Ranking

Rank Dec 2019	Rank Nov 2019	Rank Dec 2018	DBMS	Database Model	Score Dec 2019	Score Nov 2019	Score Dec 2018
1.	1.	1.	Oracle	Relational, Multi-model	1346.39	+10.33	+63.17
2.	2.	2.	MySQL	Relational, Multi-model	1275.67	+9.38	+114.42
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1096.20	+14.29	+55.86
4.	4.	4.	PostgreSQL	Relational, Multi-model	503.37	+12.30	+42.74
5.	5.	5.	MongoDB	Document, Multi-model	421.12	+7.94	+42.50
6.	6.	6.	IBM Db2	Relational, Multi-model	171.35	-1.25	-9.40
7.	7.	↑ 8.	Elasticsearch	Search engine, Multi-model	150.25	+1.85	+5.55
8.	8.	↓ 7.	Redis	Key-value, Multi-model	146.23	+1.00	-0.59
9.	9.	9.	Microsoft Access	Relational	129.47	-0.60	-10.04
10.	10.	↑ 11.	Cassandra	Wide column	120.71	-2.52	-1.10
11.	11.	↓ 10.	SQLite	Relational	120.36	-0.66	-2.65
12.	12.	12.	Splunk	Search engine	90.53	+1.46	+8.34
13.	13.	↑ 14.	MariaDB	Relational, Multi-model	86.79	+1.22	+9.53
14.	14.	↑ 15.	Hive	Relational	86.05	+1.83	+18.67
15.	15.	↓ 13.	Teradata	Relational, Multi-model	78.49	-1.86	-0.67
16.	16.	↑ 21.	Amazon DynamoDB	Multi-model	61.63	+0.26	+7.33
17.	17.	↓ 16.	Solr	Search engine	57.22	-0.56	-4.13
18.	↑ 19.	↑ 20.	SAP Adaptive Server	Relational	55.55	+0.25	-0.27

Source: https://db-engines.com/en/ranking/

Features

Strengths

- Seamless scalability through automatic instance expansion
- Data backed up to Amazon S3
- Ease of integration with other AWS services

Drawbacks

- No ACID transactions
  - Although eventual consistency is almost guaranteed
- Not suitable for large binary objects.
- Cross-region replicability not available

Setting up DynamoDB

There are several options:

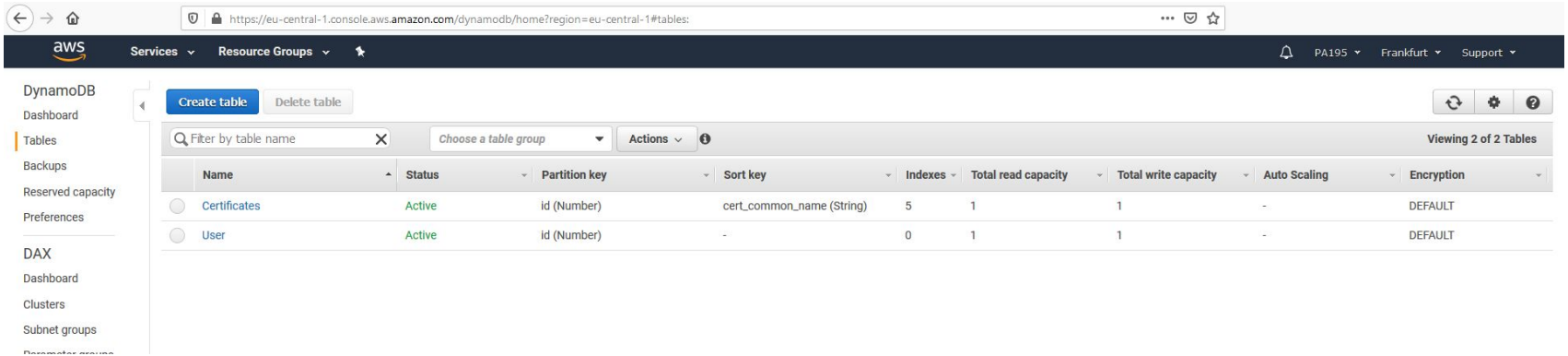
- Local
  - Windows, Linux, Mac OS *(downloadable version)*
  - Apache Maven *(POM file)*
  - Docker
- Web service
  1. Sign up to AWS
  2. Get an AWS access key
  3. Configure credentials

Accessing DynamoDB

```
denka@Denka-PC:~$ aws dynamodb list-tables
{
    "TableNames": [
        "Certificates",
        "User"
    ]
}
```

Again there are several options:

- AWS Management Console
- AWS Command Line Interface
- DynamoDB API – supports Java, JavaScript, .NET, Node.js, PHP, Python *(AWS SDK called Boto 3)*, Ruby, C++, Go, Android and iOS



Core components

- a **table** is a collection of **items**
- each **item** is a collection of **attributes**
- **primary keys** are used to uniquely identify each item in a table
- **secondary indexes** provide more querying flexibility

People


```
{
    "PersonID": 101,
    "LastName": "Smith",
    "FirstName": "Fred",
    "Phone": "555-4321"
}
```

```
{
    "PersonID": 102,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
        "Street": "123 Main",
        "City": "Anytown",
        "State": "OH",
        "ZIPCode": 12345
    }
}
```

```
{
    "PersonID": 103,
    "LastName": "Stephens",
    "FirstName": "Howard",
    "Address": {
        "Street": "123 Main",
        "City": "London",
        "PostalCode": "ER3 5K8"
    },
    "FavoriteColor": "Blue"
}
```


*Other than the primary key, the 'People' table is **schemaless**, which means that neither the attributes nor their data types need to be defined beforehand.*

Source: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html

Core components

*The **primary key** for table 'Music' **consists of two attributes** (Artist and SongTitle).*

Each item in the table must have these two attributes.

*The **combination** of Artist and SongTitle **distinguishes each item** in the table from all of the others.*

Music


```
{
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot",
    "AlbumTitle": "Hey Now",
    "Price": 1.98,
    "Genre": "Country",
    "CriticRating": 8.4
}
```

```
{
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The Road",
    "AlbumTitle": "Somewhat Famous",
    "Genre": "Country",
    "CriticRating": 8.4,
    "Year": 1984
}
```

```
{
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 2.47,
    "Genre": "Rock",
    "PromotionInfo": {
        "RadioStationsPlaying": [
            "KHCR",
            "KQBX",
            "WTNR",
            "WJJH"
        ],
        "TourDates": {
            "Seattle": "20150625",
            "Cleveland": "20150630"
        },
        "Rotation": "Heavy"
    }
}
```

```
{
    "Artist": "The Acme Band",
    "SongTitle": "Look Out, World",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 0.99,
    "Genre": "Rock"
}
```


Source: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html

Primary key

- must be specified when creating a table
- DynamoDB support two kinds of primary keys:
  - **partition key**: composed of one attribute *(PersonID)*
  - **partition key and sort key**: composed of two attributes *(Artist, SongTitle)*

- *each primary key attribute must be a scalar (of a data type: string, number, or binary)*

Source: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html

Secondary indexes

- we can **query the data** in the table **using an alternate key** (in addition to queries against the primary key)
- two kinds of indexes:
  - **global**: both partition and sort key can be different from those on the table
  - **local**: same partition key as the table, but a different sort key

- *default indexes limit per table: 20 global, 5 local*

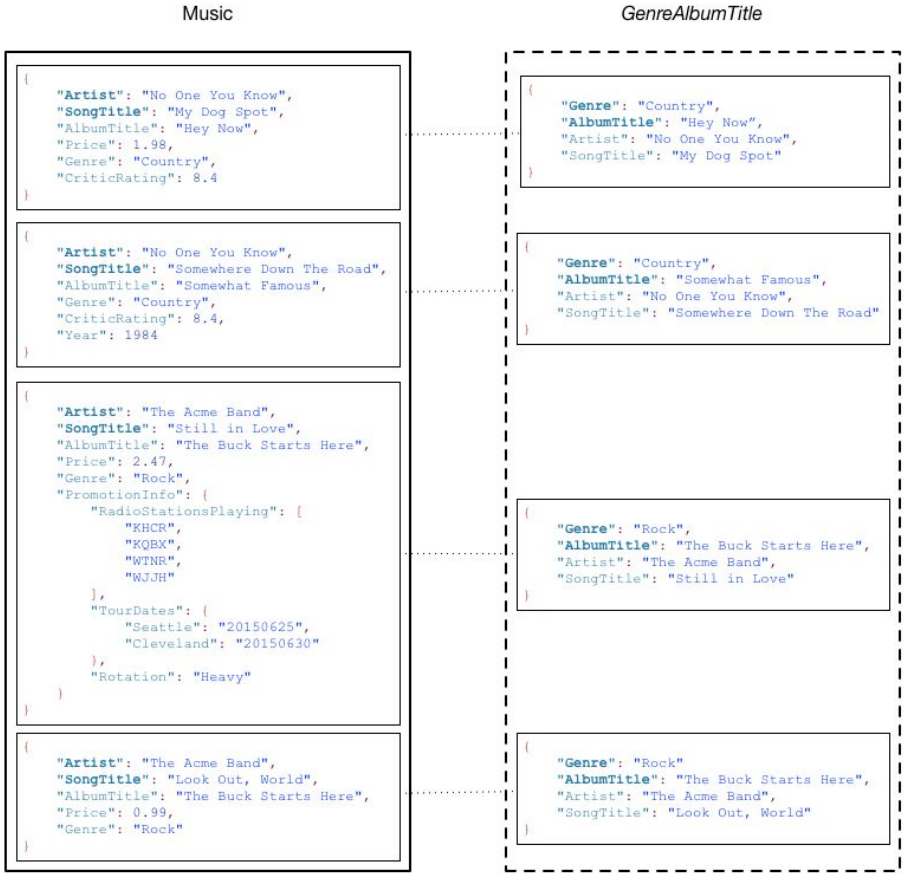
Source: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html

Secondary indexes

We can query data items by *Artist* (partition key) or by *Artist* and *SongTitle* (partition key and sort key).

If we also wanted to query the data by *Genre* and *AlbumTitle*:

1. Create an index on *Genre* and *AlbumTitle*
2. Query the index



Source: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html

RDBMS vs. DynamoDB

Characteristics	***Relational Database Management System (RDBMS)***	***Amazon DynamoDB***
Optimal Workloads	Ad hoc queries; data warehousing; OLAP	Web-scale applications
Data Model	**Requires** a well-defined **schema** (data is normalized into tables, rows, and columns).	**Schemaless** - Can manage structured or semistructured data.
Performance	**optimized for storage**	**optimized for compute**
Scaling	Scale up through faster hardware tables can be span across multiple hosts in a distributed system (**upper limits on scalability**).	Designed to scale out using distributed clusters of hardware (**No upper limit).**

Source: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SQLtoNoSQL.html

Creating a table – schema example

cert_info_link	not_valid_before	not_valid_after	cert_common_name	cert_authority	log_type
http://ct.google...	1573257600	1581119999	thesmartlocal0...	cPanel, Inc.	X509LogEntry

```
{
        "TableName": "TestCertificates",
        "KeySchema": [
                { "AttributeName": "cert_info_link", "KeyType": "HASH" },
                { "AttributeName": "cert_authority", "KeyType": "RANGE" }
        ],
        "GlobalSecondaryIndexes": [
                { "IndexName": "log_type_index",
                  "KeySchema": [
                        {
                                "AttributeName": "log_type",
                                "KeyType" : "HASH"
                        },
                        {
                                "AttributeName": "cert_authority",
                                "KeyType" : "RANGE"
                        }
                  ],
```

Creating a table – schema example

```
                "Projection": {
                        "ProjectionType": "ALL"
                },
                "ProvisionedThroughput" : {
                        "ReadCapacityUnits": 5,
                        "WriteCapacityUnits": 5
                }
               }
        ],
        "AttributeDefinitions": [
                { "AttributeName": "cert_info_link", "AttributeType": "S" },
                { "AttributeName": "cert_authority", "AttributeType": "S" },
                { "AttributeName": "log_type", "AttributeType": "S" }
        ],
        "ProvisionedThroughput": {
                "ReadCapacityUnits": 5,
                "WriteCapacityUnits": 5
        }
}
```

LIVE DEMO