Dynamo DB
:Data Model and Operations

pm_jat @ daiict

Amazon Dynamo DB

- Was proposed through Article [2] in 2007
- Was primarily created for (Amazon's) own use internally for their ecommerce applications!

Few quotes from the paper:

- Many applications "only store and retrieve data by primary key and do not require the complex querying and management functionality offered by an RDBMS."
- "Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an 'always-on' experience"
- "To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios."

Dynamo DB Data Model

- DynamoDB database has three core components
  - Table, Item, Attributes
- **Table is a collection of Items.** (distributed collection)

Correspondences with Relation DBs

RDB	DynamoDB
Table	Table
Row/Tuple	Item
Column/Attribute	Attribute

- Correspondences with Relational Databases is shown in table here.
- Items are basically data objects. Name "Items" may be bit confusing, read this as "**data item**" or "**data record**"
- Data Items are described by Attribute-Value pairs; can be in nested manner.

Dynamo DB Data Model

- Date Items are schema-less. Key (and Index) information is only schema dynamo tables have.
- Items are identified by Keys (Called as Primary Key)
- We can have secondary indexes on a table to allow querying on other (than primary key) attributes.

Key and Primary Key in RDB

- “Candidate Key” and “Primary Key” in Relational Database?
  - “Candidate Key” and Key are same (Super Key is superset of Key)
- “Candidate Key”: used to identify a row in a table. There can be multiple candidate keys.
- “Primary Key” is found in SQL-DDL (physical characteristic of a table)
  - One of CK (most suitable, decided by DBA) can be used as PK.
  - Data Organization is dictated by PK
  - Not Null constraint is included PK

Example Table

Note following

- Table named “People”
- Primary Key here is “PersonID”
- Other than the Primary Key the Table is “schema less” – that is, this is only the required attribute, other than this a item can have any attributes!

Table “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"
}
```

Example Table

Note following

- Data objects (Items) are divided in two parts
  - Key, and
  - Value
- In this case PersonID is the key where as rest of the attribute values are data.
- Attributes in value part can be nested – for example Address attribute has sub-attributes

Table “People”

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

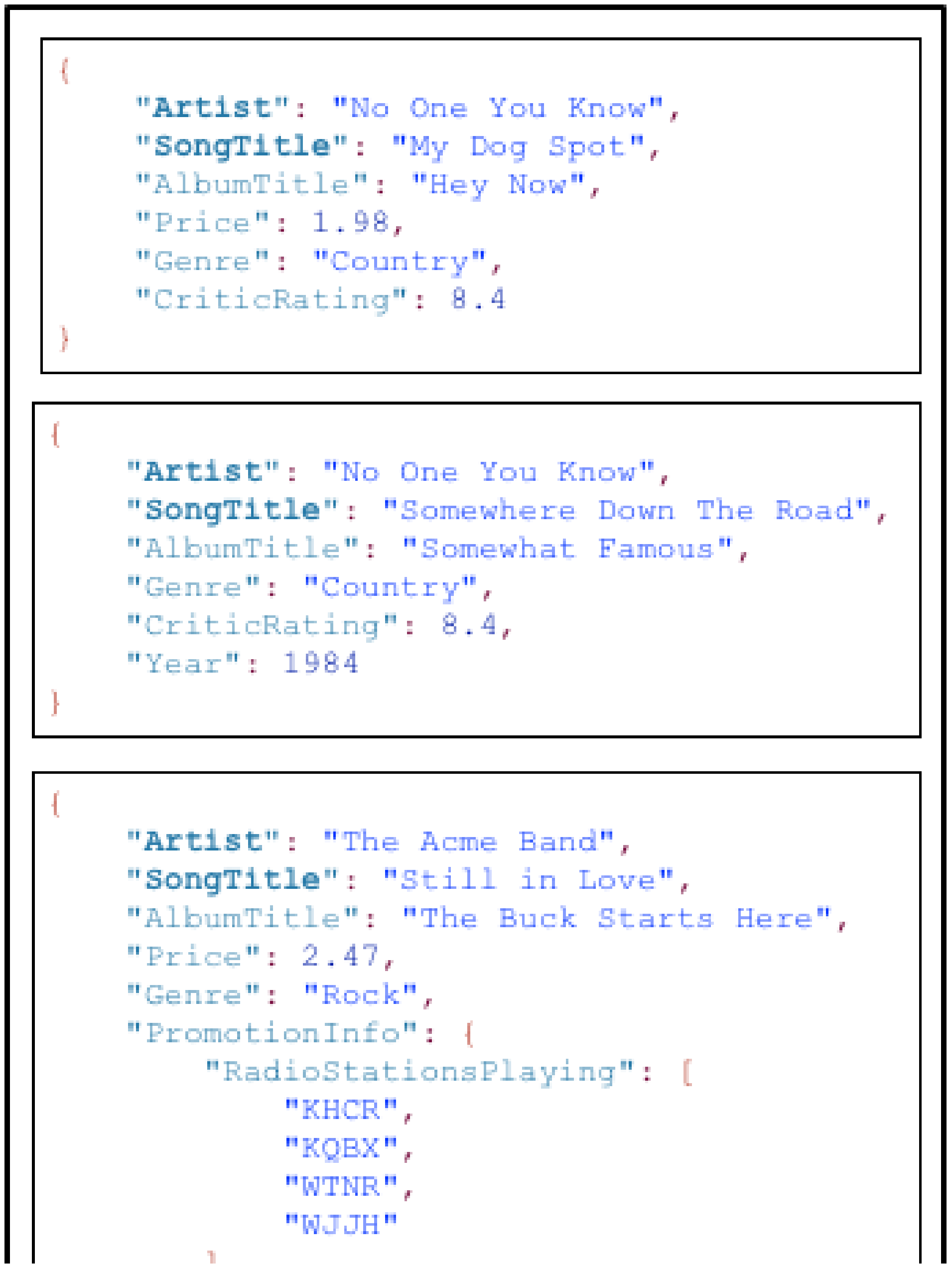
```
{
    "PersonID": 102,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
        "Street": "123 Main",
        "City": "Anytown",
        "State": "OH",
        "ZIPCode": 12345
    }
}
```

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

Example Table

- Another table Music
- Music table has composite key: {*Artist, SongTitle*}

Table “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"
        ]
```

Partitions and Data Distribution

- Amazon DynamoDB stores data in partitions.
- A partition is an allocation of storage for a table, and replicated.
- DynamoDB manages partitions very transparently.
- Global secondary indexes in DynamoDB are also composed of partitions.
- The data in a global secondary index is stored separately from the data in its base table, but index partitions behave in much the same way as table partitions.

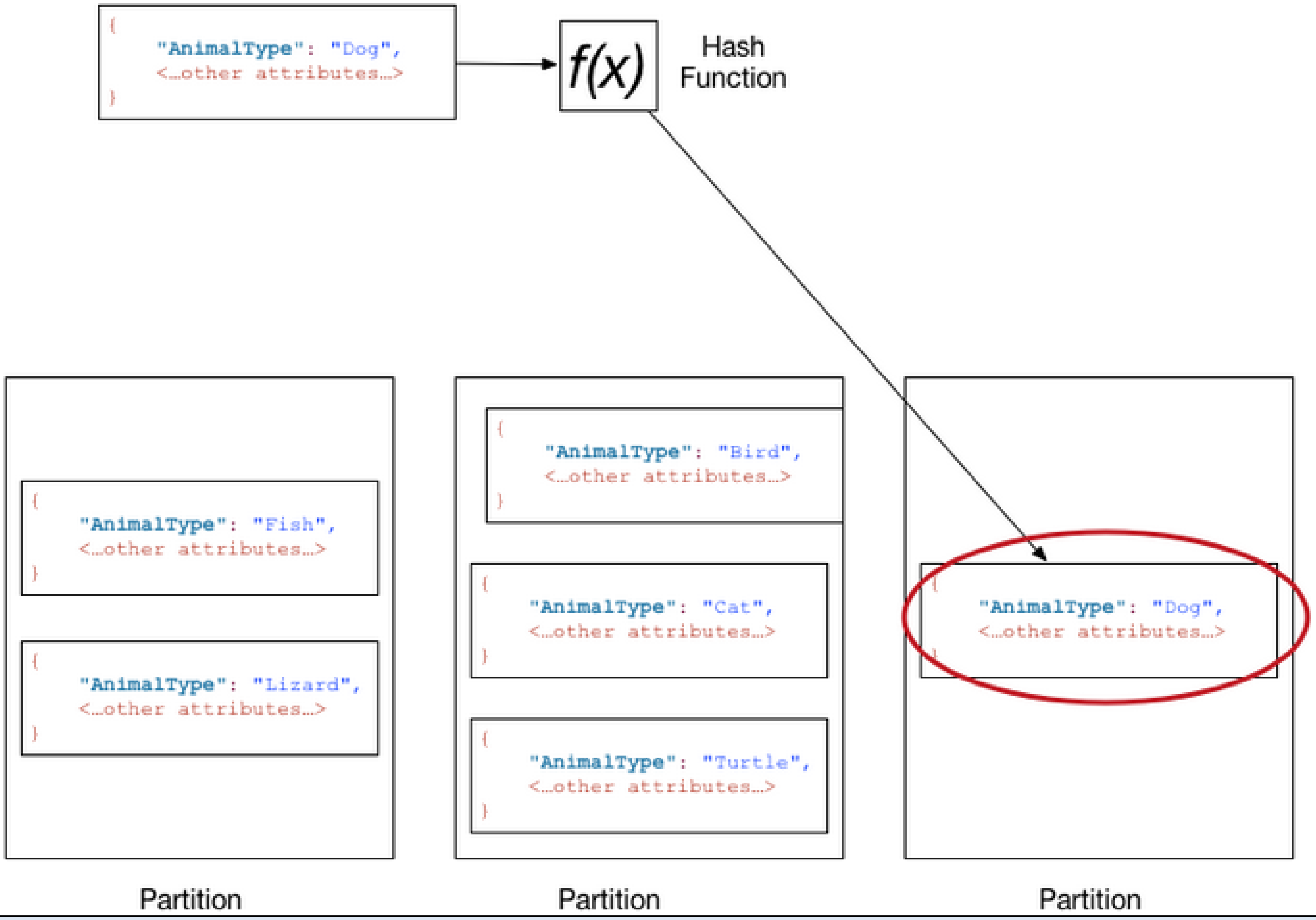
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.Partitions.html

Data Distribution: Partition Key

- Partition key determines which partition a data object is stored. "All data objects of same partition key value are stored on same partition"
- A "internal hash function" maps a partition key value to the partition.
- For tables with composite key also Partition key is used for determining partition. In this case there can be multiple data objects with same partition key value.
- All data objects of same partition key value are kept together and sorted on Sort Key.
- Figure on next slide represents a table "Pets" with Animal Type as partition key and Name as sort key.

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

Partitions and Data Distribution



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

Primary Key of Dynamo DB tables

- Primary Key is of two types
  - Partition key (PK alone)
  - “Partition key” and “Sort key” (Composite Key)

- BTW, what is difference between “key” and “primary key” in relational model?

Partition key (only) Primary Key

- A simple primary key, composed of one attribute known as the *partition key*.
- Dynamo DB uses the partition key's value as input to an internal hash function.
- The output from the hash function determines the “partition”, the physical storage in which the item will be stored.
- In a table that has only a partition key, no two items can have the same partition key value. (partition key unique)

Partition key and sort key PK

- It is a *composite primary key*, having two parts (attributes)
  - *partition key*, and
  - *sort key*.
- Partition key is used to determines the partition
  - In this case, there can be multiple items with same partition key in a partition
  - All data items are sorted by sort key value within a partition
- The *Music* table shown earlier as an example of a table with a composite primary key (*Artist* and *SongTitle*)

Uses of Primary Key

- All data access happens using Primary Key!
  - GET, PUT, DELETE, etc.
- In case of composite Key, “Range based search” on Sort Key attribute can also be specified.
  - Typically all values of “Hash Key” or “Partition Key” are on same partition!
- For example in case of Music table, we can search based on
  - An Artists – all titles of an artists will be returned
  - An artists plus a range of values for “SongTitle” can be specified

What is a index in databases?

- Index is map: Search-Key --> Location of Data Record
- Primary Index:
  - Primary Key used for.
  - Data are ordered as per PK. Can be Internal to the data file.
- Secondary Index: non-key attribute based. Non-unique. External to the data file.

Secondary Indexes in DynamoDB

- Hope you understand the difference between Primary and Secondary Indexes.
- Primary Index:
  - Index based on “Primary Key” is Primary Index.
  - Index that dictates (or used) Organization of Data Records on storage is also called primary index.
- Secondary Index:
  - Index on attributes other than primary key.
  - Basically a index stored externally, and has references to stored “data records” in a table.

Secondary Indexes in DynamoDB

- DynamoDB allows us creating one or more “Secondary Indexes” on a table.
- A *secondary index* allows query the data in the table using an “alternate key”, in addition to queries against the primary key.
- DynamoDB supports two kinds of indexes:

  Global secondary index – An index with a partition key and sort key that can be different from those on the table.

  Local secondary index – An index that has the same partition key as the table, but a different sort key.

- Make sure that you have understood “Partition Key” and “Sort Key”!

Secondary Indexes in DynamoDB

- “Partition Key”
  - Attribute that is used for “partitioning (distributing) the data” on several nodes
  - Partition key directly maps to “Data Node”
- “Sort Key”
  - Sort Key is used when we have non-unique primary key.
  - It is used to Sort the data records within the partition.
- For instance, we have secondary index on Music Table with “Gener” as Partition Key, and “Album Title” as sort Key. This means
  - Index entries are partitioned and sorted on “Gener” and “Album Title” respectively.

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

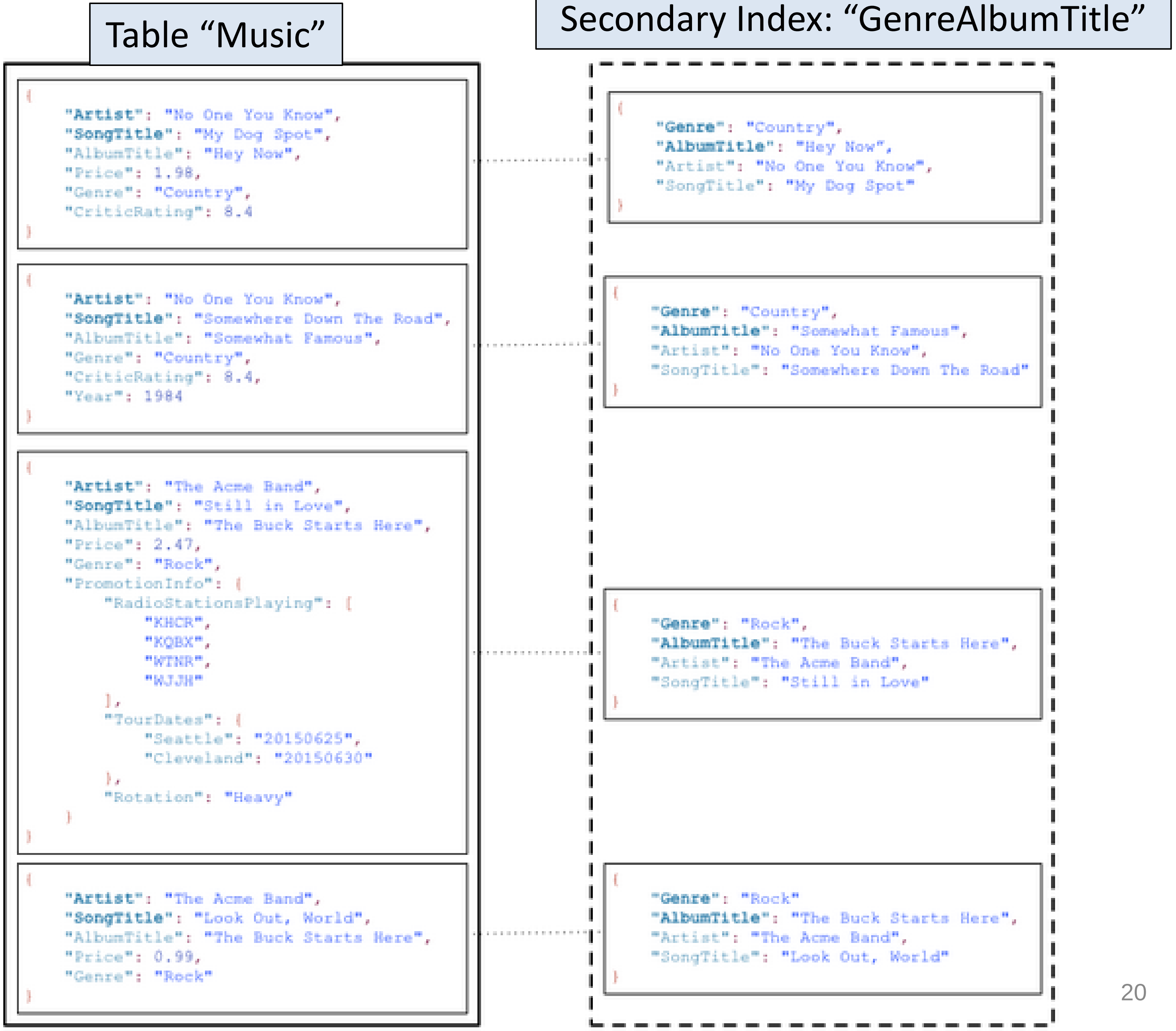


Table “Music”

```
{
    "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"
}
```

Secondary Index: “GenreAlbumTitle”

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

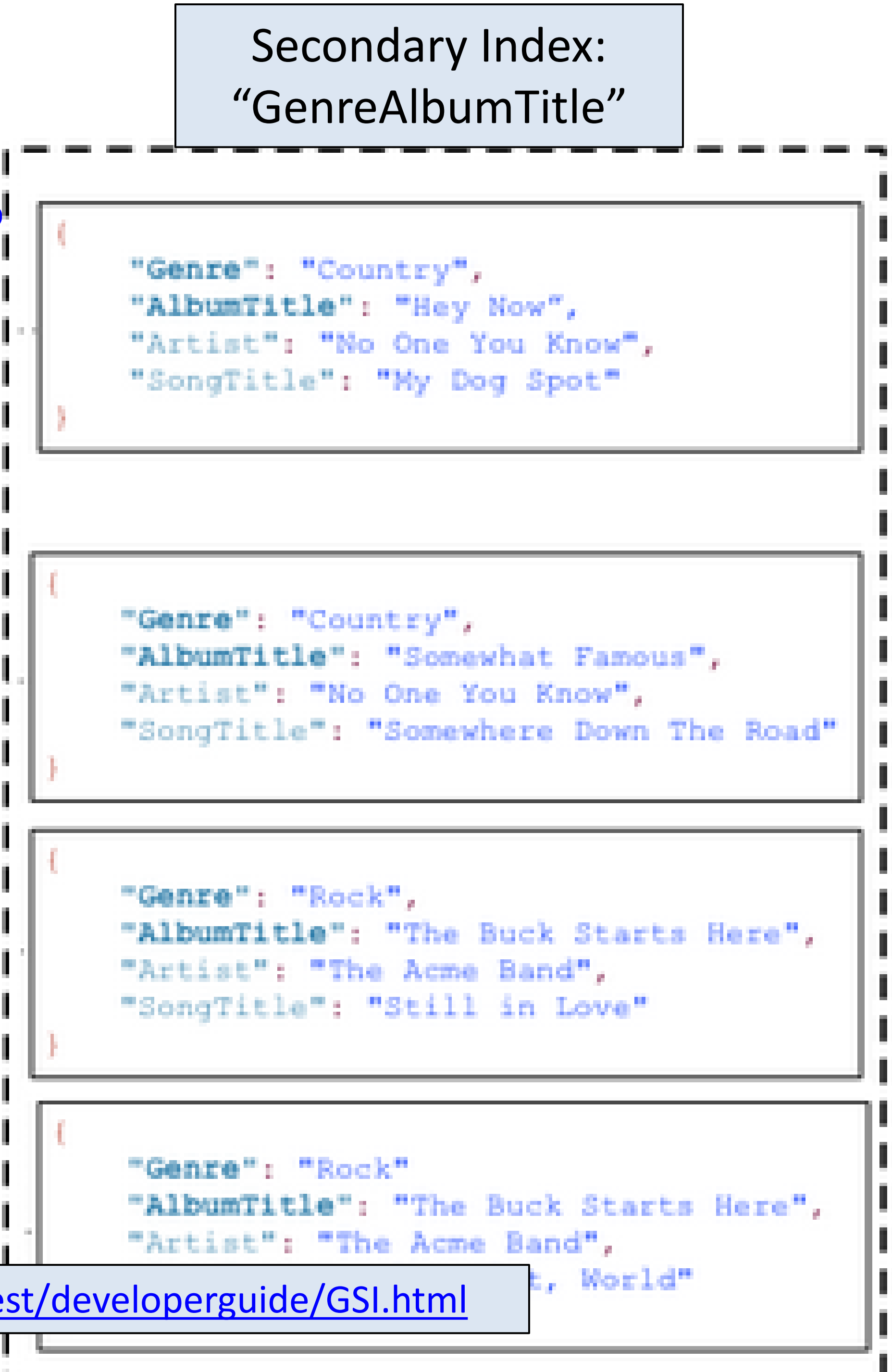
```
{
    "Genre": "Country",
    "AlbumTitle": "Somewhat Famous",
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The Road"
}
```

```
{
    "Genre": "Rock",
    "AlbumTitle": "The Buck Starts Here",
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love"
}
```

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

DynamoDB Secondary Indexes

- "Index keys" are additional "search keys"
- "Index values" are keys of data records "Items"
- Indexes belong to a table
- **DynamoDB maintains indexes automatically** –as we add, update, or delete data



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

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

```
{
    "Genre": "Rock",
    "AlbumTitle": "The Buck Starts Here",
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love"
}
```

```
{
    "Genre": "Rock"
    "AlbumTitle": "The Buck Starts Here",
    "Artist": "The Acme Band",
                            t, World"
```

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

Dynamo DB Data Model - summarize

- Dynamo uses a concept of Table.
- Table is collection of Data Items, dynamo calls it Item only
- Every table requires having primary key
- Primary Key consists of
  - Partition Key (HASH-KEY)
  - Optionally Sort Key (RANGE-KEY)
- Data items are partitioned (hashed) using partition key
- Sort key is used to order data within a partition
- Table is schema less (except having Key information)

Dynamo DB Data Model - summarize

- Primary Key is used as index to find the data storage location
  - Index is map: `search-key -> data-storage-location`
- We can define Secondary Index on a table.
- Secondary index typically contains “search-key” and “primary key” of the table (on which the index is)
- Secondary Index “search-key”, which is primary-key of index, also consists of
  - Partition Key (HASH-KEY)
  - Optionally Sort Key (RANGE-KEY)
- Index entries are partitioned (hashed) using partition key
- Sort key is used to order index entries within a partition

Dynamo DB Data Model - summarize

- Dynamo has two types of Secondary indexes:
  - Global Secondary Index, and
  - Local Secondary Index
- Global secondary index has different set of partition key and a sort key that can be different from those on the base table, where as
- Local secondary index has the same partition key as the base table, but a different sort key.
- They are local in the sense that partionally their search space is same is base table, they only differe on sort key, therefore their search can not go beyond partition (local)

Dynamo DB Data Model - summarize

- A typical characteristics of secondary indexes in dynamo is that they can have more attributes merely than primary key of the table. It allows specifying, one of following option
  - KEYS_ONLY
  - INCLUDE – we specify attributes that are to included
  - ALL– all attributes are copied in the index too (<u>in this case index in duplication of whole table with different search-key</u>)

Dynamo DB API

Dynamo DB Provides following operations

- To create and maintain tables, and indexes [data definition]
- To PUT data records into table; also update and overwrite options with predicates (only on Primary Key) [INSERT/UPDATE] [CREATE/UPDATE]
- To GET data record(s) from tables [READ]
- To DELETE data records from tables (on Primary Key) [DELETE]
- Popular term in database programming CRUD: CREATE, READ, UPDATE, DELETE

Working with DynamoDB

- For most business application, people use it from Amazon’s Services.
- Amazon also allows using DynamoDB from so called “free account”*.
- We can also create a local setup for learning purpose.
- In most cases, DynamoDB is accessed from programming languages (No SQL databases are made for that purpose only, you know?),
- However, it also provides a command line interface, called AWS CLI.

Working with DynamoDB

- Here is reference for AWS DynamoDB CLI commands https://docs.aws.amazon.com/cli/latest/reference/dynamodb/
- Dynamo DB also provides a “NoSQL Workbench for DynamoDB”, a GUI work around Dynamo DB databases.
- https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/workbench.html
- Workbench is supposed to be used to access any no-sql database for that matter but currently only available for cassandra!
- Here we shall be learning CLI and “Accessing from a programming language”. Let **CLI** be **first**!

Table Operations (data definition)

- Create Table – Creates a new table.
    - Also specifying indexes and so
- Describe Table– Returns information about a table, such as its primary key schema, throughput settings, and index information.
- List Tables – Returns the names of all of your tables in a list.
- Update Table – Modifies the settings of a table or its indexes, creates or removes new indexes on a table, and so forth.
- Delete Table – Removes a table and all of its dependent objects from DynamoDB.

Dynamo DB API – CREATE TABLE

- Before performing any operation, you require creating table.
- Parameters to create table are
  - Table Name
  - Definition of some attributes: attributes that are part of Keys, and Indexes.
  - Attribute Definition includes: Attribute Name and Data Type
  - Key Information
  - Storage Provisioning Information “Provisioned Throughput Capacity”

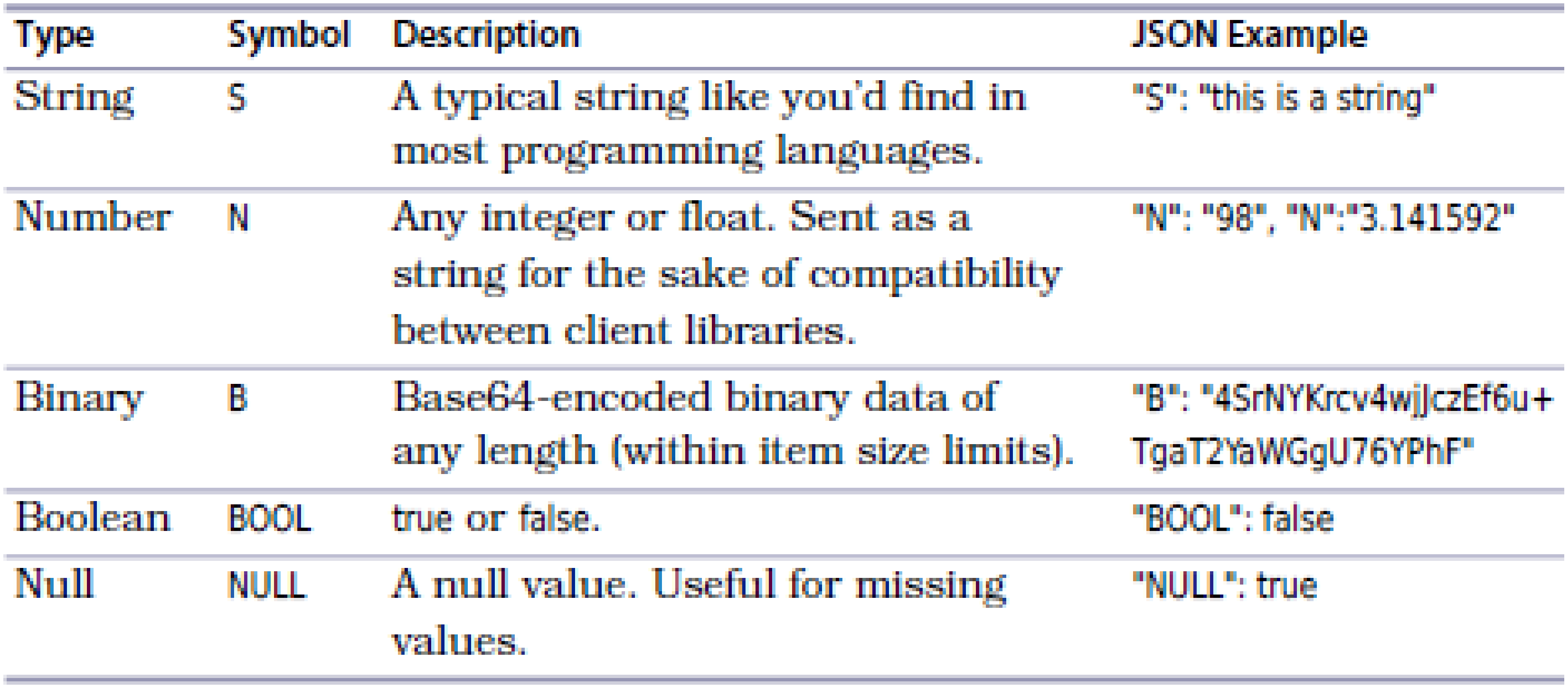
DynamoDB Data Types

- DynamoDB supports many different data types for attributes within a table. They can be categorized as follows:
- **Scalar Types** – A scalar type can represent exactly one value. The scalar types are Number, String, Binary, Boolean, and Null.
- **Document Types** – A document type can represent a complex structure with nested attributes, such as you would find in a JSON document. The document types are list and map.
- **Set Types** – A set type can represent multiple scalar values. The set types are string set, number set, and binary set.

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

DynamoDB data types [2]

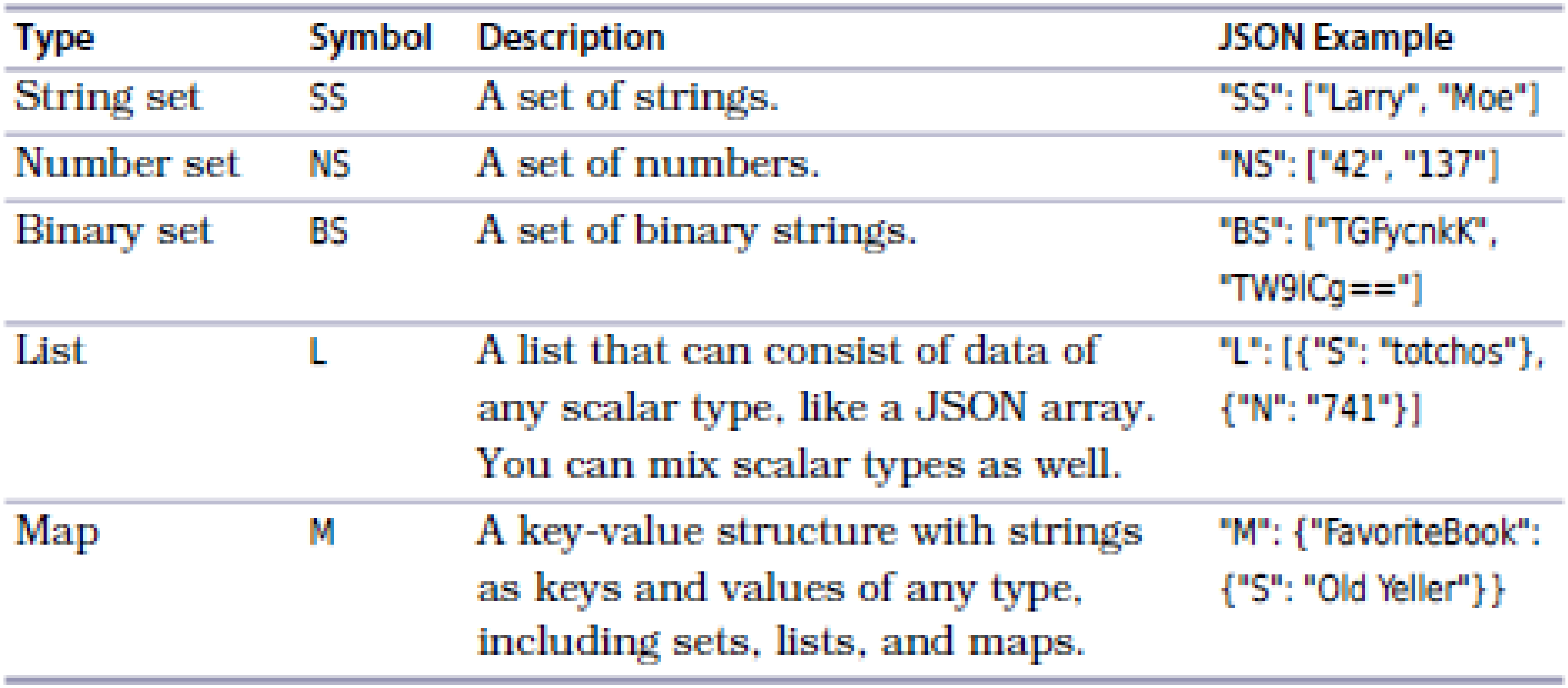
Scalar Types:



Type	Symbol	Description	JSON Example
String	S	A typical string like you'd find in most programming languages.	"S": "this is a string"
Number	N	Any integer or float. Sent as a string for the sake of compatibility between client libraries.	"N": "98", "N":"3.141592"
Binary	B	Base64-encoded binary data of any length (within item size limits).	"B": "4SrNYKrcv4wjJczEf6u+TgaT2YaWGgU76YPhF"
Boolean	BOOL	true or false.	"BOOL": false
Null	NULL	A null value. Useful for missing values.	"NULL": true

DynamoDB data types [2]

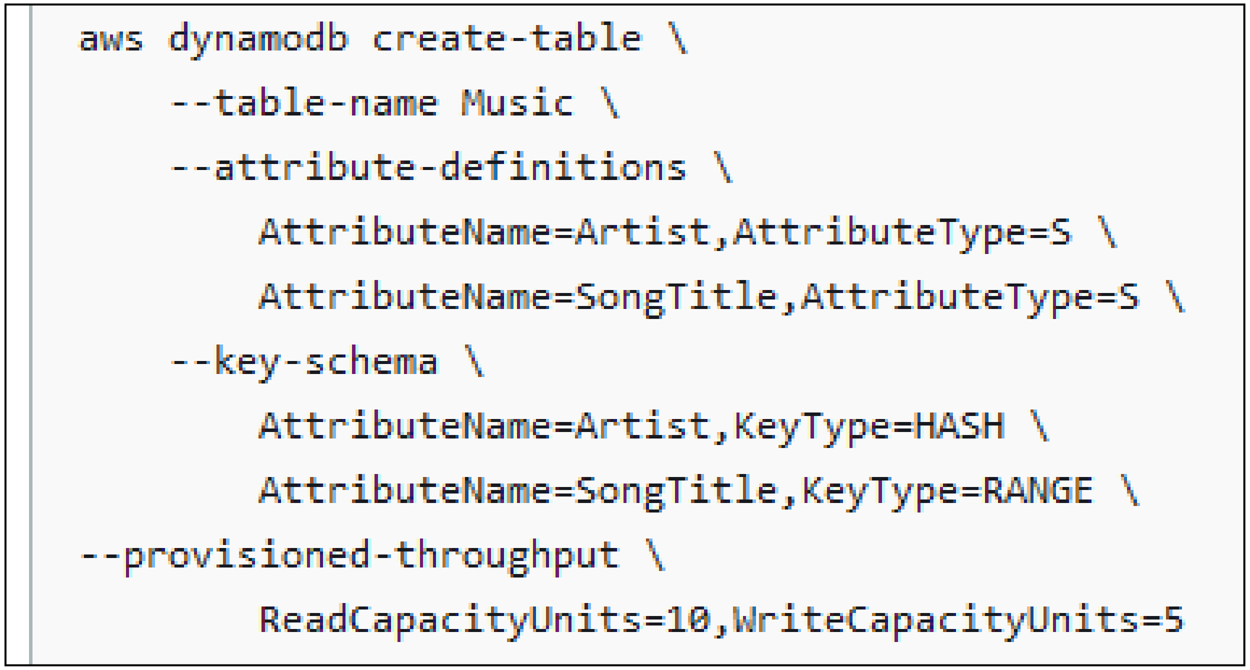
Set Types and Document Types (List and Map):



Type	Symbol	Description	JSON Example
String set	SS	A set of strings.	"SS": ["Larry", "Moe"]
Number set	NS	A set of numbers.	"NS": ["42", "137"]
Binary set	BS	A set of binary strings.	"BS": ["TGFycnkK", "TW9lCg=="]
List	L	A list that can consist of data of any scalar type, like a JSON array. You can mix scalar types as well.	"L": [{"S": "totchos"}, {"N": "741"}]
Map	M	A key-value structure with strings as keys and values of any type, including sets, lists, and maps.	"M": {"FavoriteBook": {"S": "Old Yeller"}}

Dynamo DB API – CREATE TABLE

- Here is an example: creating “**Music**” table
- Attributes: Artist (String), SongTitle (String)
- Partition Key: Artist (HASH implies “Partition”), Sort Key: SongTitle (RANGE implies Sort)



```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
--provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5
```

Table Storage Provisioning

- When we create a table, we require specifying "**provisioned throughput capacity**". Which is the amount of read and write activity that the table can support.
- DynamoDB uses this information to reserve sufficient system resources to meet user's throughput requirements
- We require specifying Read Capacity Units and Write Capacity Units. These numbers determine what read write we can perform per second in terms of numbers and in terms of record (item) size.
- Let us keep specify 10 and 5 units for these respectively, till we learn more about it.

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

Describe Table

```
aws dynamodb describe-table
    --table-name Books
```

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "isbn",
                "AttributeType": "S"
            }
        ],
        "TableName": "Books",
        "KeySchema": [
            {
                "AttributeName": "isbn",
                "KeyType": "HASH"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": "2021-02-11T15:31:12.906000+05:30",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "1970-01-01T05:30:00+05:30",
            "LastDecreaseDateTime": "1970-01-01T05:30:00+05:30",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:ddblocal:000000000000:table/Books"
    }
}
```

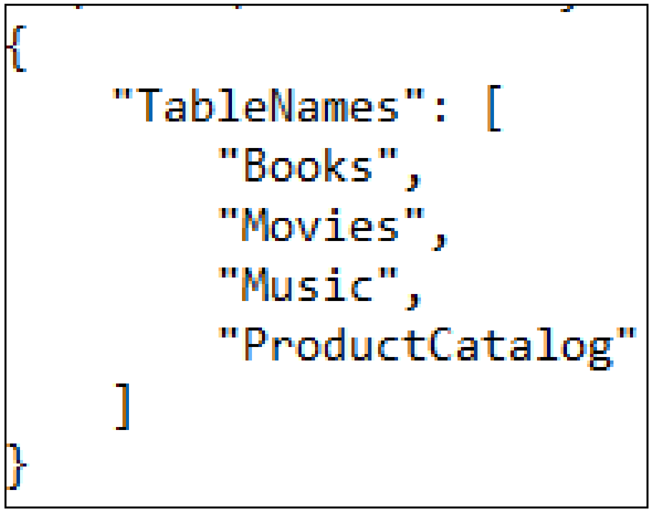
List Tables

- List all tables in “your database”

  `aws dynamodb list-tables`

  Sample output snapshot is shown below.

- Note there is no concept of “database” in dynamoDB. Amazon AWS probably shows all tables in the user’s account.



```
{
    "TableNames": [
        "Books",
        "Movies",
        "Music",
        "ProductCatalog"
    ]
}
```

Update Table and Drop Table

Update Table

- Allows making modifications in Table schema
- Mostly you provide same inputs
  - Attribute List
  - Index Definitions
  - Provisioning information
- <u>We can not Change Keys</u>

Delete Table

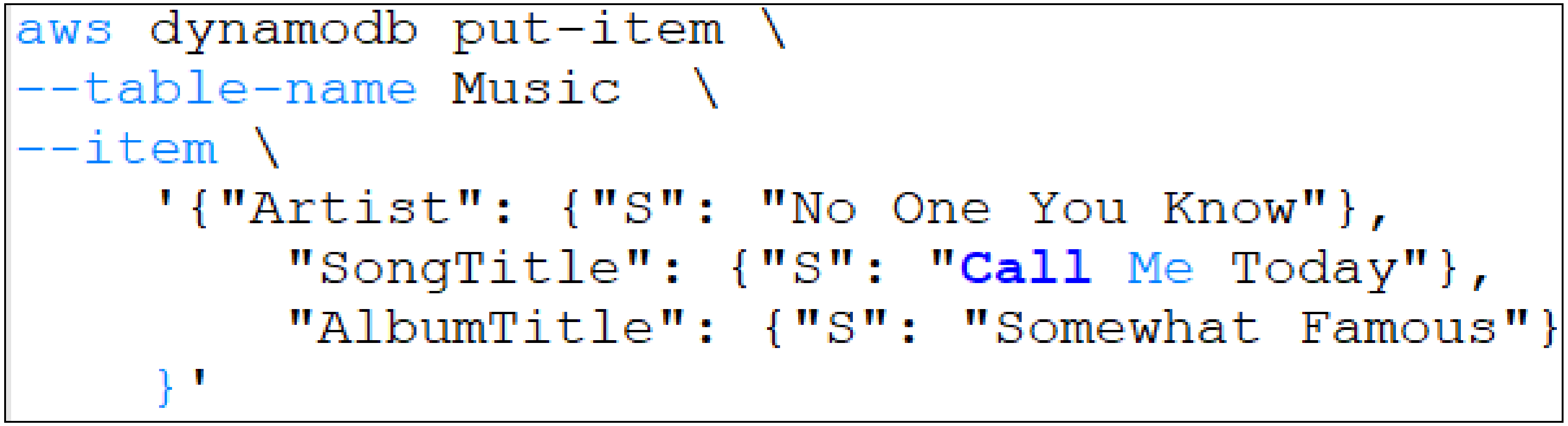
- Drops a table

DynamoDB API – WRITE

- `put-item` – Writes a single item to a table. You must specify the primary key attributes, but you don't have to specify other attributes.
- `batch-write-item` – Writes up to 25 items to a table.
- Note: Read/write operations have batch variations. The batch variations are more efficient than calling single item operation multiple times because, the application only needs a single network round trip to read/write the items.

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

“put-item” example



```
aws dynamodb put-item \
--table-name Music  \
--item \
    '{"Artist": {"S": "No One You Know"},
        "SongTitle": {"S": "Call Me Today"},
        "AlbumTitle": {"S": "Somewhat Famous"}
    }'
```

- Parameters:
  - Table name
  - Item to be added in JSON format

“put-item” example

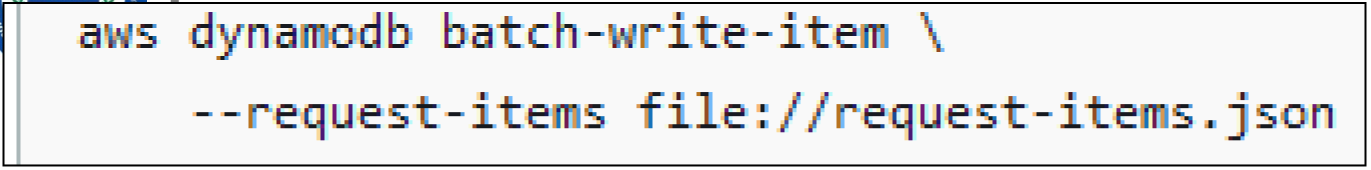
```
aws dynamodb put-item \
    --table-name Thread \
    --item file://item.json
```

item.json

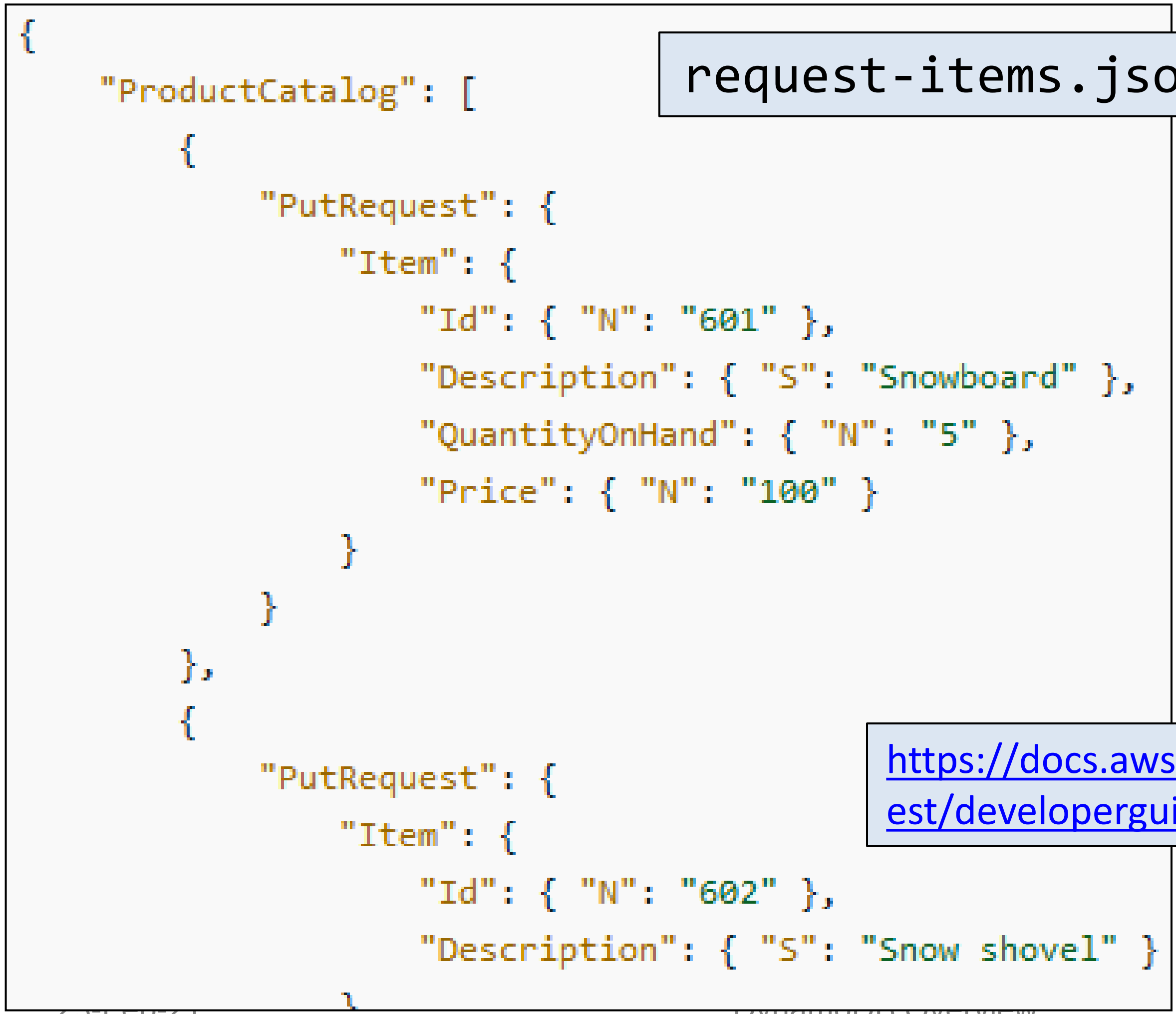
```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"},
    "Message": {"S": "First post in this thread"},
    "LastPostedBy": {"S": "fred@example.com"},
    "LastPostDateTime": {"S": "201603190422"}
}
```

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

“batch-write-item” example



```
aws dynamodb batch-write-item \
    --request-items file://request-items.json
```



request-items.json

```
{
    "ProductCatalog": [
        {
            "PutRequest": {
                "Item": {
                    "Id": { "N": "601" },
                    "Description": { "S": "Snowboard" },
                    "QuantityOnHand": { "N": "5" },
                    "Price": { "N": "100" }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": { "N": "602" },
                    "Description": { "S": "Snow shovel" }
                }
```

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

DynamoDB API – Update

- `Update-Item` – Modifies one or more attributes in an item.
- We must specify the primary key for the item that we want to modify.
- We can add new attributes and modify or remove existing attributes.
- We can also perform conditional updates, so that the update is only successful when a user-defined condition is met.

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

“update-item” example

- Parameters: (1) Key, (2) Update expressions, (3) new values

```
aws dynamodb update-item \
    --table-name Music \
    --key '{ "Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"}}' \
    --update-expression "SET AlbumTitle = :newval" \
    --expression-attribute-values '{":newval":{"S":"Updated Album Title"}}' \
    --return-values ALL_NEW
```

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id": { "N": "601" }}' \
    --update-expression "SET Price = Price + :incr" \
    --expression-attribute-values '{":incr":{"N":"5"}}' \
    --return-values UPDATED_NEW
```

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

“update-item” example

```
aws dynamodb update-item \
    --table-name Thread \
    --key file://key.json \
    --update-expression "SET Answered = :zero, Replies = :zero, LastPostedBy = :lastpostedby" \
    --expression-attribute-values file://expression-attribute-values.json \
    --return-values ALL_NEW
```

key.json

```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"}
}
```

expression-attribute-values.json

```
{
    ":zero": {"N":"0"},
    ":lastpostedby": {"S":"barney@example.com"}
}
```

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

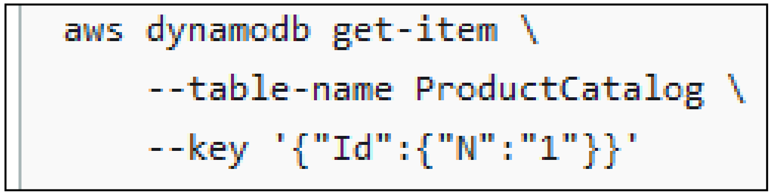
DynamoDB API - Read

- `get-item` – Retrieves a single item from a table. Require specifying the primary key for the item that is to be read.
- We can retrieve the entire item, or a subset of its attributes.
- `batch-get-item` – Retrieves up to 100 items from one or more tables.
- `query` – querying based on Partition Key (and optionally on sort key, if applicable)
    - Query can also be based on of secondary index: its partition key (and optionally sort key).
- `scan` – Retrieves all items in the specified table or index.
    - Optionally, we can apply a filtering condition to return items that meet specified criteria.

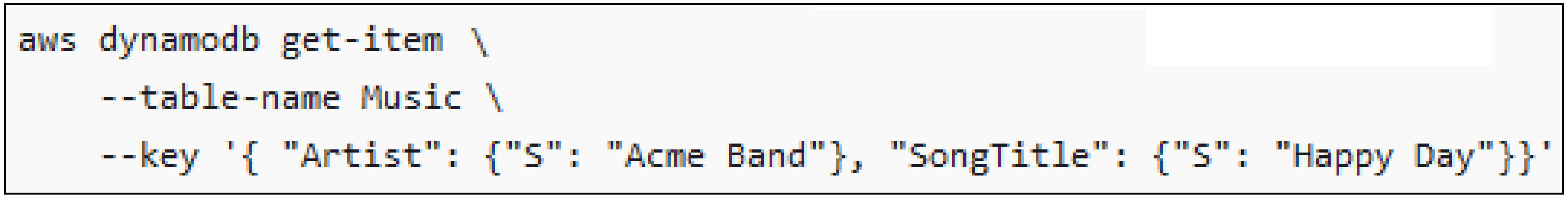
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.API.html

"get-item" example

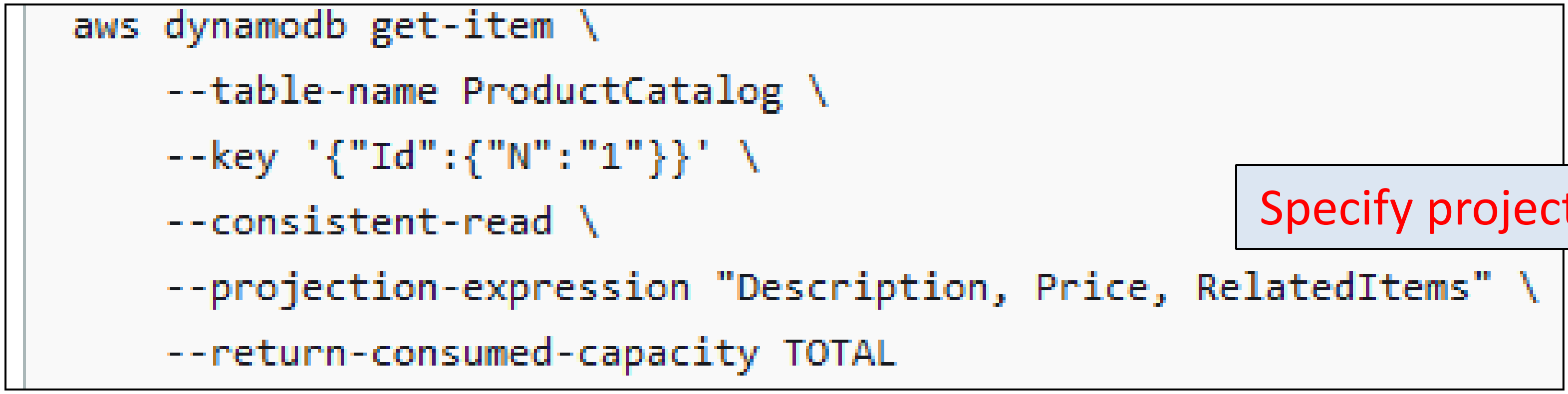
- Input: "Key"



```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}'
```



```
aws dynamodb get-item \
    --table-name Music \
    --key '{ "Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"}}'
```



```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --consistent-read \
    --projection-expression "Description, Price, RelatedItems" \
    --return-consumed-capacity TOTAL
```

Specify projection

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

“batch-get-item” example

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json
```

request-items.json

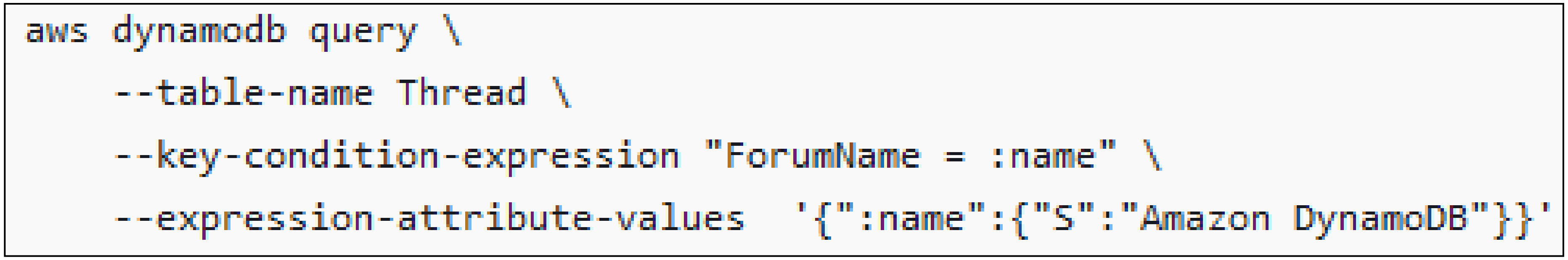
```
{
    "Thread": { //table-name
        "Keys": [
            {     // key values
                "ForumName":{"S": "Amazon DynamoDB"},
                "Subject":{"S": "DynamoDB Thread 1"}
            },
            {
                "ForumName":{"S": "Amazon S3"},
                "Subject":{"S": "S3 Thread 1"}
            }
        ],
        "ProjectionExpression":"ForumName, Subject, LastPostedDateTime, Replies"
    }
}
```

Two key values have been specified.

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

“query” example

- Query (also) allows querying based on Key only
- We specify “Key-Expression” “Key Value Pairs”
- In the example below:
  - Table: `Thread`. Partition Key: `ForumName`
  - Returns all of the items with given value for given partition key.



```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :name" \
    --expression-attribute-values  '{":name":{"S":"Amazon DynamoDB"}}'
```

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

“query” example

- Table: Thread. Partition Key: Forum Name
- Returns all of the items with given value for given partition key value

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :name and Subject = :sub" \
    --expression-attribute-values  file://values.json
```

values.json

```
{
    ":name":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"}
}
```

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

Process “query” Results

- After query has been retrieved based on Partition Key (and Sort Key), results of that query can be further processed
  - Filtering
  - Counting
  - Limiting the items in results set (LIMIT parameter)
  - Paginating the results
- Note that a query operation can retrieve a maximum of 1 MB of data (before the filter)

Filter Expressions for Query

- Filter can be on any attribute; here is a query on “ForumName” partition key, and “Subject” sort key.

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :fn and Subject = :sub" \
    --filter-expression "#v >= :num" \
    --expression-attribute-names '{"#v": "Views"}' \
    --expression-attribute-values file://values.json
```

values.json

```
{
    ":fn":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"},
    ":num":{"N":"3"}
}
```

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

Limiting the Number of Items in the Result Set

- The Query operation allows us limit the number of items that it reads.
- To do this, set the Limit parameter to the maximum number of items that you want.
- Note that the filter is applied after limit.

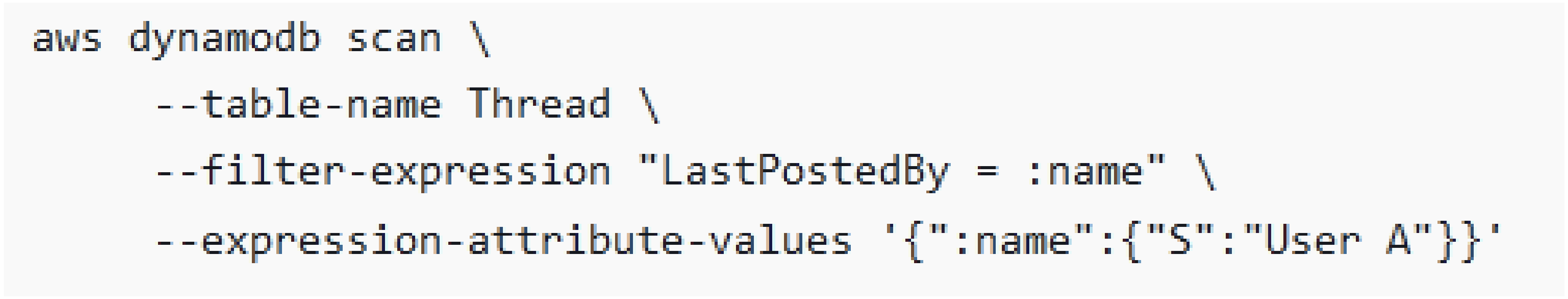
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Query.html

Scanning a Table

- Scan retrieves all items in the specified table or index.
- Scans whole table and allow doing
  - Filtering
  - Counting
  - Limiting the items in results set (LIMIT parameter)
  - Paginating the results
- Note that a scan can also retrieve a maximum of 1 MB of data
- We can also specify projecting attributes

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

Scan Examples



```
aws dynamodb scan \
     --table-name Thread \
     --filter-expression "LastPostedBy = :name" \
     --expression-attribute-values '{":name":{"S":"User A"}}'
```

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

Scan Examples

```
aws dynamodb scan \
    --table-name ProductCatalog \
    --filter-expression "contains(Color, :c) and Price <= :p" \
    --expression-attribute-values file://values.json
```

The arguments for `--expression-attribute-values` are stored in the `values.json` file.

```
{
    ":c": { "S": "Black" },
    ":p": { "N": "500" }
}
```

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

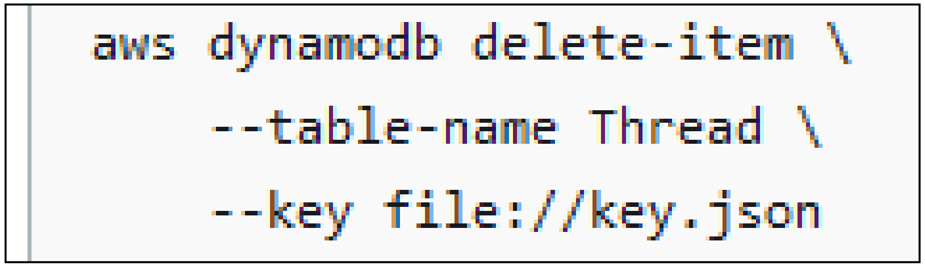
Parallel Scan

- By Default Amazon Scan is sequential, even if data are distributed across nodes
- We can make it parallel by specifying total number of segments.

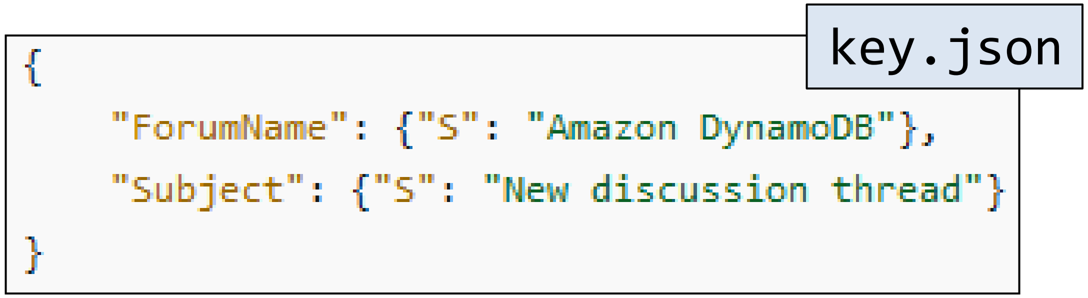
Query and Scan operations

- Query:
  - Can be only on Key
  - However, we may only specify partition key, sort-key is optional
  - Filter can be applied on any attribute
- Scan is complete scan of a table
  - Filter can be applied on any attribute

“delete-item” example



```
aws dynamodb delete-item \
    --table-name Thread \
    --key file://key.json
```

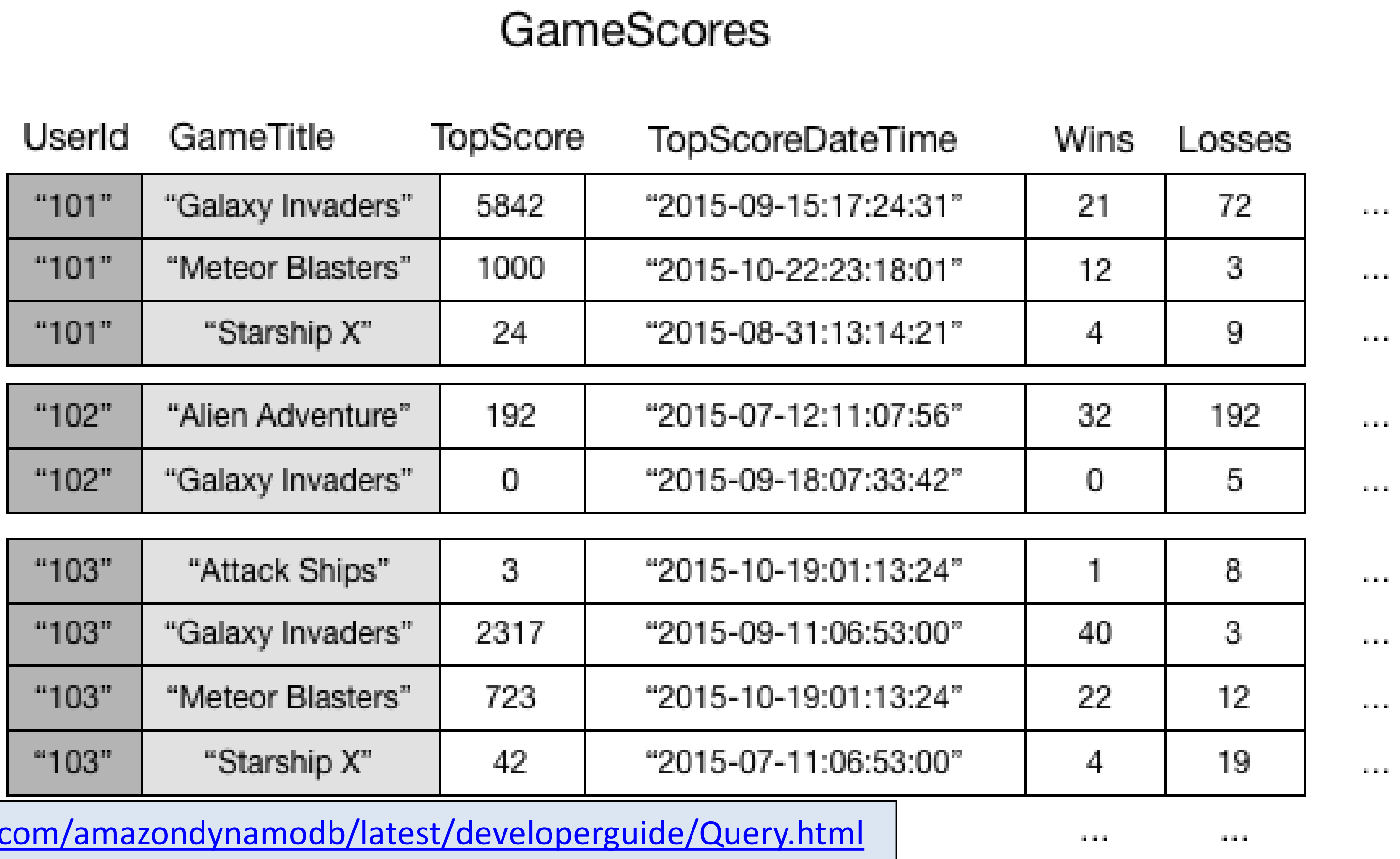


key.json

```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"}
}
```

Create a Global Secondary Index

- Let us understand Global Secondary Index (GSI) further.
- Here is a table “GameScore” with “UserID” as Partition Key and “Game Title” as Sort Key



GameScores

UserId	GameTitle	TopScore	TopScoreDateTime	Wins	Losses	
“101”	“Galaxy Invaders”	5842	“2015-09-15:17:24:31”	21	72	...
“101”	“Meteor Blasters”	1000	“2015-10-22:23:18:01”	12	3	...
“101”	“Starship X”	24	“2015-08-31:13:14:21”	4	9	...
“102”	“Alien Adventure”	192	“2015-07-12:11:07:56”	32	192	...
“102”	“Galaxy Invaders”	0	“2015-09-18:07:33:42”	0	5	...
“103”	“Attack Ships”	3	“2015-10-19:01:13:24”	1	8	...
“103”	“Galaxy Invaders”	2317	“2015-09-11:06:53:00”	40	3	...
“103”	“Meteor Blasters”	723	“2015-10-19:01:13:24”	22	12	...
“103”	“Starship X”	42	“2015-07-11:06:53:00”	4	19	...
				...	...	

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

Create a Global Secondary Index

- Now such a design answering of queries based on UserId and Game Title would be very efficient.
- However, if the application needed to query as following:

  Q#1: What is the top score ever recorded for the game “Meteor Blasters”?

  Q#2: Which user had the highest score for Galaxy Invaders?

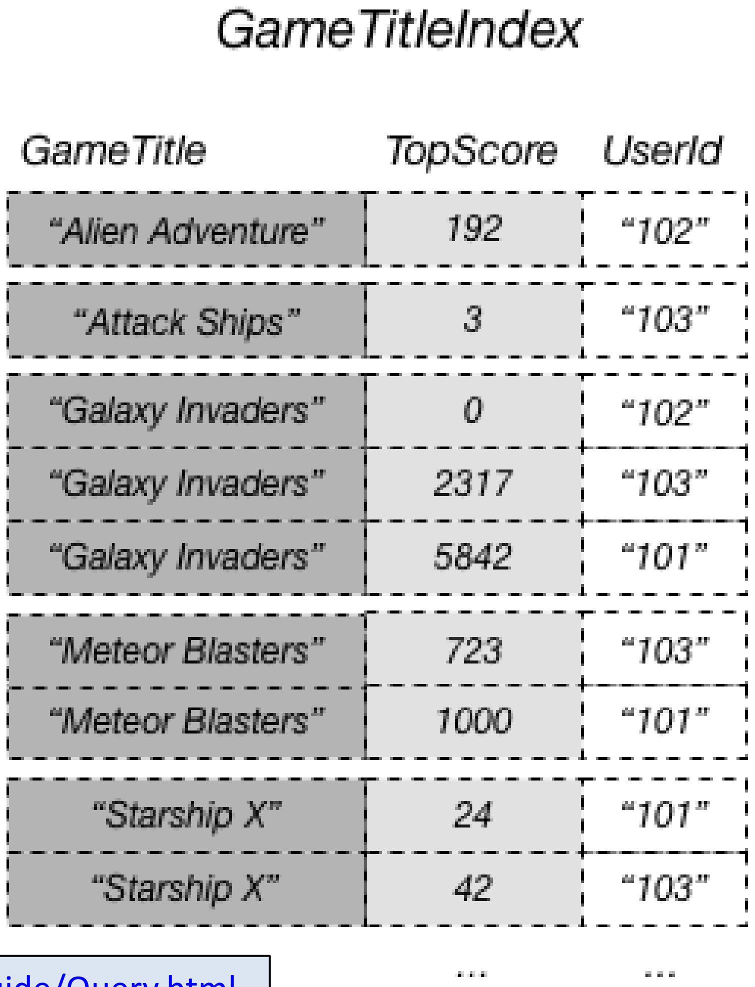
  Q#3: What was the highest ratio of wins vs. losses?
- Q#1 require looking for a data item on attribute “GameTitle”
- Q#2 require looking for a data item on attribute “GameTitle” and descending sort on Score
- Q#3 require sort data items on ratio of “Wins/Losses”
- In absence of Indexes on said attributes, querying shall require “SCAN” which may become unacceptable on huge files.

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

Create a Global Secondary Index

- A secondary index on GameTitle, and TopScore shall make Q#1, and Q#2 efficient.
- Q#3, may either require different index or may accept SCAN.
- Here is an example GSI for table GameScore on attributes
  - GameTitle (Partition Key), and
  - TopScore (Sort Key)

  let the index be named as "GameTitleIndex"!



GameTitleIndex

GameTitle	TopScore	UserId
"Alien Adventure"	192	"102"
"Attack Ships"	3	"103"
"Galaxy Invaders"	0	"102"
"Galaxy Invaders"	2317	"103"
"Galaxy Invaders"	5842	"101"
"Meteor Blasters"	723	"103"
"Meteor Blasters"	1000	"101"
"Starship X"	24	"101"
"Starship X"	42	"103"
	...	...

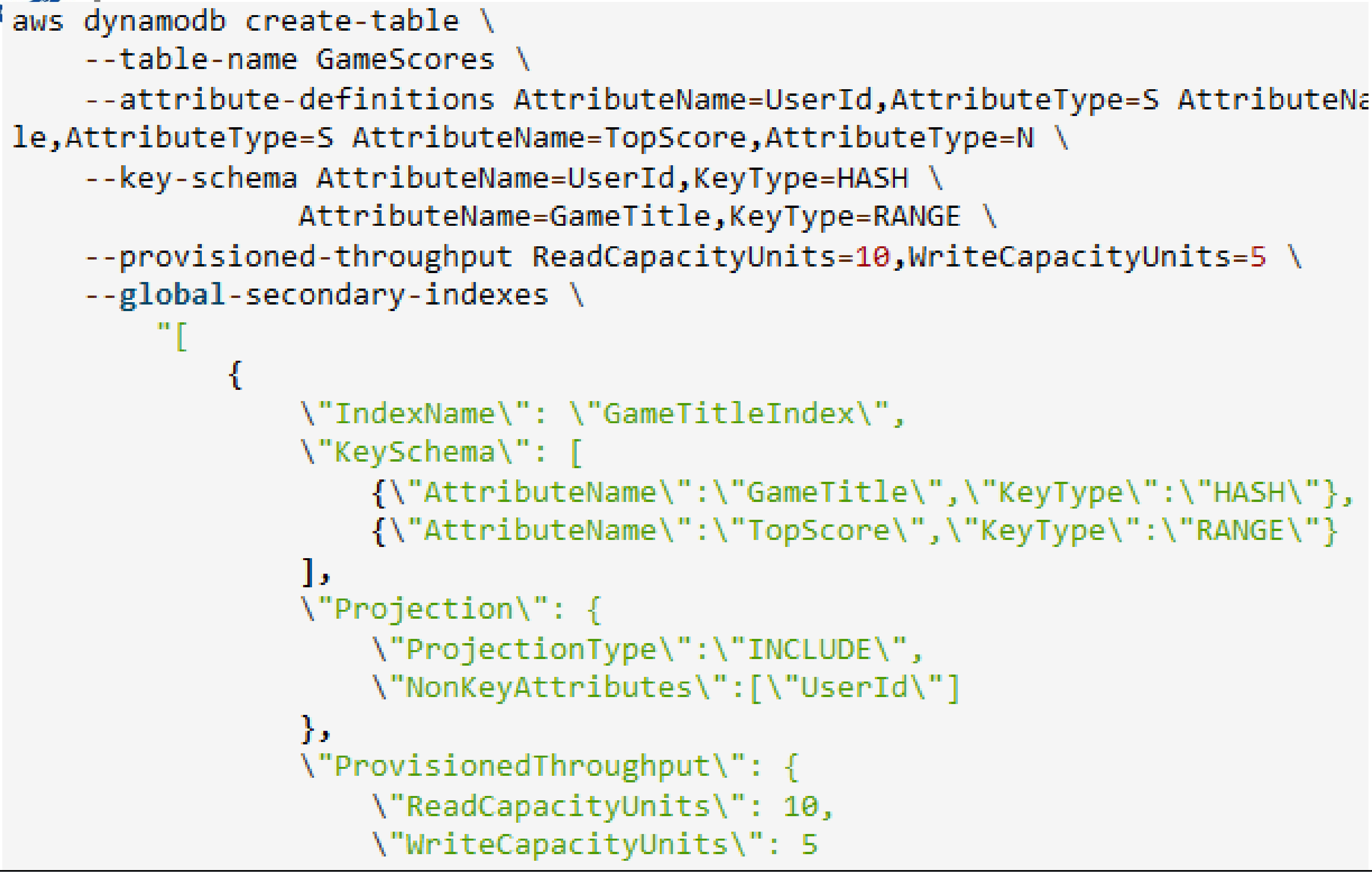
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Query.html

Create a Global Secondary Index

- Secondary index is aimed to store "references" to base table items, therefore, index normally store "Keys" of data records.
- In order to perform search for more attributes, we require getting relevant records through retrieved keys from the index.
- We can however project more attributes in the index, just to avoid two level look-up!
- We have following attribute projection options in GSI:
  - KEYS_ONLY
  - INCLUDE – we specify attributes that are to included
  - ALL– all attributes are copied in the index too.
- Decision of projecting non-key attributes in index may add to storage, and a tradeoff should be used between storage cost and efficiency benefit.

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

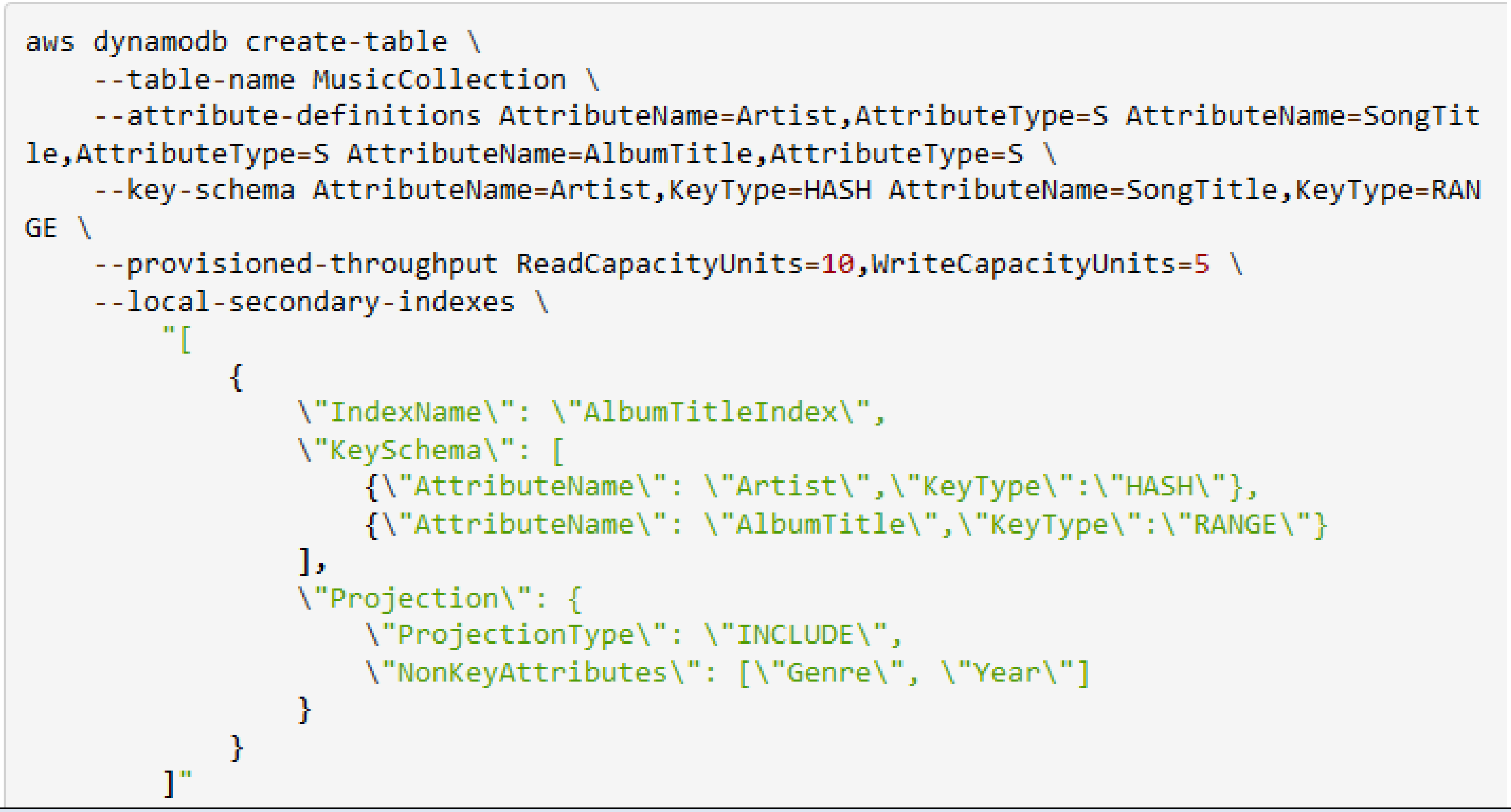
Example: create-table with GSI



```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S AttributeNa
le,AttributeType=S AttributeName=TopScore,AttributeType=N \
    --key-schema AttributeName=UserId,KeyType=HASH \
                AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
```

https://awscli.amazonaws.com/v2/documentation/api/latest/reference/dynamodb/create-table.html#examples

Example: create-table with LSI



```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=Artist,AttributeType=S AttributeName=SongTit
le,AttributeType=S AttributeName=AlbumTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RAN
GE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\": \"Artist\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\": \"AlbumTitle\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\": \"INCLUDE\",
                    \"NonKeyAttributes\": [\"Genre\", \"Year\"]
                }
            }
        ]"
```

https://awscli.amazonaws.com/v2/documentation/api/latest/reference/dynamodb/create-table.html#examples

Example: update-table with GSI

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=AlbumTitle,AttributeType=S \
    --global-secondary-index-updates file://gsi-updates.json
```

Contents of `gsi-updates.json`:

```
[
    {
        "Create": {
            "IndexName": "AlbumTitle-index",
            "KeySchema": [
                {
                    "AttributeName": "AlbumTitle",
                    "KeyType": "HASH"
                }
            ],
            "ProvisionedThroughput": {
                "ReadCapacityUnits": 10,
                "WriteCapacityUnits": 10
            },
            "Projection": {
                "ProjectionType": "ALL"
```

https://awscli.amazonaws.com/v2/documentation/api/latest/reference/dynamodb/update-table.html

Querying through GSI

- We can retrieve items from table through GSI using the Query and Scan operations only.
  - GetItem operations can't be used on GSI
- Parameters to “Query” through GSI are
  - Name of the index
  - Name of the base table
  - The attributes to be returned in the query results, and any query conditions that you want to apply
  - DynamoDB can return the results in ascending or descending order.

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

Example: Query through GSI

- Returns User, TopScore for game “Meteor Blasters”. Searching is done using GSI “GameTitleInex”!

```
aws dynamodb query \
  --table-name GameScores \
  --index-name GameTitleIndex \
  --key-condition-expression "GameTitle=:v_title" \
  --expression-attribute-values '{
      ":v_title": {"S": "Meteor Blasters"}
  }' \
  --projection-expression "UserId, TopScore"
```

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