Amazon DynamoDB Cheat Sheet

Console Commands

BlowStack

Table Management

Partition Key (HASH) | Sort Key (RANGE)

unique for each item | range-based queries

Create Table

```
$ aws dynamodb create-table \
  --table-name TableName \
  --attribute-definitions \
      AttributeName=PartitionKey,AttributeType=S \
      AttributeName=SortKey,AttributeType=N \
  --key-schema \
      AttributeName=PartitionKey,KeyType=HASH \
      AttributeName=SortKey,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=5,WriteCapacityUnits=5
```

Update Table

```
$ aws dynamodb update-table \
  --table-name TableName \
  --provisioned-throughput
ReadCapacityUnits=10,WriteCapacityUnits=10
```

Delete Table

```
$ aws dynamodb delete-table --table-name
TableName
```

List Tables with a Specific Prefix

```
$ aws dynamodb list-tables --exclusive-start-table-
name TableNamePrefix
```

Describe Table (Get Info)

```
$ aws dynamodb describe-table --table-name Name
```

Data Types

String (S)

```
{ "S": "Example string" }
```

String Set (SS)

```
{ "SS": ["a", "b"] }
```

Number (N)

```
{ "N": "123.45" }
```

Number Set (NS)

```
{ "NS": ["1", "2"] }
```

Boolean (BOOL)

```
{ "BOOL": true }
```

List (L)

```
{ "L": [{"S": "a"}, {"N": "1"}] }
```

Map (M)

```
{
  "M": {
    "key1": { "S": "value1" }
  }
}
```

Binary (B)

```
{ "B": "base64EncodedString" }
```

Binary Set (BS)

```
{ "BS": ["base64S1", "base64S2"] }
```

Null (NULL)

```
{ "NULL": true }
```

Index Management - GSI

Can be added to a table at any time

Can have different partition keys from the base table, providing more flexibility in query patterns

Write throughputis separate from the table

Only support eventual consistency

Creating a Global Secondary Index (GSI)

```
$ aws dynamodb update-table \
    --table-name TableName \
    --attribute-definitions
AttributeName=AttributeType,AttributeNa
me=AttributeType \
    --global-secondary-index-updates "
[{\"Create\":{\"IndexName\":
\"NewIndexName\",\"KeySchema\":
[{\"AttributeName\":\"PartitionKeyName\",
\"KeyType\":\"HASH\"},
{\"AttributeName\":\"SortKeyName\",\"Key
Type\":\"RANGE\"}],\"Projection\":
{\"ProjectionType\":\"ALL\"},\"ProvisionedT
hroughput\":{\"ReadCapacityUnits\":
10,\"WriteCapacityUnits\": 10}}}]"
```

Example - Creating GSI

```
$ aws dynamodb update-table \
    --table-name Users \
    --attribute-definitions
AttributeName=Email,AttributeType=S \
    --global-secondary-index-updates "
[{\"Create\":{\"IndexName\":
\"UserEmailIndex\",\"KeySchema\":
[{\"AttributeName\":\"Email\",\"KeyType\":\
"HASH\"}],\"Projection\":
{\"ProjectionType\":\"ALL\"},\"ProvisionedT
hroughput\":{\"ReadCapacityUnits\":
5,\"WriteCapacityUnits\": 5}}}]"
```

Deleting a Global Secondary Index

```
$ aws dynamodb update-table \
    --table-name TableName \
    --global-secondary-index-updates "
[{\"Delete\":{\"IndexName\":
\"IndexName\"}}]"
```

Example - Deleting GSI

```
$ aws dynamodb update-table \
    --table-name Users \
    --global-secondary-index-updates "
[{\"Delete\":{\"IndexName\":
\"UserEmailIndex\"}}]"
```

Modifying GSI Provisioned Throughput

```
$ aws dynamodb update-table \
    --table-name TableName \
    --global-secondary-index-updates "
[{\"Update\":{\"IndexName\":
\"IndexName\",\"ProvisionedThroughput\":
{\"ReadCapacityUnits\": NewReadCapacity,
\"WriteCapacityUnits\":
NewWriteCapacity}}}]"
```

Example - Modifying GSI Provisioned Throughput

```
$ aws dynamodb update-table \
    --table-name TableName \
    --global-secondary-index-updates "
[{\"Update\":{\"IndexName\":
\"IndexName\",\"ProvisionedThroughput\":
{\"ReadCapacityUnits\": NewReadCapacity,
\"WriteCapacityUnits\":
NewWriteCapacity}}}]"
```

Index Management - LSI

Share the same partition key as the table
Allow additional querying capabilities on the same partition key but different sort keys

Defined at the time of table creation
Allow for both eventual consistency and strongly consistent reads

Define the Table and Local Secondary Index

```
$ aws dynamodb create-table \
    --table-name Employees \
    --attribute-definitions \
        AttributeName=EmployeeID,AttributeType=S \
        AttributeName=LastName,AttributeType=S \
        AttributeName=HireDate,AttributeType=S \
    --key-schema \
        AttributeName=EmployeeID,KeyType=HASH \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        IndexName=LastNameIndex,KeySchema=
[{AttributeName=EmployeeID,KeyType=HASH},
{AttributeName=LastName,KeyType=RANGE}],Projection=
{ProjectionType=INCLUDE,NonKeyAttributes=["FirstName","Department"]}
```

Parameters

--attribute-definitions
Attributes used in the table's primary key and the index's key schema

--key-schema
Defines the primary key for the table

--provisioned-throughput
Read and write throughput the table can handle before throttling

--local-secondary-indexes
Defines the LSI. The LastNameIndex uses the same partition key (EmployeeID) and a different sort key (LastName). INCLUDE projects only certain attributes

TIPS

Projection Types
KEYS_ONLY - only the key attributes, INCLUDE -specify additional attributes, ALL - includes every attribute

Cost Considerations
Share throughput capacity with the main table - do not incur separate costs for provisioned throughput. Storage costs and data retrieval times can be affected

Use Cases
Efficient access patterns based on alternative sort keys without the need to Scan.

Item Management

Get Item

```
$ aws dynamodb get-item \
    --table-name TableName \
    --key '{"PrimaryKey": {"S": "KeyValue"}}'
```

Put Item

```
$ aws dynamodb get-item \
    --table-name TableName \
    --key '{"PrimaryKey": {"S": "KeyValue"}}'
```

Conditional Put (Insert) Item

```
$ aws dynamodb put-item \
    --table-name YourTableName \
    --item '{"PrimaryKey": {"S": "KeyValue"},
"Attribute": {"S": "Value"}}' \
    --condition-expression
"attribute_not_exists(PrimaryKey)"
```

Delete Item

```
$ aws dynamodb delete-item \
    --table-name TableName \
    --key '{"PrimaryKey": {"S": "KeyValue"}}'
```

Conditional Delete Item

```
$ aws dynamodb delete-item \
    --table-name YourTableName \
    --key '{"PrimaryKey": {"S": "KeyValue"}}' \
    --condition-expression
"attribute_exists(PrimaryKey) AND #attr =
:val" \
    --expression-attribute-names
'{"#attr":"AttributeName"}' \
    --expression-attribute-values '{":val":
{"S":"ExpectedValue"}}'
```

Update Item

```
$ aws dynamodb update-item \
    --table-name TableName \
    --key '{"PrimaryKey": {"S": "KeyValue"}}' \
    --update-expression "SET AttributeName
= :value" \
    --expression-attribute-values '{":value":
{"S": "NewAttributeValue"}}'
```

Conditional Update Item

Updates the attribute "AttributeName" to "NewValue" only if its current value is "OldValue

```
$ aws dynamodb update-item \
    --table-name YourTableName \
    --key '{"PrimaryKey": {"S": "KeyValue"}}' \
    --update-expression "SET #attr = :val" \
    --condition-expression "#attr = :oldval" \
    --expression-attribute-names
'{"#attr":"AttributeName"}' \
    --expression-attribute-values '{":val":
{"S":"NewValue"}, ":oldval":
{"S":"OldValue"}}' \
    --return-values UPDATED_NEW
```

Parameters

--table-name
Specifies the DynamoDB table

--item or --key
Specifies the item or key to manipulate

--condition-expression
Describes the condition execution

-expression-attribute-names and --expression-attribute-values
Mappings for attribute names and values

Batch Items Management

Batch Get Items

```
$ aws dynamodb batch-get-item --request-items '{
   "TableName": {
      "Keys": [
         {"PrimaryKey": {"S": "KeyValue1"}},
         {"PrimaryKey": {"S": "KeyValue2"}}
      ]
   }
}'
```

Batch Get Items (from file)

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

```
{
   "RequestItems": {
      "YourTableName": {
         "Keys": [
            {"PrimaryKey": {"S": "Key1"}},
            {"PrimaryKey": {"S": "Key2"}}
         ],
         "AttributesToGet": [
            "Attribute1", "Attribute2"
         ],
         "ConsistentRead": true
      }
   }
}
```

JSON <=> DynamoDB

DynamoDB two way converter tool

- converts JSON into DynamoDB
- converts DynamoDB into JSON

blowstack.com/tools/dynamo-dbconverter

Batch Write Items

```
$ aws dynamodb batch-write-item --request-items '{
   "TableName": [
      {
        "PutRequest": {
          "Item": {
            "PrimaryKey": {"S": "KeyVal1"},
             "AttributeName": {"S": "AttVal1"}
          }
        }
      },
      {
        "PutRequest": {
          "Item": {
            "PrimaryKey": {"S": "KeyVal2"},
            "AttributeName": {"S": "AttVal2"}
          }
        }
      }
   ]
}'
```

Batch Write Items (from file)

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

```
{  "RequestItems": {
      "YourTableName": [
        {   "PutRequest": {
              "Item": {
                 "PrimaryKey": {"S": "Key1"},
                 "Attribute1": {"S": "Val1"}
              }  }  },
        { "DeleteRequest": {
              "Key": {
                 "PrimaryKey": {"S": "Key2"}
              }  }  }  ]  } }
```

Query and Scan Items

Query

Find an item using a primary key or indexed attribute

```
$ aws dynamodb query \
    --table-name TableName \
    --key-condition-expression
"AttributeName = :value" \
    --expression-attribute-values  '{":value":
{"S":"ActualValue"}}'
```

Example

```
$ aws dynamodb query \
    --table-name Users \
    --key-condition-expression "UserId =
:userId" \
    --expression-attribute-values  '{":userId":
{"S":"12345"}}'
```

Query an index (LSI or GSI)

```
$ aws dynamodb query \
    --table-name TableName \
    --index-name IndexName \
    --key-condition-expression
"PartitionKeyName = :partitionkeyval" \
    --expression-attribute-values
'{":partitionkeyval":
{"S":"PartitionKeyValue"}}'
```

Query Pagination (can be Scan)

```
$ aws dynamodb query \
    --table-name TableName \
    --key-condition-expression
"PartitionKeyName = :partitionkeyval" \
    --expression-attribute-values
'{":partitionkeyval":
{"S":"PartitionKeyValue"}}' \
    --exclusive-start-key
'{"PartitionKeyName":
{"S":"LastEvaluatedKey"}}'
```

Scan

Reads and return every item in a table

```
$ aws dynamodb scan --table-name Name
```

Filtering a Scan (can be Query)

```
$ aws dynamodb scan \
  --table-name TableName \
  --filter-expression "AttributeName = :val"
\
  --expression-attribute-values '{":val":
{"S":"FilteredVal"}}'
```

Common Parameters

--table-name
Name of the table to query/scan

--filter-expression
Filter results of the query/scan

--expression-attribute-values
Placeholders used in the expressions

--exclusive-start-key
Starting point for the next set of results

Parameters for Query

--index-name
Name of the index to query/scan

--key-condition-expression
Conditions what to retrieve

Parameters for Scan

--projection-expression
Attributes in the scan result

--segment and --total-segments
Used for parallel scans