CARLSON SCHOOL OF MANAGEMENT
UNIVERSITY OF MINNESOTA

DynamoDB: Amazon's Highly Available Key-value Store

MSBA 6330 Prof Liu

Slides credits go to Rick Houlihan's 2016 AWS webinar "Introduction to DynamoDB" and Zuhair Khayyat's Presentation on Dynamo & Amazon DynamoDB

Learning Objectives

- Understand the characteristics and use cases of DynamoDB
- Understand DynamoDB's data model
- Understand the different ways to interact with DynamoDB
- Be familiar with the design patterns for DynamoDB
- Understand the ecosystem around DynamoDB

DynamoDB: Amazon's Highly Available Key-value Store

OVERVIEW OF DYNAMODB

What is Dynamo

PA

- Dynamo is an ***eventually-consistent*** key-value storage system to support scalable highly available data access.
- Its design requirements:
  - **Simple** reads and writes to binary objects **not larger than 1 MB** while no operation spans for multiple data.
  - Very fast data access, (<300) ms response time.
  - Work with heterogeneous commodity hardware infrastructure.
  - Highly available (always on); expect small frequent network and server failures.
- Optimized for scalability and availability (always-on experience)

Amazon DynamoDB

- DynamoDB is a fully managed key-value store NoSQL database service on AWS (based on Dynamo).
  - *Fast*: single digit millisecond latencies.
  - *Scalable*: Automatic scaling to any workload
  - *Easy administration*: Easy to create. Easy to adjust.
  - *Consistent*: eventually consistent (default) or strong consistency (but will limit availability).
  - *Durable*: Replication across data centres and availability zones.

DynamoDB Use Cases

Time Series Data

Messaging

Online Voting

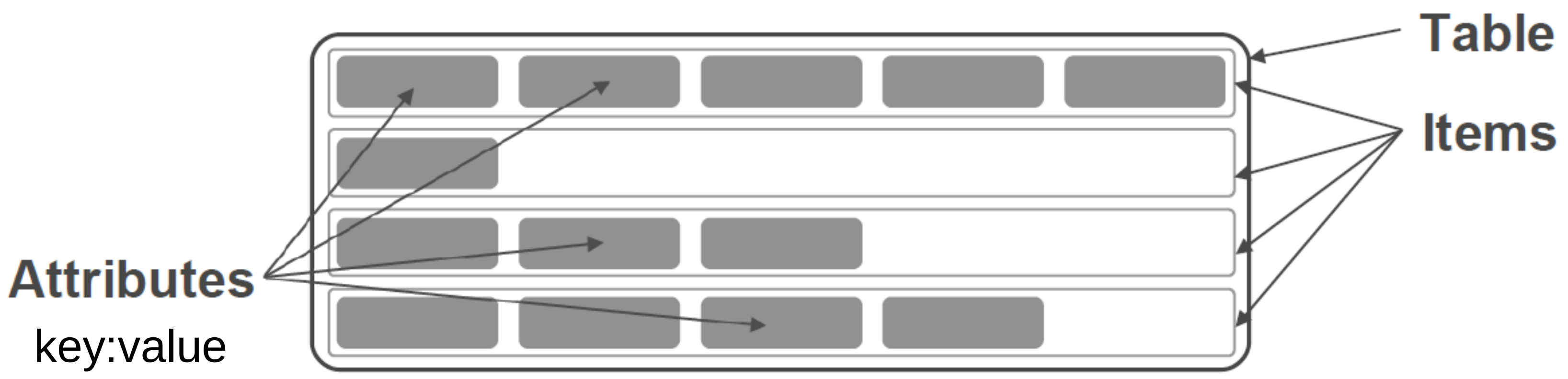
Product Catalog

Gaming

Data Model

- The **database** is a collection of tables.
- A **table** is a collection of items.
- An **item** is a collection of attributes (key-value pairs)
  - Items can vary in the number of attributes (*max size is 400 KB*)
  - **Primary key** is required to uniquely identify an item in a table.
- **Data types**:
  - Scalar data types: Number, String, and Binary (No nulls or empty strings)
  - Multi-valued types: String Set, Number Set, and Binary Set.

Large data stored in S3. Location stored in DynamoDB.



A Product Catalog Example

- Table name: ProductCatalog

```
{   Id = 101
    ProductName = "Book 101 Title"
    ISBN = "111-1111111111"
    Authors = [ "Author 1","Author 2" ]
    Price = -2
    Dimensions = "8.5 x 11.0 x 0.5"
    PageCount = 500
    InPublication = 1
    ProductCategory = "Book"
}
```

```
{   Id = 202
    ProductName = "21-Bicycle 202"
    Description = "202 description"
    BicycleType = "Road"
    Brand = "Brand-Company A"
    Price = 200
    Gender = "M"
    Color = [ "Green", "Black" ]
    ProductCategory = "Bike"
}
```

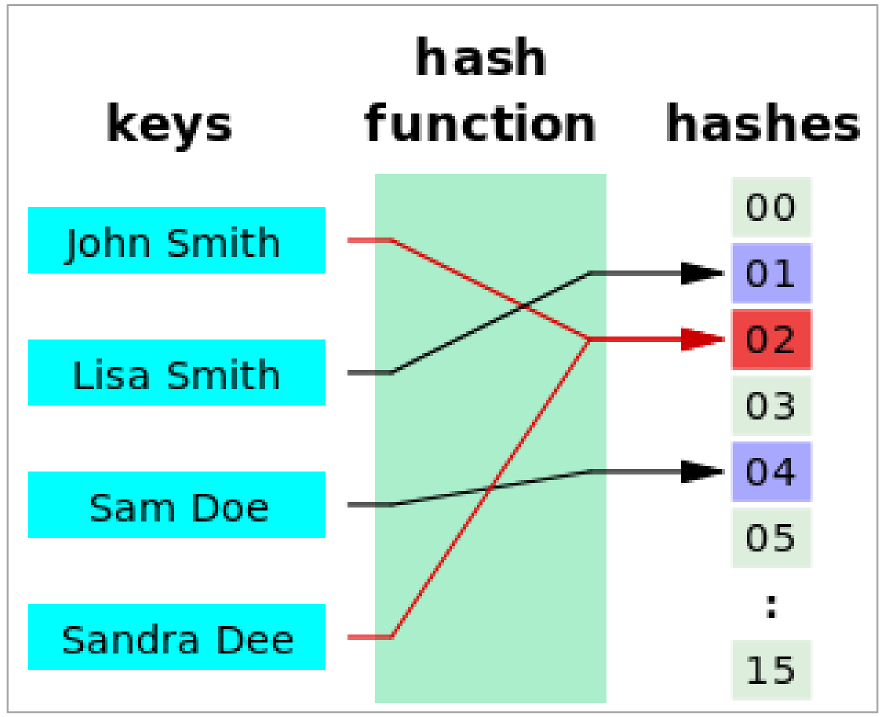
```
{   Id = 201
    ProductName = "18-Bicycle 201"
    Description = "201 description"
    BicycleType = "Road"
    Brand = "Brand-Company A"
    Price = 100
    Gender = "M"
    Color = [ "Red", "Black" ]
    ProductCategory = "Bike"
}
```

Primary key (partition key) is Id, but item attributes could vary depend on product type

Primary Keys

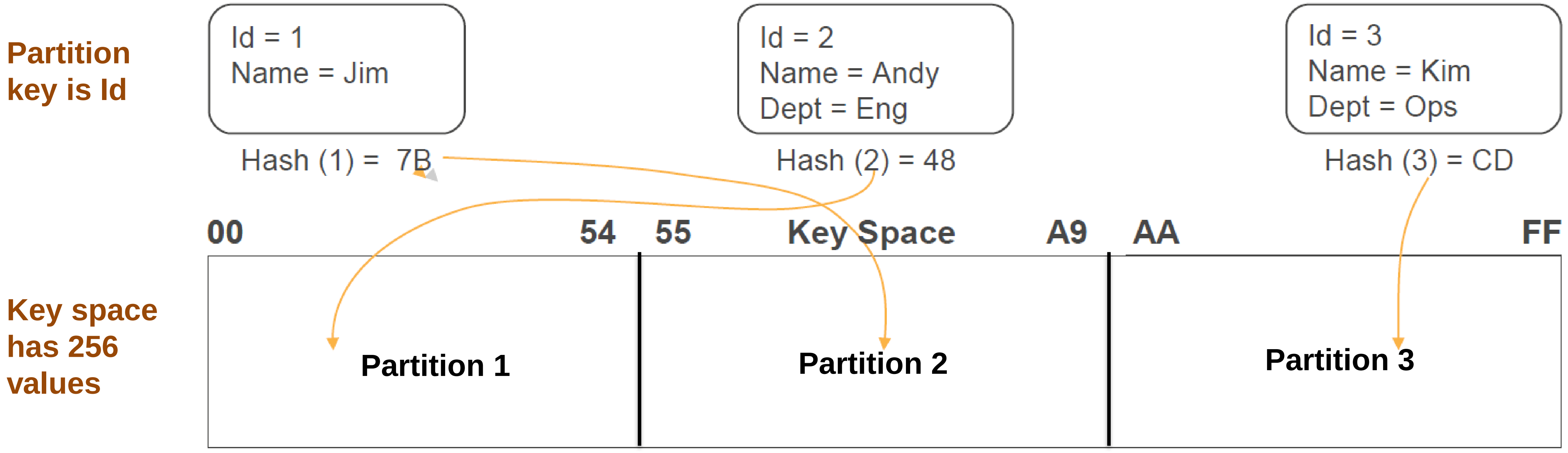
- DynamoDB supports two different kinds of primary keys.
  - **Partition key only:** consists of a single attribute
    - DynamoDB uses a hash of this value to determine the partition where the item will be stored
    - Enable a key-value access pattern
  - **Partition and sort key**:
    - Hashes the partition key to determine the partition where the item will be stored.
    - All items with the same partition are stored together, sorted by the sort key value.
    - The combination of partition and sort key must be unique
- The partition/sort key must be scalar type (cannot be set)

What is a hash?



Partition Key Only

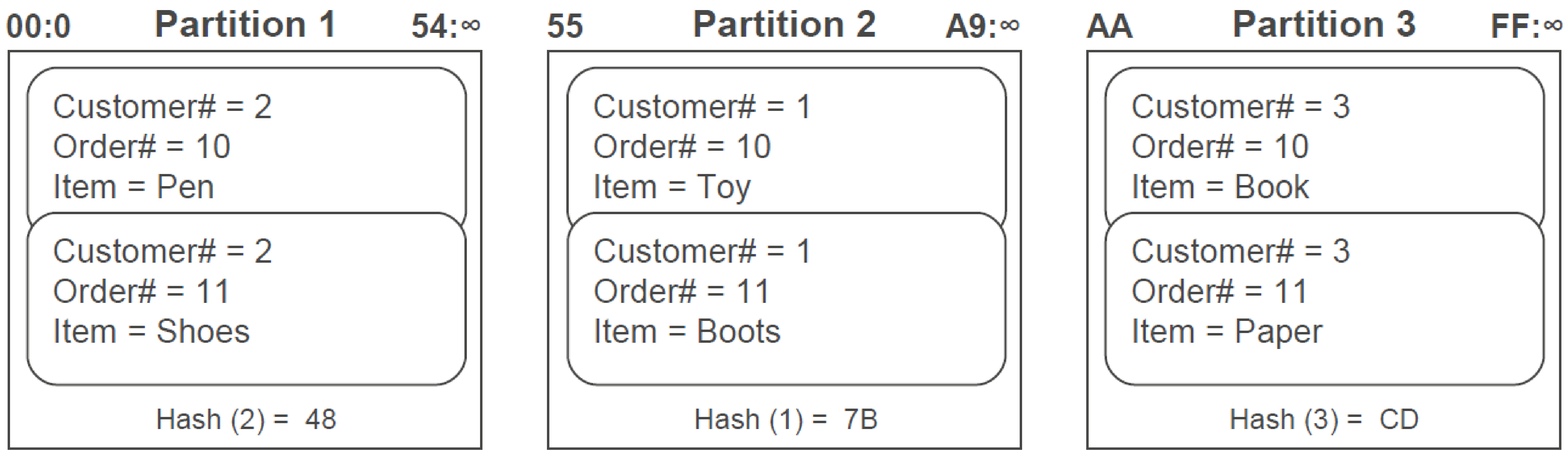
- Partition key must uniquely identifies an item in the table
- Partition Key is used for building an *unordered* hash index
- Allows table to be partitioned for scale -- create more partitions for scale up!
- Each partition will be assigned to a virtual node
  - Each partition is replicated 3 times



Partition and Sort key

- Partition and Sort Key uses two attributes together to uniquely identify an Item
  - Useful for modeling 1:N relationships
- Within unordered hash index, data is arranged by the sort key (or "range key")
- No limit on the number of items per partition key
  - Except if you have *local secondary indexes*

Partition key is customer # sort key is order #



there are three partitions for holding hash indices 00:54, 55:A9, and AA:FF respectively

Primary key and query patterns

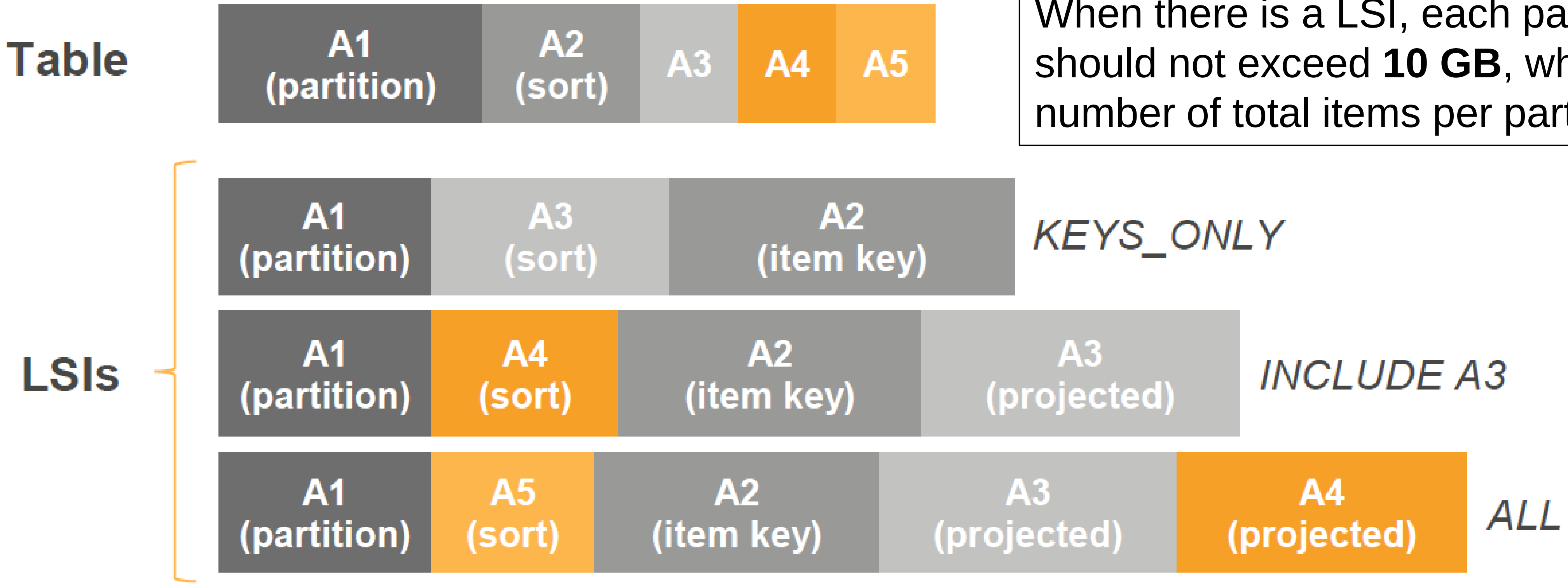
- Partition-key only: support key-value access patterns
  - fetch an item by its partition key
- Partition and sort key: enable richer queries.
  - Retrieve all items that has the same partition key.
  - Retrieve items that meet certain sort-key conditions.
    - ==, <, >, >=, <=, begins with, between, contains, in
    - counts
    - sorted results
    - top and bottom n values.

Indexes: Local secondary index (LSI)

- alternate sort key attribute
- index is local to a partition key
- enable richer access/query patterns

When there is a LSI, each partition key should not exceed **10 GB**, which limits the number of total items per partition.

We can additionally create local secondary indices on attribute A3, A4, and A5



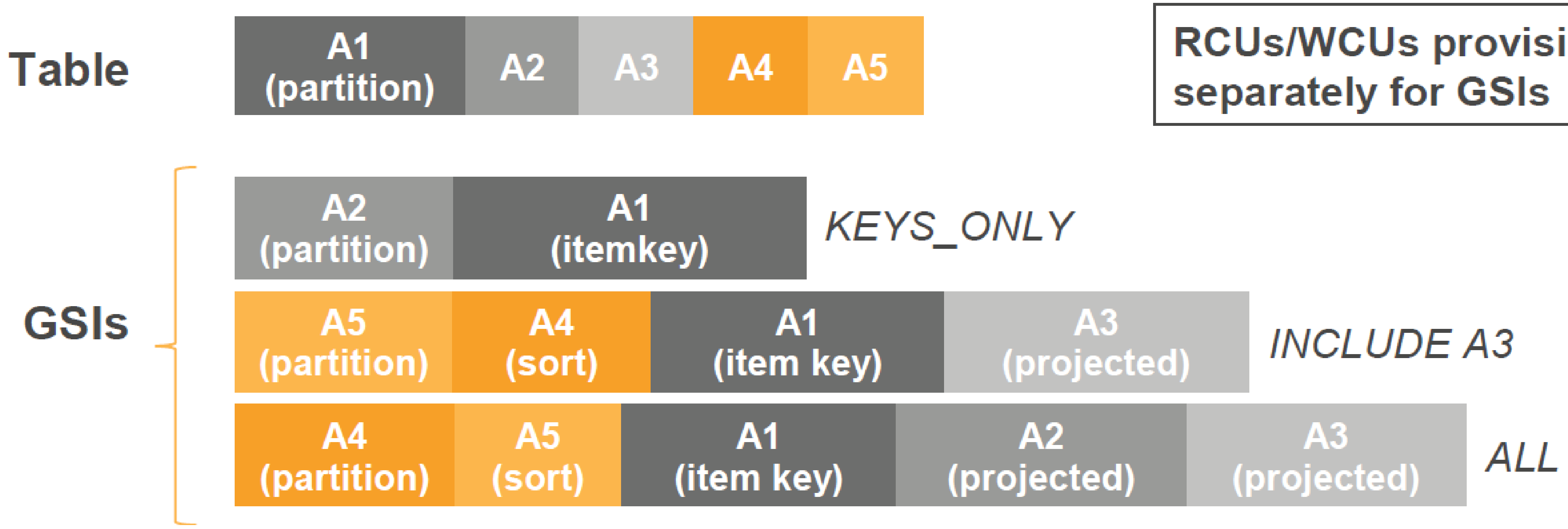
Item key here refers to the original sort key

Indexes: Global secondary index (GSI)

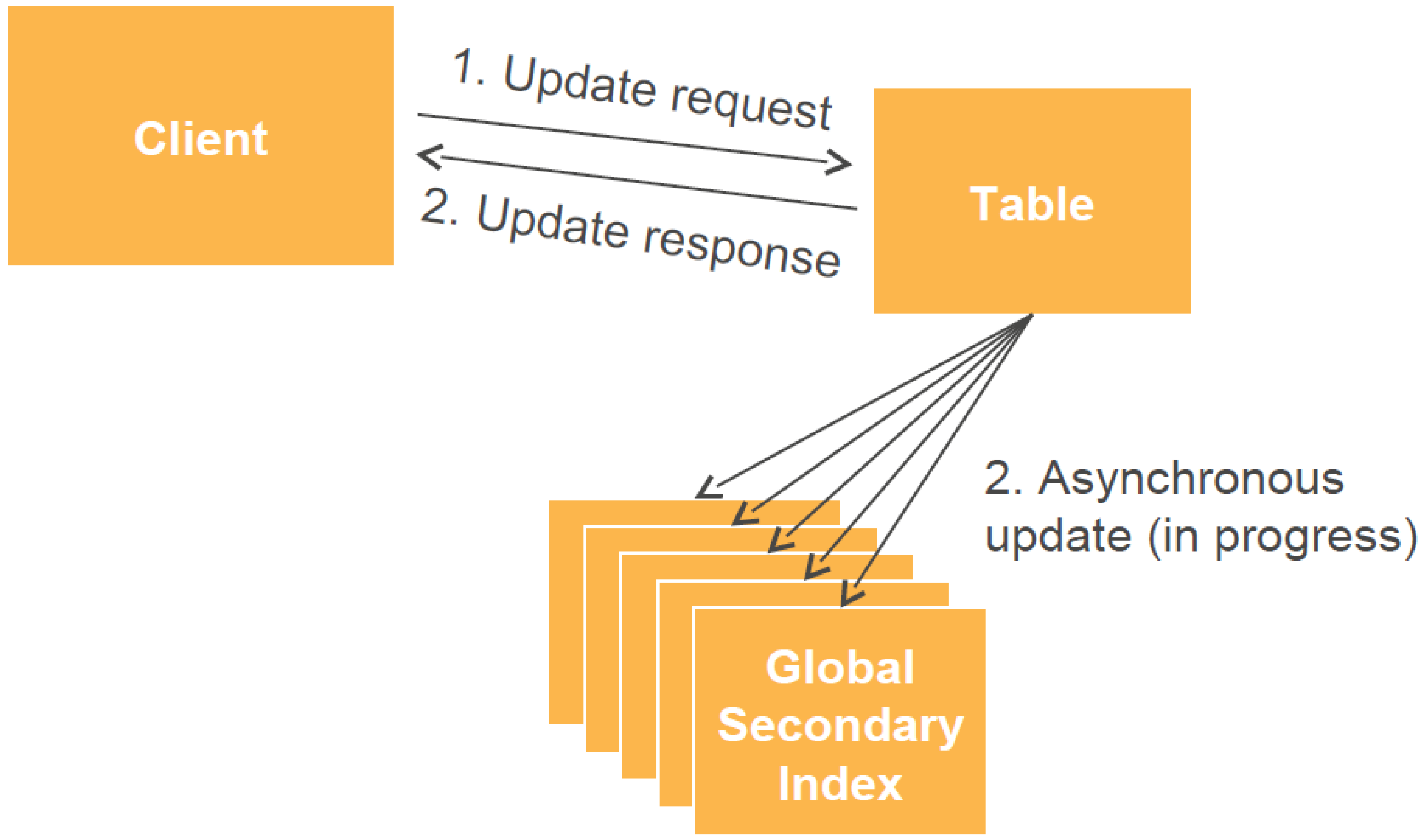
- Alternate partition and/or sort key
- Index is across all partition keys
- Use composite sort keys for compound indexes

R/WCU: read/write capacity unit

RCUs/WCUs provisioned separately for GSIs



How do GSI updates work?



- If GSIs don't have enough write capacity, table writes will be throttled!

Scaling and Throughput

- Scaling is achieved through partitioning
  - More data → more partitions
  - Higher throughput → more partitions
- Provision any amount of throughput at the table level
  - Write capacity units (WCUs) are measured in 1 KB per second
  - Read capacity units (RCUs) are measured in 4 KB per second
    - RCUs measure strictly consistent reads
    - Eventually consistent reads cost 1/2 of consistent reads
- Read and write throughput limits are independent
  - GSIs require its own WCUs and RCUs

DynamoDB: Amazon's Highly Available Key-value Store

OPERATIONS

DynamoDB: Interface

- `GetItem(table_name,key)`: returns a set of attributes for the item with the given primary key (using eventually consistent read by default).
- `PutItem(table_name,item)`: creates a new item, or replace an old item with the new item (if it already exists).
  - MD5 hashing is applied on the key to generate 128-bit identifier.

Programing Interface

Manage tables

- CreateTable
- UpdateTable
- DeleteTable
- DescribeTable
- ListTables

“Select”, “insert”, “update” items

- PutItem
- GetItem
- UpdateItem
- DeleteItem

Query specific items OR scan the full table

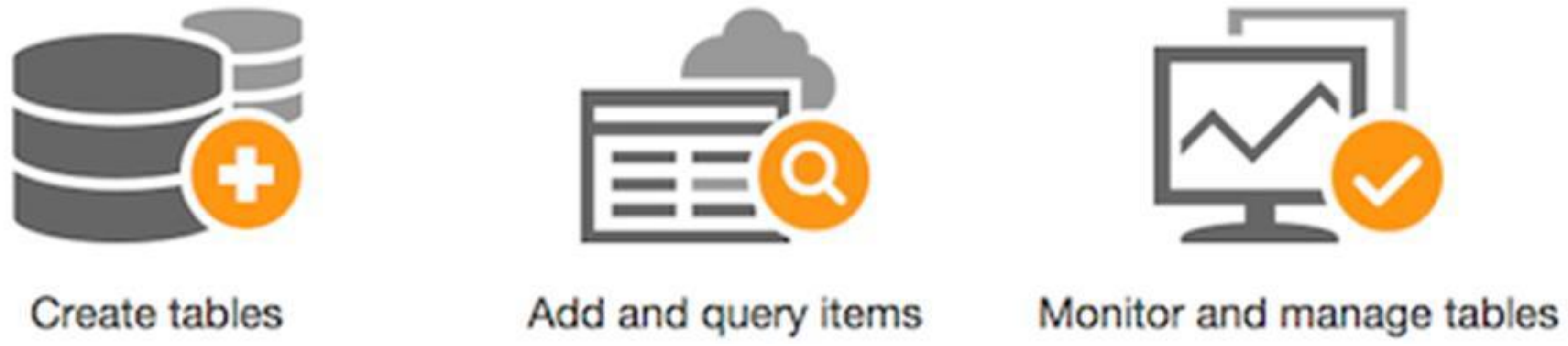
- Query
- Scan

Bulk select or update (max 1MB)

- BatchGetItem
- BatchWriteItem

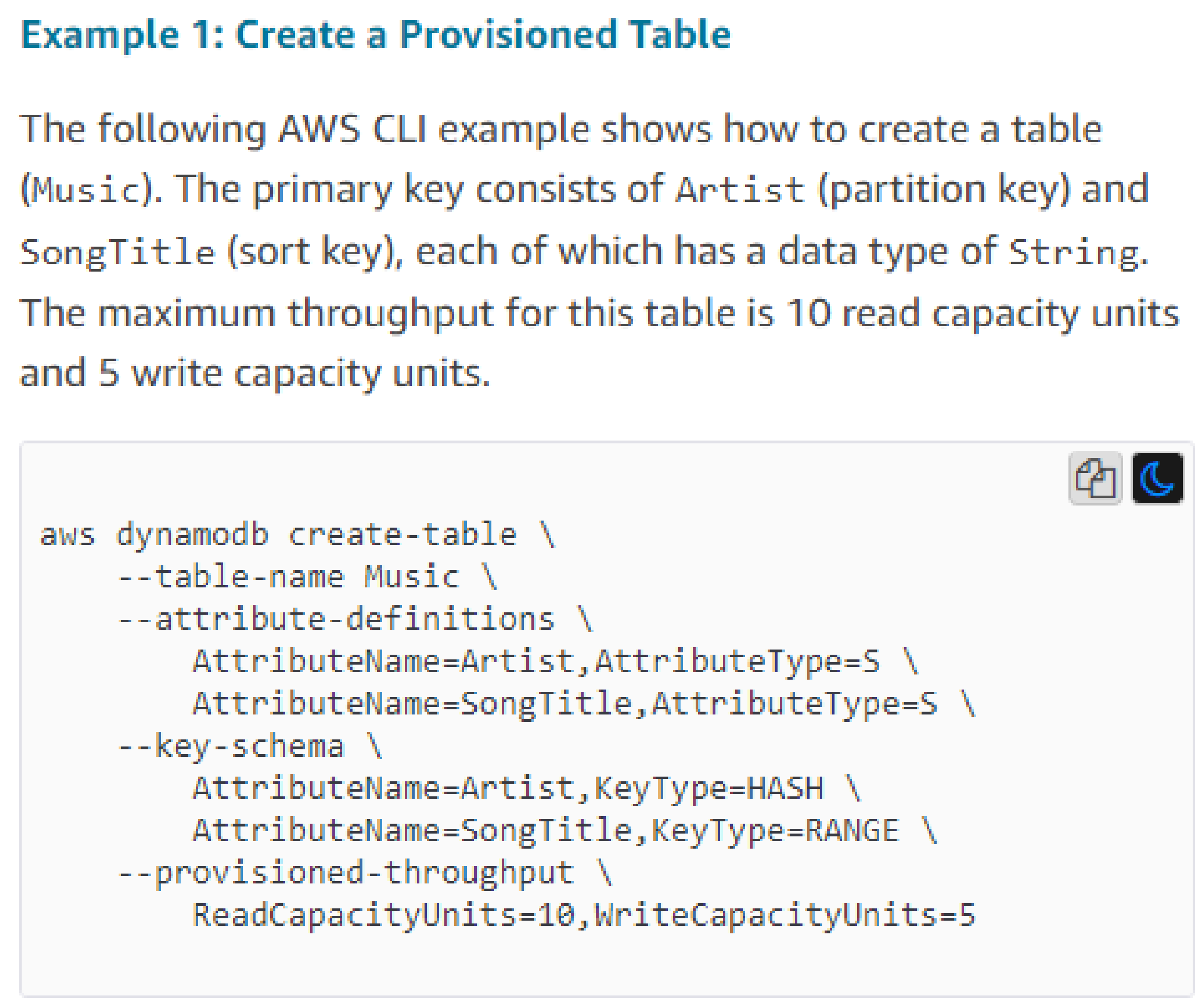
Tools for Interacting with DynamoDB: AWS Web Console

- AWS Management Console for DynamoDB
  - The GUI console for DynamoDB can be found at https://console.aws.amazon.com/dynamodb/home
- It allows you to perform the following tasks:
  - CRUD
  - View Table Items
  - Perform Table Queries
  - Set Alarms for Table Capacity Monitoring
  - View Table Metrics in Real-Time
  - View Table Alarms



Tools for Interacting with DynamoDB: AWS CLI

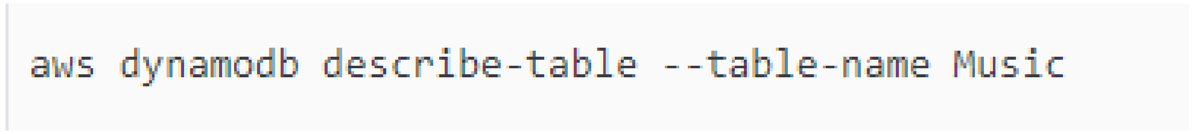
- You can using AWS's command-line interface CLI to interact with DynamoDB
  - https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/WorkingWithDynamo.html



Example 1: Create a Provisioned Table

The following AWS CLI example shows how to create a table (`Music`). The primary key consists of `Artist` (partition key) and `SongTitle` (sort key), each of which has a data type of `String`. The maximum throughput for this table is 10 read capacity units and 5 write capacity units.

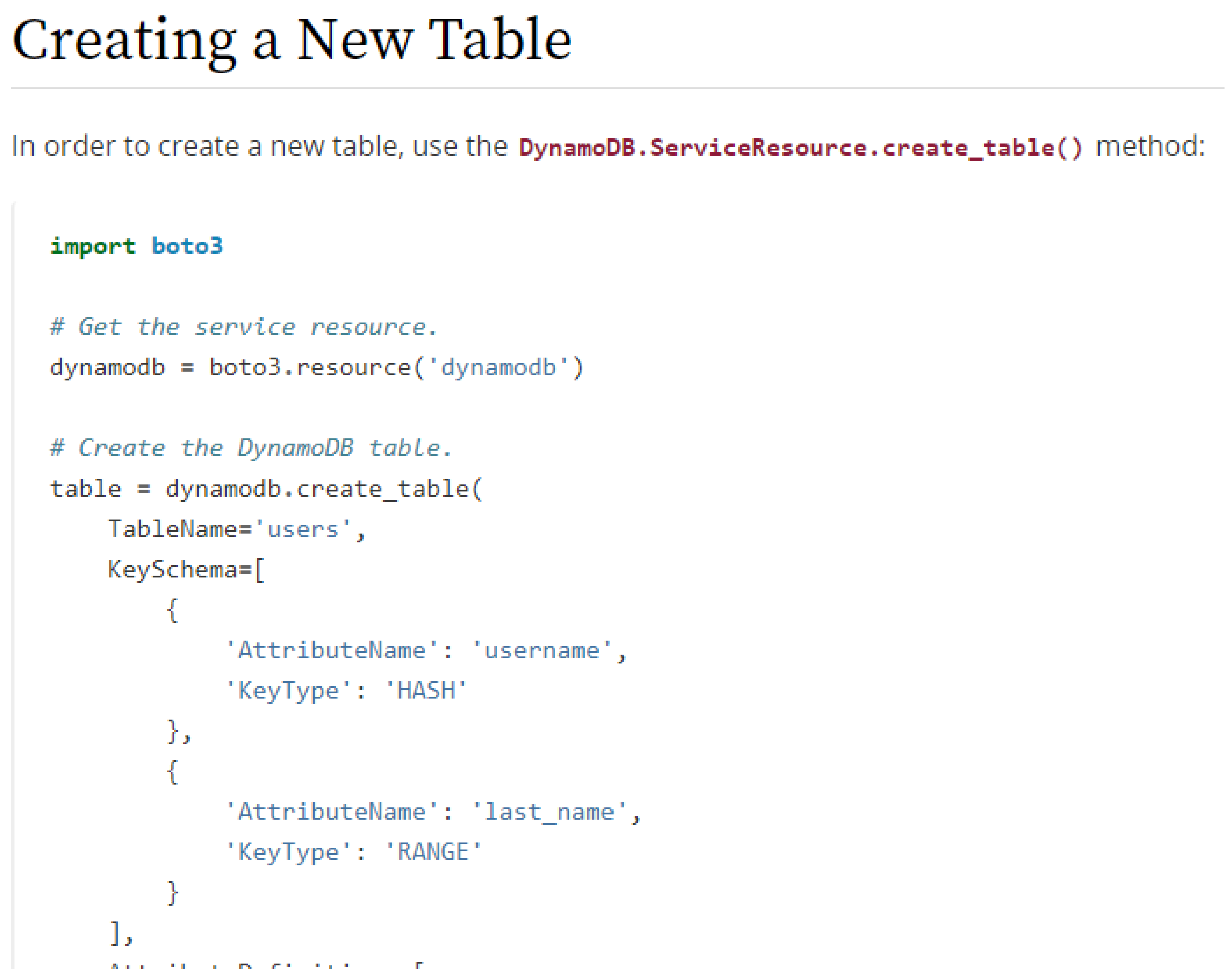
```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput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



```
aws dynamodb describe-table --table-name Music
```

Tools for Interacting with DynamoDB: AWS SDK (boto)

- You can use Python to interact with dynamoDB through the AWS SDK (bobo3)
  - https://boto3.amazonaws.com/v1/documentation/api/latest/guide/dynamodb.html



Creating a New Table

In order to create a new table, use the `DynamoDB.ServiceResource.create_table()` method:

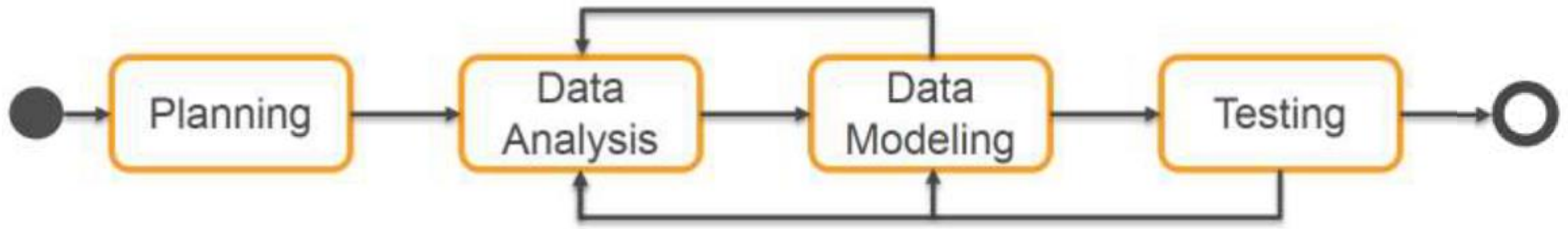
```
import boto3

# Get the service resource.
dynamodb = boto3.resource('dynamodb')

# Create the DynamoDB table.
table = dynamodb.create_table(
    TableName='users',
    KeySchema=[
        {
            'AttributeName': 'username',
            'KeyType': 'HASH'
        },
        {
            'AttributeName': 'last_name',
            'KeyType': 'RANGE'
        }
    ],
```

DynamoDB: Amazon's Highly Available Key-value Store

DATA MODELING

Data Modeling Considerations

- "To get the most out of DynamoDB throughput, create tables where the hash key element has a large number of distinct values, and values are requested fairly uniformly, as randomly as possible." *— DynamoDB Developer Guide*
  - **Space:** access is evenly spread over the key-space
  - **Time:** requests arrive evenly spaced in time
- Data modeling should be based on analyzing data and access patterns



1:1 relationships or key-values

- Use a table or GSI with an alternate partition key
- Use GetItem or BatchGetItem API
- **Example:** Given an SSN or license number, get attributes

Users Table

Partition key	Attributes
SSN = 123-45-6789	Email = johndoe@nowhere.com, License = TDL25478134
SSN = 987-65-4321	Email = maryfowler@somewhere.com, License = TDL78309234

Users-License-GSI

Partition key	Attributes
License = TDL78309234	Email = maryfowler@somewhere.com, SSN = 987-65-4321
License = TDL25478134	Email = johndoe@nowhere.com, SSN = 123-45-6789

1:N relationships or parent-children

- Use a table or GSI with partition and sort key
- Use Query API
- **Example:** Given a device, find all readings between epoch X, Y

Device-measurements		
Partition Key	Sort key	Attributes
DeviceId = 1	epoch = 5513A97C	Temperature = 30, pressure = 90
DeviceId = 1	epoch = 5513A9DB	Temperature = 30, pressure = 90

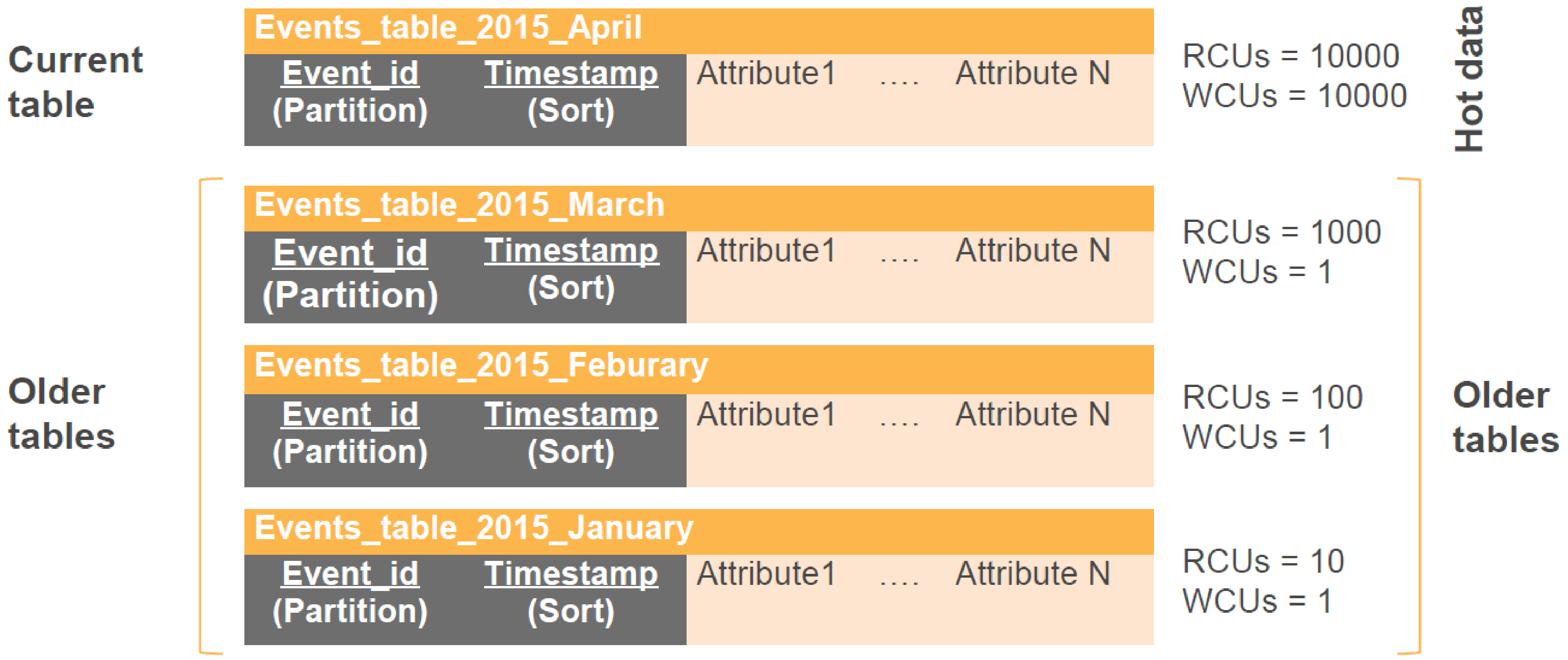
N:M relationships

- Use a table and GSI with partition and sort key elements switched
- Use Query API
- **Example:** Given a user, find all games. Or given a game, find all users.

User-Games-Table	
Partition Key	Sort key
UserId = bob	GameId = Game1
UserId = fred	GameId = Game2
UserId = bob	GameId = Game3

Game-Users-GSI	
Partition Key	Sort key
GameId = Game1	UserId = bob
GameId = Game2	UserId = fred
GameId = Game3	UserId = bob

Time Series Tables

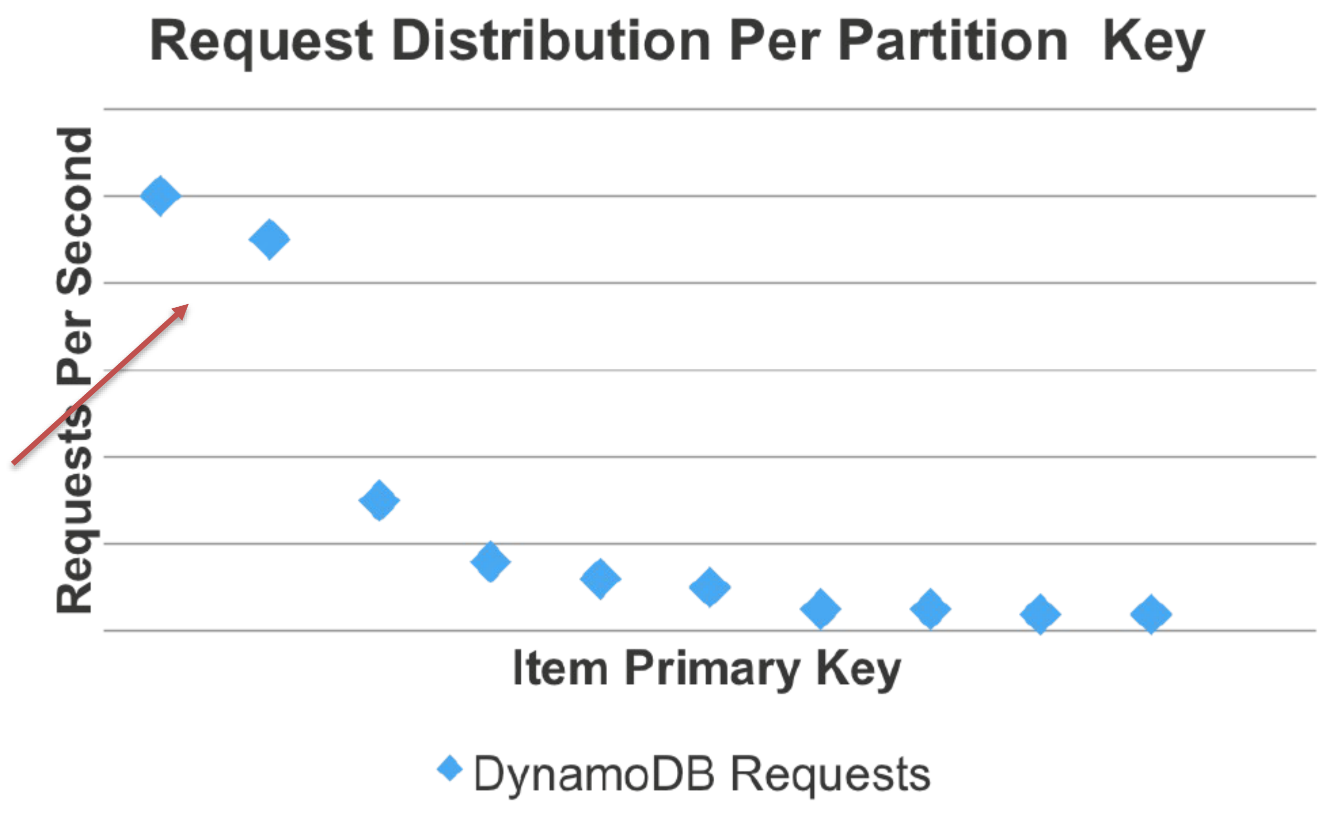


- Don’t mix hot and cold data; archive cold data to Amazon S3
- Pre-create daily, weekly, monthly tables;
  - provision required throughput for current table; Turn off (or reduce) throughput for older tables

Uneven access patterns across partition keys

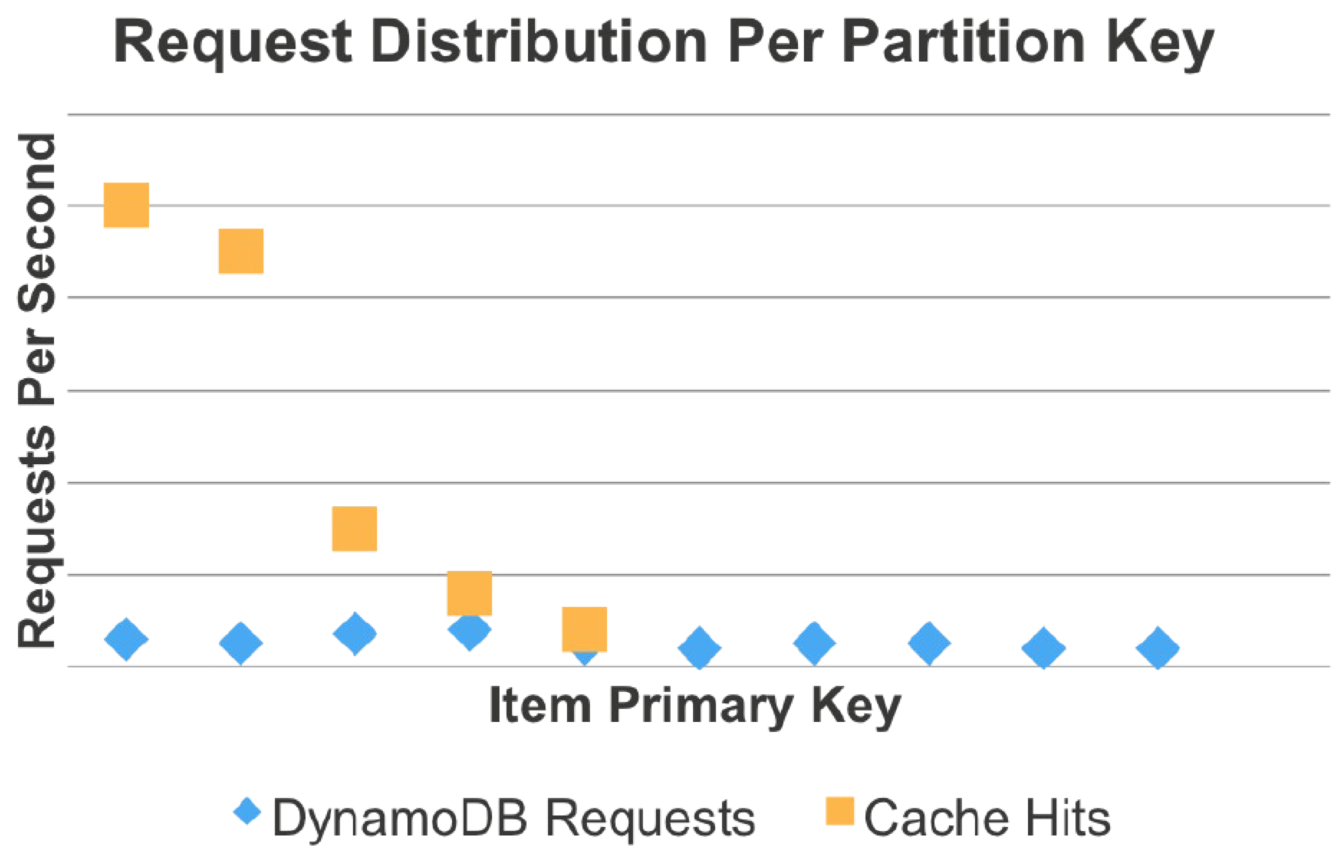
- Product catalog example

Partitions for popular items will be hit much more



Use Cache to serve popular items

- cache popular items

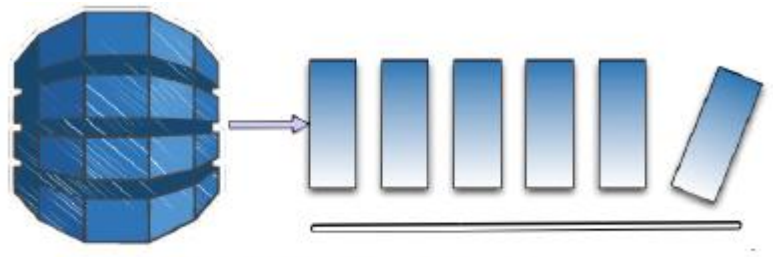


DynamoDB: Amazon's Highly Available Key-value Store

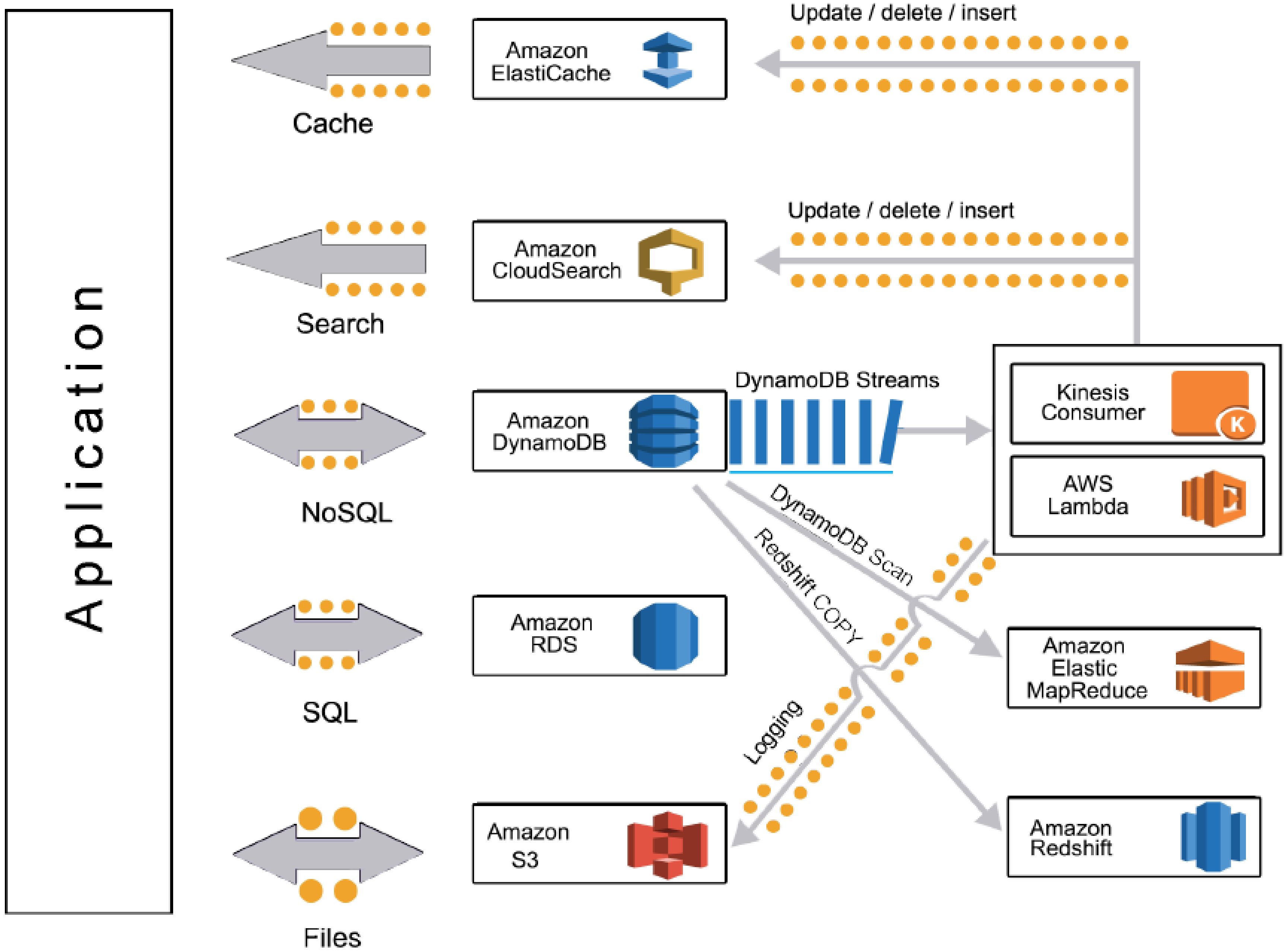
ARCHITECTURE AND INTEGRATION

DynamoDB Streams

- Stream of updates to a table
- Asynchronous
- Exactly once
- Strictly ordered
- Highly durable
- 24-hour lifetime
- Sub-second latency


DynamoDB Streams

Reference architecture


Application
Cache
Search
NoSQL
SQL
Files
Amazon ElastiCache
Amazon CloudSearch
Amazon DynamoDB
Amazon RDS
Amazon S3
Update / delete / insert
Update / delete / insert
DynamoDB Streams
Kinesis Consumer
AWS Lambda
DynamoDB Scan
Redshift COPY
Logging
Amazon Elastic MapReduce
Amazon Redshift