Amazon Dynamo DB

- Highly scalable key value store
  - Support for document operations
- Hosted service
  - Focus on ease of configuration and use
- Support for transactions and consistent reads

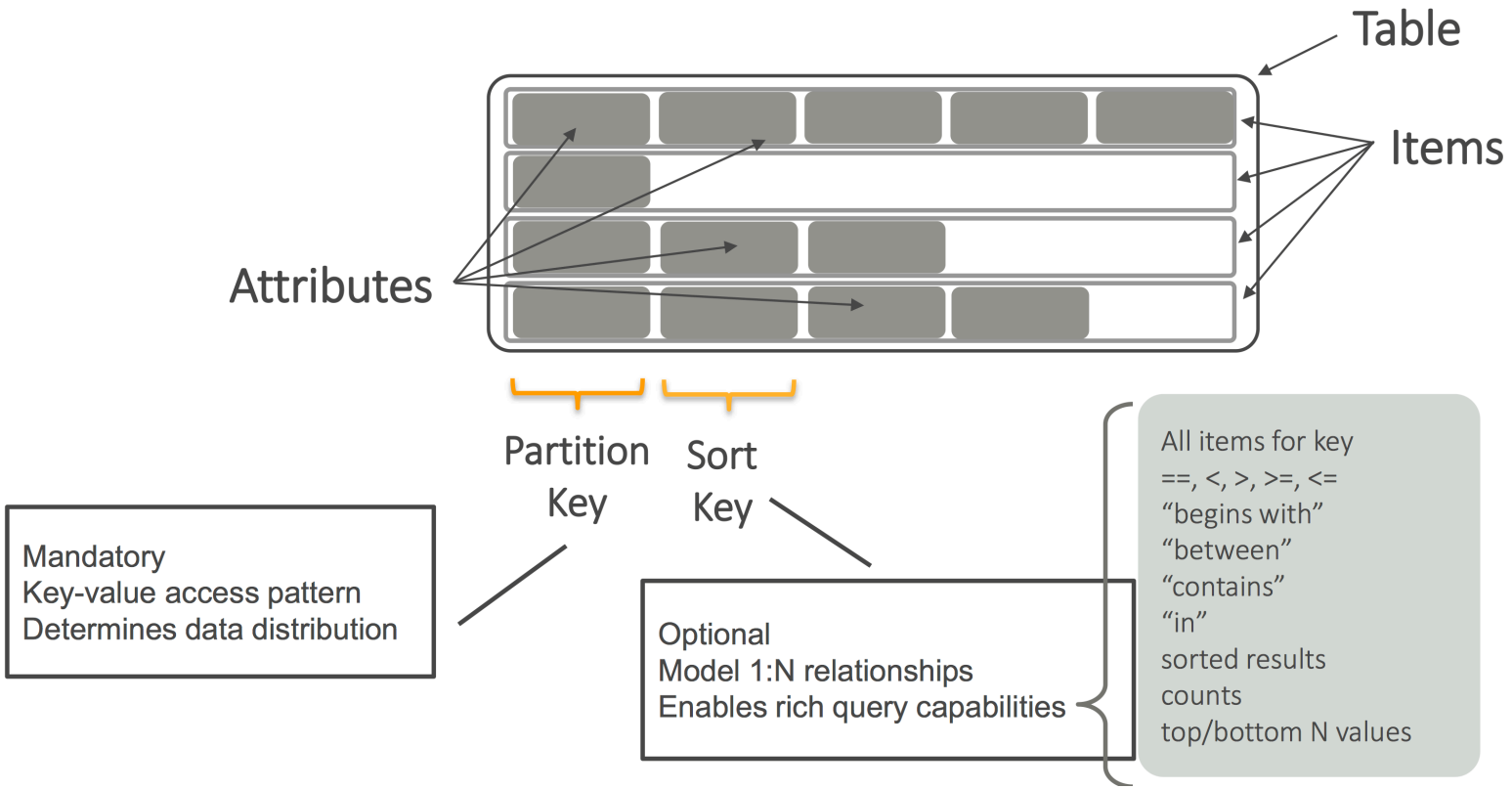
Amazon Dynamo:

- Query model
  - Simple read/write operations on small data items
- ACID properties
  - Weaker consistency model
  - No isolation, mostly single key updates
  - Transactions added reciently, but expensive
- Efficiency
  - Tradeoff between performance, cost efficiency, availability and durability guarantees

DynamoDB Data Model

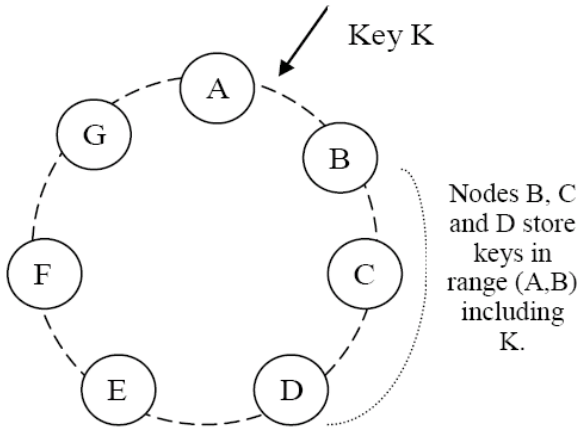
- Tables, items, attributes
  - Table contains multiple items, each item may have multiple attributes
- DynamoDB only requires you specify primary key
  - Unlike relation, attributes and data types for an item are do not have to be defined in advance
  - Total size of item is 400KB
  - Primary key specified when table is created

Core Idea: Table



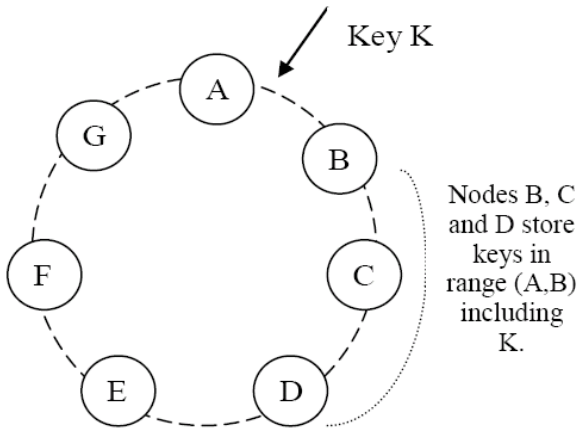
Partition Algorithm

- Consistent hashing: the output range of a hash function is treated as a fixed circular space or “ring”.
- ”Virtual Nodes”: Each node can be responsible for more than one virtual node.



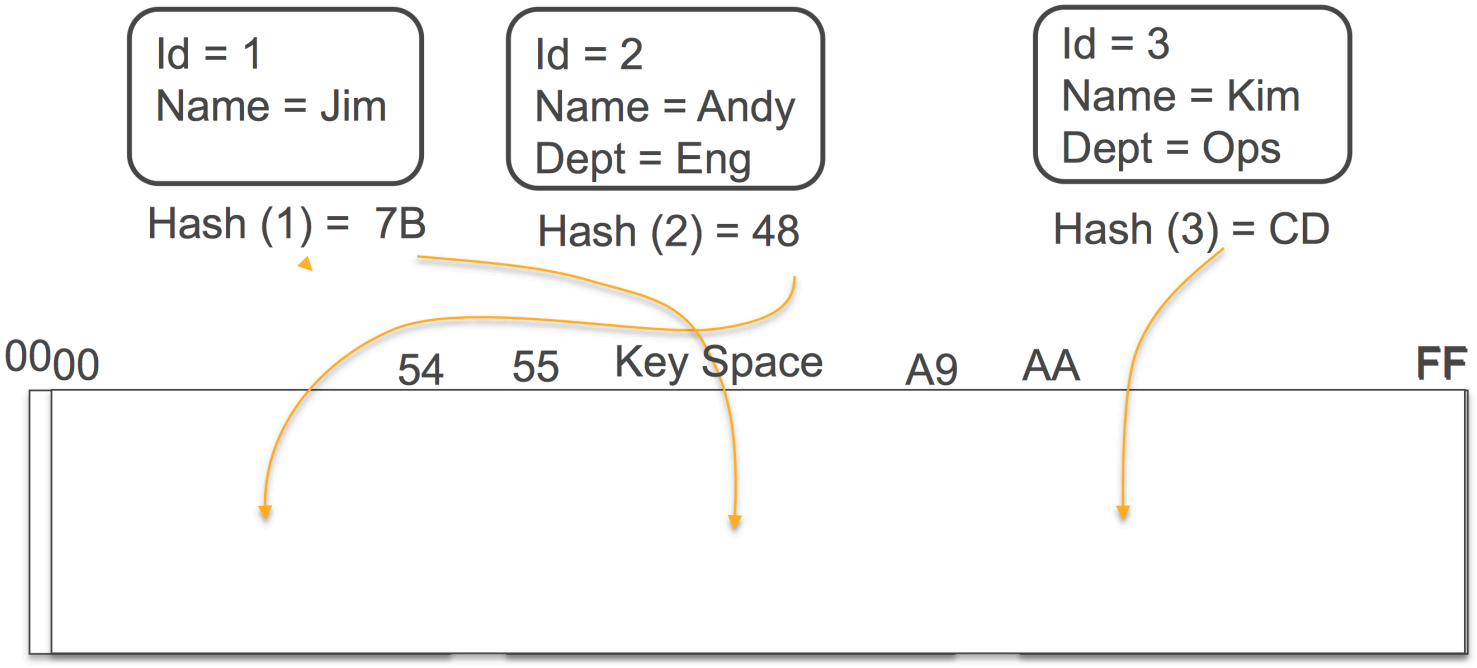
Replication

- Each data item is replicated at N hosts.
- "*preference list*": The list of nodes that is responsible for storing a particular key.



Partition Table

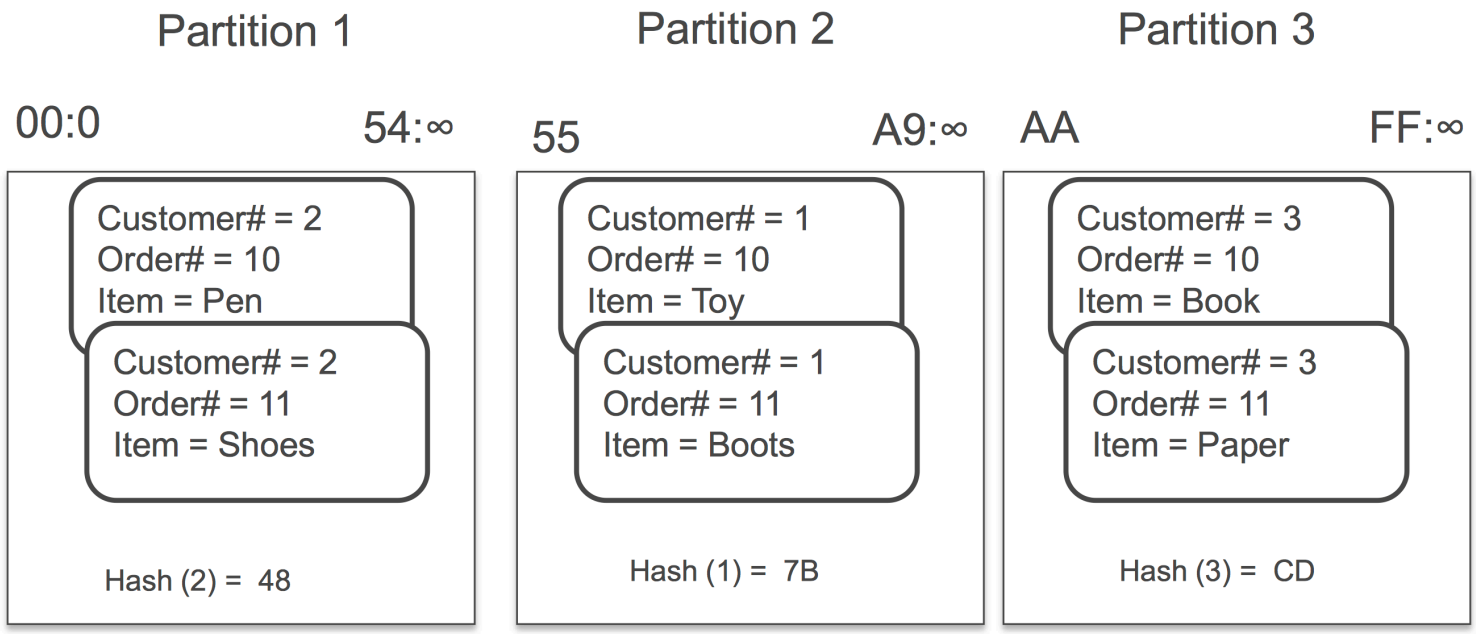
Partition Key uniquely identifies an item
Partition Key is used for building an unordered hash index
Allows table to be partitioned for scale



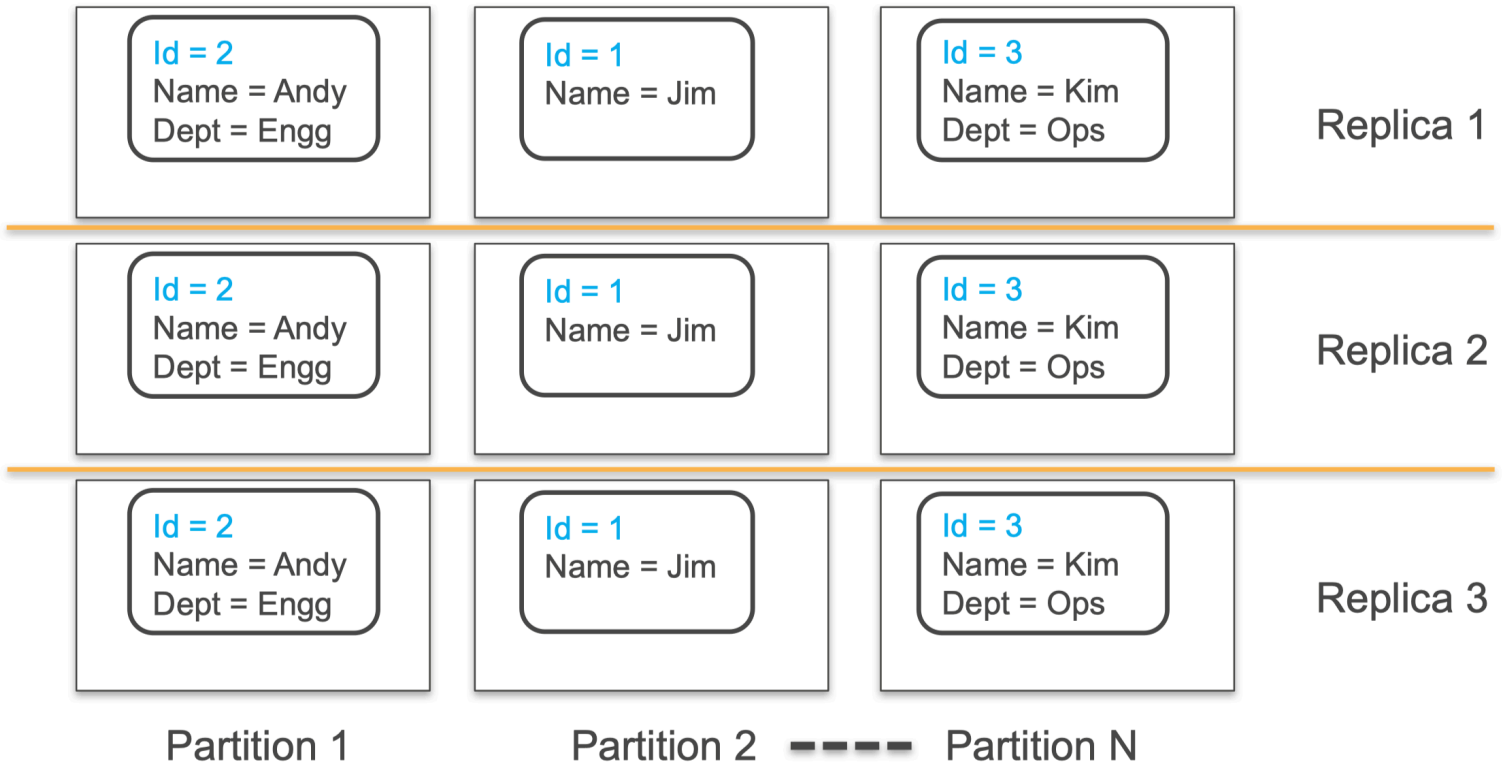
Partition-sort key table

Partition:Sort Key uses two attributes together to uniquely identify an Item
Within unordered hash index, data is arranged by the sort key
No limit on the number of items (∞) per partition key

- Except if you have local secondary indexes



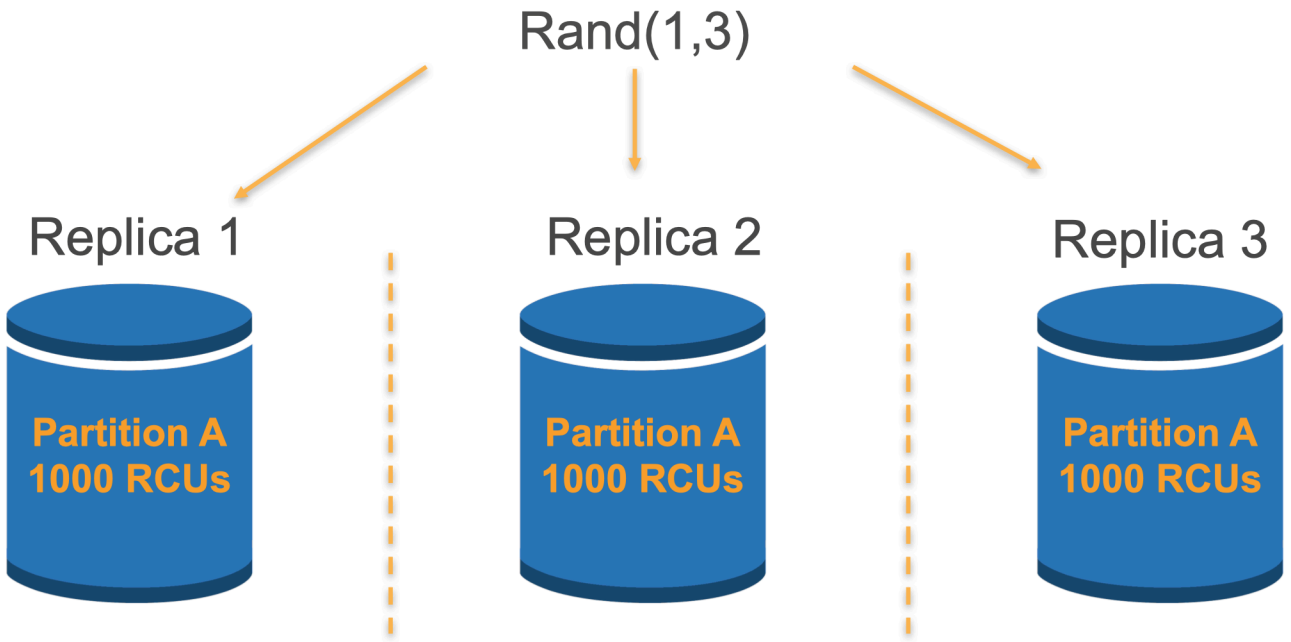
Partitions are Replicated



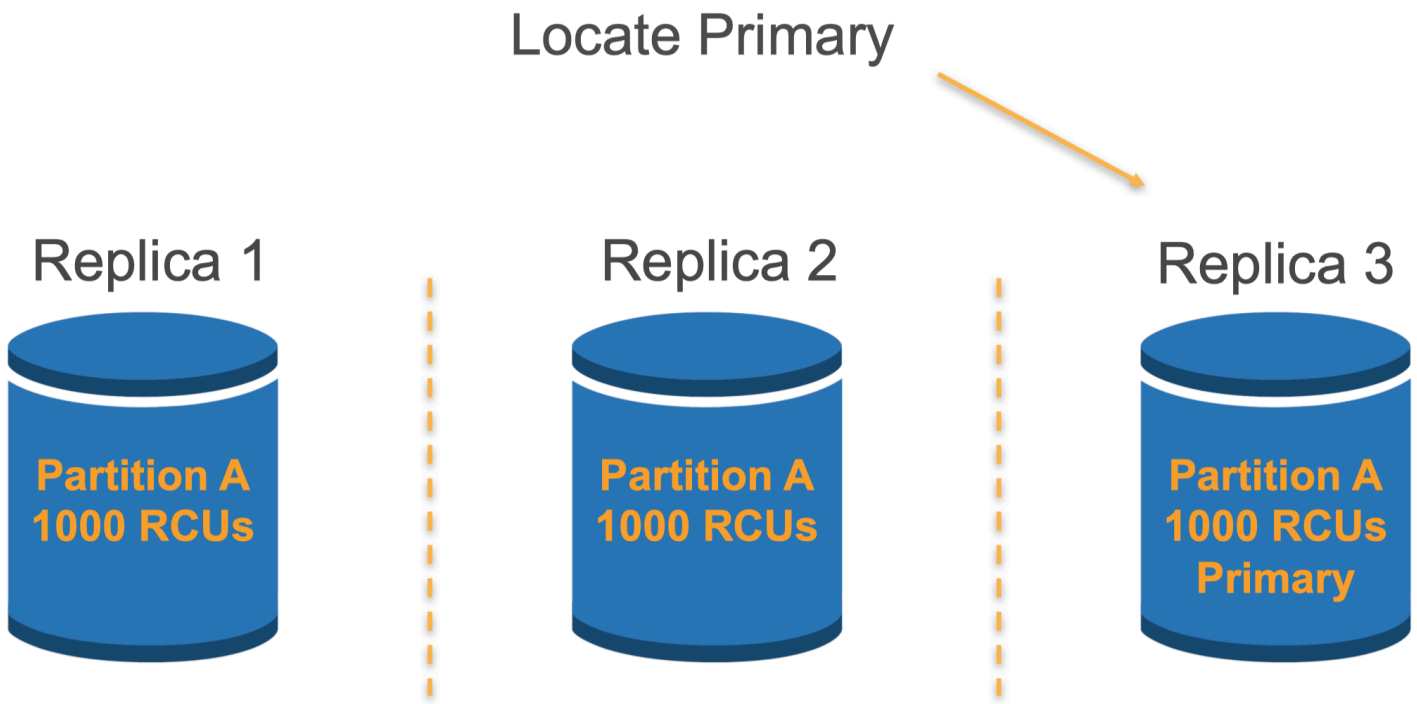
Read Constancy

- Default is eventually consistent
- Can request a strongly consistent value
  - This option may come at cost of reduced availability

Eventually Consistent Reads



Strongly Consistent Reads



Create a table….

- CreateTable API
  - Arguments: {TableName : “Music” …}
- Every table must have a primary key, which is one of its attributes
- Simple Primary Key
  - Identifed by “hash attribute”. DynamoDB uses an unsorted hash index on this value to identify items

```
{
   TableName: “Music”,
   KeySchema: { AttributeName: “Artist”, KeyType: “HASH” }
}
```

Composite Keys

- Use more then one attribute
  - One attribute is used for hash, multiple entries may have the same hash
  - Additional attributes are “ranges” and searched using sorted index, needs to be unique.
- You must provide all of the attributes for a primary key

Create Table Arguments

```
{
   TableName : "Music",
   KeySchema: [
      { AttributeName: "Artist", KeyType: "HASH", },
      { AttributeName: "SongTitle", KeyType: "RANGE" }
   ],
};
```

Can have multiple songs per artist, so first use hash on artist, then use index to find item for song.

```
{ "TableDescription": {
    "TableArn": "arn:aws:dynamodb:us-east1:1289012:table/Music",
    "AttributeDefinitions": [
       {"AttributeName": "Artist","AttributeType": "S" },
       {"AttributeName": "SongTitle", "AttributeType": "S" } ],
    "ProvisionedThroughput": {…},
    "TableSizeBytes": 0,
    "TableName": "Music",
    "TableStatus": "CREATING",
    "TableId": "12345678-0123-4567-a123-abcdefghijkl",
    "KeySchema": [
         { "KeyType": "HASH", "AttributeName": "Artist" },
         { "KeyType": "RANGE", "AttributeName": "SongTitle" } ],
    "ItemCount": 0, ”
    CreationDateTime": 1542397215.37 }
}
```

Table Operations

- CreateTable
- DescribeTable
- DeleteTable
- ListTables

Items

- Items can have multiple attributes
  - Name-value pair
  - Value can be single value, or value set.
- Items are retrieved by a GetItem operation
  - Must provide complete primary key
  - Returns all of the items attributes
  - Eventually consistent

Item Operations

- PutItem
- GetItem
- UpdateItem
- DeleteItem
- BatchGetItem
  - Read up to 100 items from one or more tables.
- BatchWriteItem
  - Create or delete up to 25 items in one or more tables.

GetItem

- Specify key, get back item
  - {"Id":{"N":"1"}}

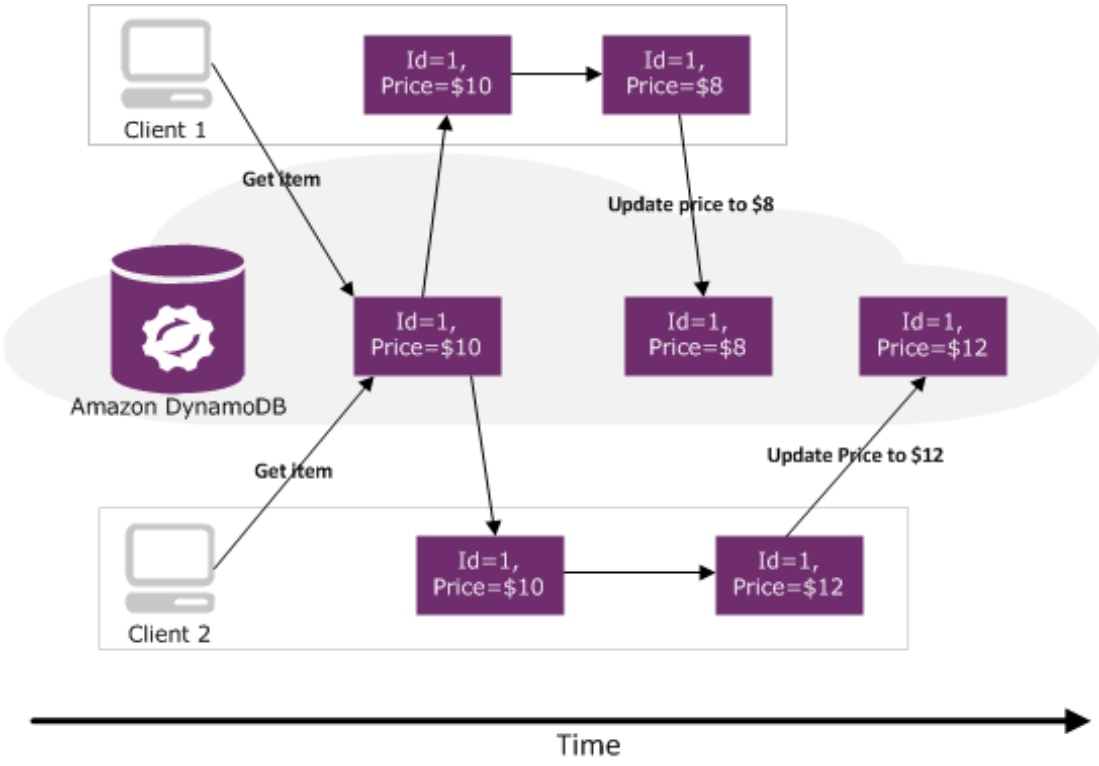
Data Types

- Scalar Types
  - String
    - Unicode with UTF8 encoding
  - Number, binary,boolean, null
- Multi-value Data Types
  - Sets: [“black”, “Green”, “Red”]
  - Documents
    - List: ordered collection of value
    - Map: unordered collection of name-value pairs

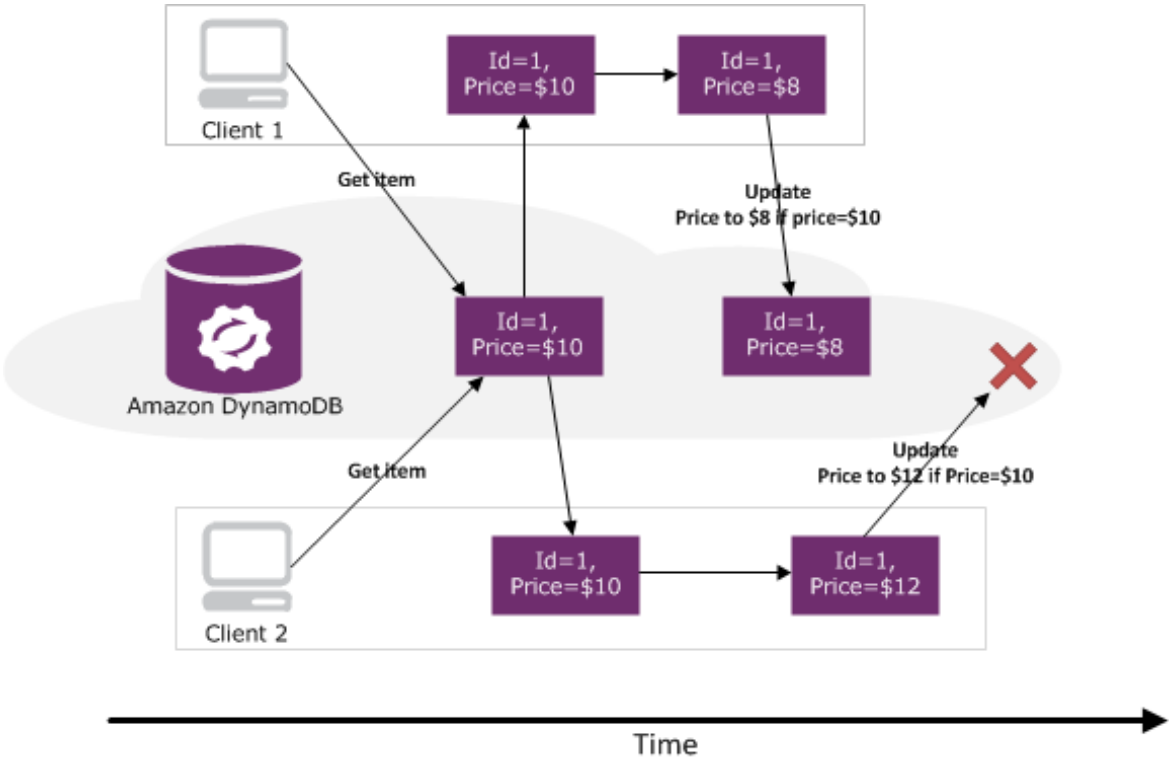
PutItem

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

Concurrent Writes



Conditional Write



Conditional Write

```
{
   “TableName”: “ProductCatalog”,
    Key: {"Id":{"N":"1"} ,
    UpdateExpression "SET Price = :newval" ,
    ”ConditionalExpression”:  "Price = :currval" ,
    ExpressionAttributeValues : {
        ”:newval”: {“N”:”8”},
        “:currval”:{“N”:”8”}
}
```

Updating an Item

- PutItem
  - Creates a new item, or replaces an existing item
- UpdateItem
  - Changes values of exiting item attributes
- DeleteItem
  - Removes item with matching primary key

Example: Map

```
{
   Day: "Monday",
   UnreadEmails: 42,
   ItemsOnMyDesk: [
      "Coffee Cup",
      "Telephone",
      {
         Pens: { Quantity : 3},
         Pencils: { Quantity : 2},
         Erasers: { Quantity : 1}
      }
   ]
}
```

Specifying Attributes

- Top level Attributes
  - Day, UnreadEmails
- Nested Attributes – [n] for list, “.” for map elements.
  - ItemsOnMyDesk[1]
  - .ItemsOnMyDesk[3].Pens.Quantity

Projection Expressions

- You can project a subset of attributed by listing them:

```
{
    “TableName”: ”ProductCatalog”,
    ”Key”: {“Id”: {“N”: ”123”}},
    ”ProjectionExpression”:
        ”Description,  RelatedItems[0], ProductReviews.FiveStar”
}
```

Attribute Names and Values

- Cannot include many names or values directly, so use a dictionary.
- Names: GET {“TableName”: “ProductCatalog”,

  “Key “: {"Id":{"N":"123"}} ,

  “ProjectionExpression”: "#pr.#1star" ,

  “ExpressionAttributeNames”:

  {"#pr":"ProductReviews", "#1star":"OneStar"}
- Values: { ":c": { "S": "Black" }, ":p": { "N": "500" } }

Query Operation

- Query uses primary key to find items in a table
  - Only works on one table, no joins!!
- By defalt all attributes are returned, but you can do a projection by specifying a *ProjectionExpression* parameter
- Can return a subset of items by using a *KeyConditionExpression* on a range attribute

What about Queries?

- To build a DynamoDB Database, we need to know the questions
- Eg:
  - Find hotels in a given area
  - Find information about a given hotel
  - Find points of interest near a given hotel

Consider this example data

Table Name	Primay Key Type	Hash Attribute Name and Type	Range Attribute Name and Type
ProductCatalog(Id, …)	Hash	Attribute Name: Id Type: Number	-
Forum(Name, …)	Hash	Attribute Name: Name Type: String	-
Thread(ForumName, Subject, …)	Hash and Range	Attribue Name: ForumName Type: String	Attribue Name: Subject Type: String
Reply(Id, ReplyDateType, …)	Hash and Range	Attribue Name: Id Type: String	Attribute Name; ReplyDateType Type: String

Queries

- Query the Thread table for a particular ForumName (hash attribute). All of the items with that ForumName value will be read by the query, because the range attribute (Subject) is not included in *KeyConditionExpression*.
  - ForumName = :name
- Query the Thread table for a particular ForumName (hash attribute), but this time return only the items with a given Subject (range attribute).
  - Forum = :name and Subject = :subj
- Query the Reply table for a particular Id (hash attribute), but return only those items whose ReplyDateTime (range attribute) begins with certain characters.
  - Id = :id and begins_with(ReplyDateTime, :dt)

Scan

- Read every item in the table
- Return every data item
  - Can use *ProjectionExpression* parameter to return only subset of columns.

What about “where” clause

- Can filter results of Query or Scan using *FilterExpression* parameter.

Filter…

- Query the Thread table for a particular ForumName (hash attribute) and Subject (range attribute). Of the items that are found, return only the most popular discussion threads—for example, those threads with more than a certain number of Views (views is a reserved word in DynamoDB, so we need to “escape” it).
  - #V > :num
- Scan the Thread table and return only the items that were last posted to by a particular user.
  - LastPostedBy = :name

Secondary Indexes

- Items in table consist of a hash key, range key and attributes
- Fast lookup on hash key or hash and range
  - Not on attribute values or range alone
- What if you want to find a value quickly?
  - Global secondary index comes to the rescue
  - Specify alternative key/range attributes

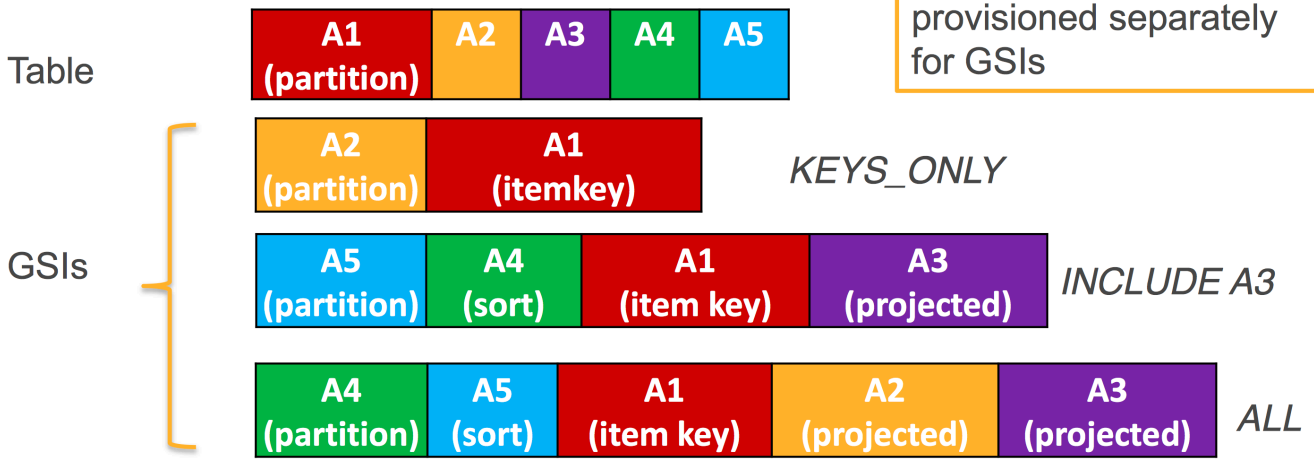
Global secondary index (GSI)

Online indexing

Alternate partition and/or sort key

Index is across all partition keys

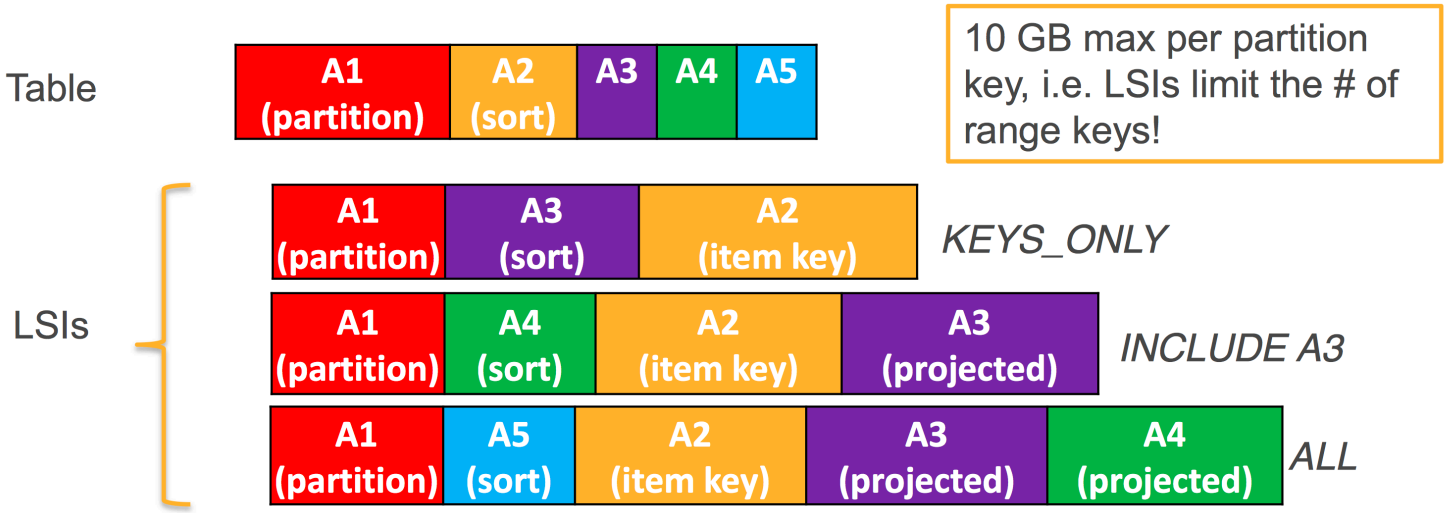
RCUs/WCUs provisioned separately for GSIs



Local secondary index (LSI)

Alternate sort key attribute

Index is local to a partition key



GameScores

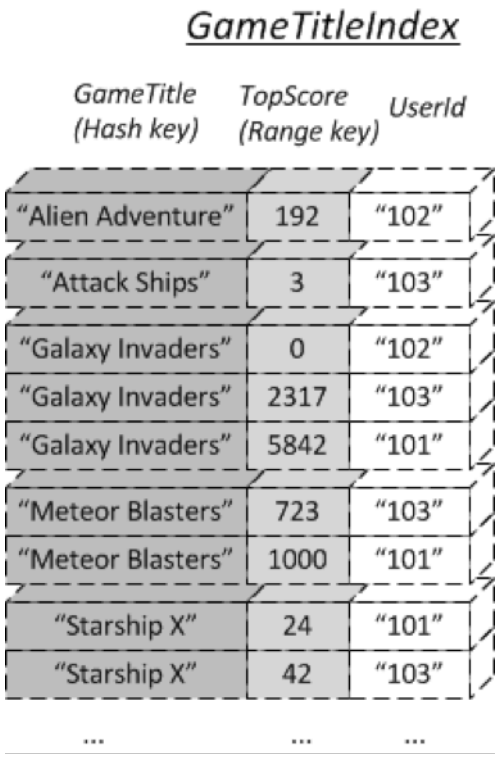
UserId (Hash key)	GameTitle (Range key)	TopScore	TopScoreDateTime	Wins	Losses
"101"	"Galaxy Invaders"	5842	"2013-09-15:17:24:31"	21	72
"101"	"Meteor Blasters"	1000	"2013-10-22:23:18:01"	12	3
"101"	"Starship X"	24	"2013-08-31:13:14:21"	4	9
"102"	"Alien Adventure"	192	"2013-07-12:11:07:56"	32	192
"102"	"Galaxy Invaders"	0	"2013-09-18:07:33:42"	0	5
"103"	"Attack Ships"	3	"2013-10-19:01:13:24"	1	8
"103"	"Galaxy Invaders"	2317	"2013-09-11:06:53:00"	40	3
"103"	"Meteor Blasters"	723	"2013-10-19:01:13:24"	22	12
"103"	"Starship X"	42	"2013-07-11:06:53:00"	4	19

From W. Vogels)

What about

- Query by user is fast, but
  - What is the top score ever recorded for the game "Meteor Blasters"?
  - Which user had the highest score for "Galaxy Invaders"?
  - What was the highest ratio of wins vs. losses?
- Would have to use a scan to do this

Secondary Global Index



Many to Many

Table	Primary Key		Data Attributes...	
	Partition Key	*Sort Key (and GSI PK)*		
	Invoice-92551	**Inv_ID:** Invoice-92551 *(invoice ID)*	**Dated:** 2018-02-07 *(date created)*	More attributes of this invoice...
		Bill_ID: Bill-4224663 *(bill ID)*	**Dated:** 2017-12-03 *(date created)*	Attributes of this bill *in this invoice*..
		Bill_ID: Bill-4224687 *(bill ID)*	**Dated:** 2018-01-09 *(date created)*	Attributes of this bill *in this invoice*..
	Invoice-92552	**Inv_ID:** Invoice-92552 *(invoice ID)*	**Dated:** 2018-03-04 *(date created)*	More attributes of this invoice...
		Bill_ID: Bill-4224687 *(bill ID)*	**Dated:** 2018-01-09 *(date created)*	Attributes of this bill *in this invoice*..
	Bill-4224663	**Bill_ID:** Bill-4224663 *(bill ID)*	**Dated:** 2017-12-03 *(date created)*	More attributes of this bill...
	Bill-4224687	**Bill_ID:** Bill-4224687 *(bill ID)*	**Dated:** 2018-01-09 *(date created)*	More attributes of this bill...

GSI

Primary Key *Partition Key*	Projected Attributes...	
Bill-4224663	**Bill_ID:** Bill-4224663 *(table primary key)*	Attributes of this bill...
	Inv_ID: Invoice-92551 *(table primary key)*	Attributes of this bill *in this invoice*..
Bill-4224687	**Bill_ID:** Bill-4224687 *(table primary key)*	Attributes of this bill...
	Inv_ID: Invoice-92551 *(table primary key)*	Attributes of this bill *in this invoice*..
	Inv_ID: Invoice-92552 *(table primary key)*	Attributes of this bill *in this invoice*..
Invoice-92551	**Inv_ID:** Invoice-92551 *(table primary key)*	Attributes of this invoice...
Invoice-92552	**Inv_ID:** Invoice-92552 *(table primary key)*	Attributes of this invoice...