**Amazon DynamoDB**

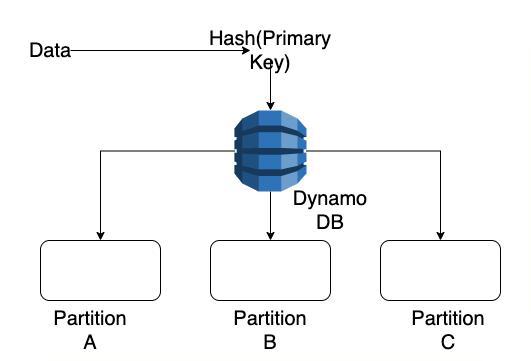
* Fast, scalable, distributed for any scale
* Flexible NoSQL Key-value & document database (schemaless) Single-digit millisecond responses for million of TPS
* Do not worry about scaling, availability or durability
  + Automatically partitions data as it grows
  + Maintains 3 replicas within the same region
* No need to provision a database
  + Create a table and configure read and write capacity (RCU and WCU)
  + Automatically scales to meet your RCU and WCU
* Provides an expensive serverless mode
* Use cases: User profiles, shopping carts, high volume read write applications

**DynamoDB Tables**

* Hierarchy : Table > item(s) > attribute (key value pair) Mandatory primary key
* Other than the primary key, tables are schemaless
  + No need to define the other attributes or types
  + Each item in a table can have distinct attributes
* Max 400 KB per item in table
  + Use S3 for large objects and DynamoDB for smaller objects

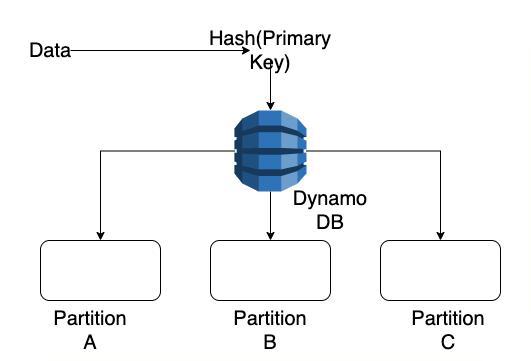
**DynamoDB - Keys**

* Two parts:
  + (Mandatory) Partition key
  + (Optional) Sort key
* Primary key should be unique
* Partition key decides the partition (input to hash function)
* Same partition key items stored together (sorted by sort key)



**DynamoDB - Indexes**

* (Optional) Secondary indexes to query on keys other than primary key
* Local secondary index
  + Same partition key as primary key but different sort key
  + Should be created at the table creation
* Global secondary index
* Partition and sort key different from primary key
* Can be added and removed at any point in time Stored separately from the original table



**DynamoDB Query vs Scan**

* Query
  + Search using a partition key attribute and a distinct value to search
  + Optional - sort key and filters
  + Results are sorted by primary key Max 1 MB
* Scan
  + Reads every item in a table
  + Expensive compared to query Returns all attributes by default Supports paging above 1 MB Filter items using expressions

DynamoDB Consistency Levels

* By default, eventually consistent (lag of about a second)
* Request for strongly consistent reads
  + Set ConsistentRead to true
  + Slow and more expensive
* Supports transactions
  + All-or-nothing changes to multiple items both within and across tables
  + Twice the cost

**DynamoDB Read/Write Capacity Modes**

* Provisioned
  + Provision read and write capacity
  + Dynamically adjustable
  + Unused capacity can be used in bursts
  + You are billed for the provisioned capacity irrespective of whether you make use of it or not
* On Demand
  + Truly serverless and expensive
  + For unknown workloads or traffic with huge spikes Use On Demand only when your
  + Workloads are really spiky causing low utilization of Provisioned Capacity OR
  + Usage is very low (for example, in test environments) making manual adjustments expensive

**DynamoDB – Operations**

* Performance Monitoring - CloudWatch
* Alerts on RCU, WCU and Throttle Requests - CloudWatch Alarms
* Migrate data from RDS or MongoDB to DynamoDB - AWS Database Migration Service
* (Feature) Enable point-in-time recovery (max 35 days)
* Use Time to Live (TTL) to automatically expire items

**DynamoDB - IAM and Encryption**

* Server-side encryption in integration with keys from KMS
  + Always enabled
  + Automatically encrypts tables, DynamoDB streams, and backups
* Client-side encryption with DynamoDB Encryption Client
  + You can manage your keys with KMS or CloudHSM
* Use IAM roles to provide EC2 instances or AWS services access to DynamoDB tables
  + Predefined policies available for DynamoDB
    - AmazonDynamoDBReadOnlyAccess
    - AmazonDynamoDBFullAccess etc
  + Fine-grained control at the individual item level

**DynamoDB Accelerator (DAX)**

* In-memory caching for DynamoDB providing microsecond response times
  + Typical DynamoDB response times - single-digit milliseconds
* Very few changes needed to connect to DAX
  + Can reduce your costs by saving your read capacity units
* Not recommended
  + If you need strongly consistent reads or
  + Your application is write-intensive with very few reads