The DynamoDb is an fully-managed region level service, that works based on server-less technology. We dont need to provision/create an instance of DynamoDB database like RDS Service. It is provisioned by the AWS Cloudplatform per region level and is available for every cloud account users to creating and storing the data.

Here the region-level service means, when we create a table in DynamoDB database we dont need to choose an vpc or availability zone or subnet in creating the table. Upon creating a table within a region, it is accessible across all the vpcs, availability zones and subnets of the region. When we create a table with name as "Customer" in ap-south-1a region, we cannot create one more table with name "Customer" under the same region.

in a table we store data in collections

For each type of information we need to create a table in DynamoDB

How do we store data in DynamoDB?
To store the data in DynamoDB we need to create tables. Per each type of data we want to store we need to create one table. While creating the tables we dont need to define fixed-set of columns. Because no-sql means "not all the data contains same fields of values". or "not all the data has same structure".

In the tables we need to create collections, a collection is nothing but a record in RDBMS Table. In each collection we store key/value pair of data where in the key/value could be different for different collections of data we stored.

While creating a Table in DynamoDB we need to create a mandatory column called "partition_key", it is mandatory to have partition_key for every table we create, because based on partition_key only the DynamoDB distributes the data and stores across.

when we store a collection of items in a DynamoDB table, the DynamoDB takes the partition_key value and computes the hash of the keyvalue to determine in which partition of the table the collection should be stored.

The partition_key acts as an primary key for the collections in the table. no 2 collections cannot have the same value for partition key.

customer (table)

(partition_key: customer_no)
customer_no=100, customer_nm=joe, age=23, gender=male
customer_no=101, fullname=mathew t, dob=10/01/1999, gender=male
customer_no=102, customer_nm=bob, age=25, gender=male, mobile_no=938494, email_address=bob@gmail.com

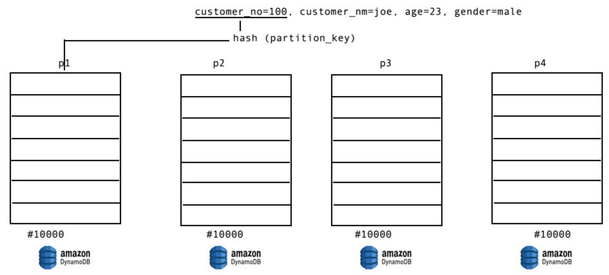
How does the DynamoDB identifies the partition in which the collection should be stored?
It uses hashing technic.
hash is a function that computes a unique value based on the data we passed as an input

h(data) = unique hashvalue
1. no 2 different data will computes to the same hashvalue
h(david) = h03
h(bob) = h04, but will not be h03

2. always for the same data, the hashvalue that is computed will be same
h(david) = h03
h(david) = h03

3. from the hashvalue we cannot compute the original data back
h(david) = h03
from h03 -> we cannot find the original data back

DynamoDB while storing the data in tables, it uses partitioning technic to store and access the data quickly.



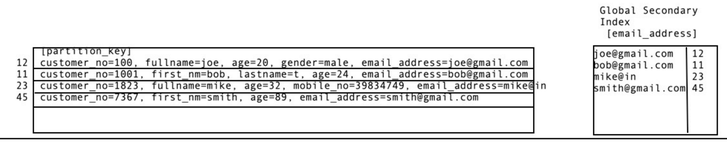
1. for a table if we define only partition_key it acts as an unique_column (primary key) and no 2 collections of the table can have same partition_key value
2. while creating table, along with partition_key we can create or define another key called "sort_key" and it is optional and is recommended to have for a table. Incase if the table has been defined with both partition_key and sort_key then the combination of both these keys should be unique. In this case the partition_key of any 2 collections can be same. but the combination of these 2 cannot be same as shown below:

customer_first_nm (partition_key)	customer_last_name (sort_key)
Alexandar	P
Alexandar	J

While searching or looking up for the data within the table, we always need to search through the search index as
1. if the table has only partition_key, then search index is : partition_key
2. if it has both, partition_key & sort_key then search index is : both of them

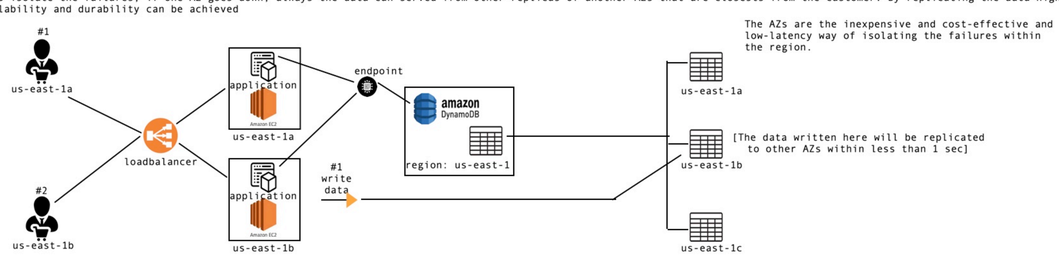
apart from partition_key or partition_key + sort_key we can search for the collections with any other attributes or keys of the collections in the table, the process of searching the data with other attributes of the table is called "scan"
"scan" is very costly interms of performance and time and should be avoided. Because DynamoDb has to scan through all the collections stored across different partitions that are distributed across the AZs of the region of that table to identify the data.

To overcome the problem with scan, the DynamoDb has introduced Global Secondary Index (GSI). We can choose any other item which is common across all the collections of the table and ask DynamoDB to create an GSI on that item. Now searching and locating the data will be quick even using GSI index as well. But DynamoDB charges additional for creating each GSI on the item for the table.



Read Consistency / Write Consistency
---
upon creating a table in DynamoDb it replicates the table across all the availability zones of the region in which it has created.
Replication means = The same copy of the table is stored across all the AZs of the region.

There are 2 reasons for replicating the table across the AZs
1. to reduce the latency in accessing the data from the Table
2. to isolate the failures, if one AZ goes down, always the data can served from other replicas of another AZs that are closests from the customer. By replicating the data high-availability and durability can be achieved



Eventual Consistent Read

When we try reading the data from DynamoDb table, the results may not include the recent writes to the table. Because upon a write operation the data always will be written to the closest AZ region replica and made it replicated across other AZs over a less than 1 sec of time. between this interval a read request comes may not include the recent writes. So the data written/read will be eventually consistent.

Always the write operations are eventual consistent

Strong Consistent Read

When we requesting to read the data from DynamoDB table we can specify strong consistent read, so that it fetches the data by querying from all the replicas across the AZ of the region in which the table is replicated and returns the latest snapshot of the data. But there are lot of dis-advantages with strong consistent read:
1. if a strong consistent read is taking more amount of time (due to network or any other issues), then it results in timeout operation and reports it as 500 error to the client
2. strong consistent reads takes more time in access the data than eventual consistent reads
3. strong consistent reads cannot be performed on GSI
4. strong consistent reads requires more read capacity.

Read Capacity and Write Capacity
---
since dynamodb is server-less database, there is no fixed computing/shape/capacity alloted to an aws account, so they cannot charge based on computing capacity being alloted. The only way we will be charged is based on utilization which means how many data we stored and how many read/write operations we are performing on the tables

There are 2 charges that are imposed while using the DynamoDB:
1. storage cost
2. read/write operations on the table

How does the read/write operations are charged, on which basis?
The read and write units consumed are the once based on which we will be charged in Dynamodb

Read Capacity:
1. on read [request unit] represents
  2 eventual consistent read requests upon the item size of 4kb
  1 strongly consistent read request upto the item size of 4kb
if you are reading an item/collection of more than 4kb in size then dynamodb allocates more read units for processing request

Write Capacity:
1. one write request unit [write unit] represents:
  1. one write upto 1 item of size: 1kb
  if it is an transactional write then 1 write operation of 1kb size consumes 2 write units
if we are writing more than 1kb size then dynamodb allocates more write units for processing the request

during the time of creating the DynamoDB table we can choose the read/write units to be allocated to the table for controlling the cost.
For e.g if we create a table
With read capacity as 2 units and write capacity as 1 unit means:

per second: DynamoDB allows 2 read units on the table and only 1 write unit on the table, so if we allocate less read/write capacity, then more no of request will be blocked/kept waiting if it cross the allocated units.

DynamoDB allows us to allocate read/write capacity of a table in 2 modes
1. provisioned
2. ondemand

1. provisioned
The cloud engineer is going to allocate fixed read/write capacity on the table while creating allowing to perform operations. So that the read/write operations are allowed based on the enforce limits only. By default the table will be created with provisioned mode

2. ondemand
aws will automatically scaleup/scaledown the read/write capacity of the table based on usage.