Dynamo db

Sharding is **a method of splitting and storing a single logical dataset in multiple databases**. for large data
complex queries become difficult, joining become difficult.

CAP theorem: it is impossible for a distributed data store to simultaneously provide more than two feature out of three .

c-consisiteny, a -avilabilty, p= partition tolderance(system continue inspite of network failure)

sql select c and a aland lacks p
nosql select c or a and p

sql: optimized storage, scla vertically

nosql

Denormalized Hierarchical

optimise for compute, scale horizontally, instationed views used for oltp scale

nosql engine type:
key value
documents
column oriented: store data of each column in a block
graph : fraud detection, recommendation website

Basic Availability: the db appear work most of the time
soft scale: stores need not to write consistent nor do diff replica have to mutually consistent
Eventual consistency: store consistency at some later point of time

weak consistency, faster, availability first, approximate ans are accepted, easier schema evolution

scaling in sql is costly, scale back down is impossible

best advantage of no sql is horizontal scaling (adding more machine, sharding)

Dynamo Db

non-relational database, supports both key value and document data models.
consistent responsiveness, single digit millisecond latency, virtually unlimited throughput and storage. ,automatically scale up or down ,can handle trillions of requests per day.
supports ACID transactions.
don't really have to make too many trade-offs between ACID and BASE compliance.

get on-demand backups and point in time recovery.This helps you protect your DynamoDB tables from accidental writes or delete operations. don't have to worry about creating, maintaining,or scheduling any on-demand backups.encryption at rest for all of your data data is highly durable.It's replicated across multiple AWS availability zones.
And with DynamoDB, you get a service level agreement with up to 99.999% uptime.

Architecture:
Partition key: Primary key. composed of one attribute calle partition key. also called hash attribute
sort key: optional , 1 to many relationship

partition ans sort key combined composite primary key

Dynamo Db performance:
1 on demand capacity: db scales according to workload
2 provisioned capacity: allow to have consistent and predictable performance
specify read and write capacity throughput, have less charges than on demand

Item: unique gp of attribute(similar to row)

Data Type:
1. Scalar : exactly one value- string number bool, must encode in base 64
2. Document: complex structure with nested attribute(eg json)- list , maps, set

DynamoDB uses primary keys to uniquely identify each item in a table and secondary indexes to provide more querying flexibility.
Secondary index: contain a subset of attribute from table along with alternate key to support query operation
you can have more than one secondary index
DynamoDB maintains indexes automatically. When you add, update, or delete an item in the base table, DynamoDB adds, updates, or deletes the corresponding item in any indexes that belong to that table.

Amazon DynamoDB supports two types of secondary indexes:

- **Global secondary index** – An index with a partition key and sort key that can be different from those on the table.

- **Local secondary index** – An index that has the same partition key as the table, but a different sort key. must be created at time of table creation

DynamoDB Streams is an optional feature that captures data modification events in DynamoDB table

consistent hashing
eg k=mark

hash(k)=123456
selected node= hash(k)mode N= 52(let) if it is not avaiable then we move forward to find avaiable sotrage node
N= storage node

Provisioned throughput: the maximum amount of capacity that an application can consume from a table or index.
Read capacity unit(RCU)

Each API call to read data from your table is a read request. Read requests can be strongly consistent, eventually consistent, or transactional. For items up to 4 KB in size, one RCU can perform one strongly consistent read request per second or 2 eventually read request per second

Strong Consistency offers **up-to-date data but at the cost of high latency**. While Eventual consistency offers low latency but may reply to read requests with stale data since all nodes of the database may not have the updated data

Filtered query and scan consume full read unit

Query is faster than scan
multiple get item using api call:batch item

Batch get item: return attributes for multiple items from multiple tables, retrieves item in parallel to reduce latency
Batch put item: put or delete multiple items from multiple tables, write item in parallel to reduce latency(threading)

projecting set of attributes copied from table to secondary index
Dynamodb writes a corresponding index item only if index sort key is present in the item.If sort key doesn't appear in every table the index is called sparse.
GSI are sparse by default
ON Demand: overall backup of your db on demad, consitent
p**oint time:** incremental recovery, on prevent from accidental data losss