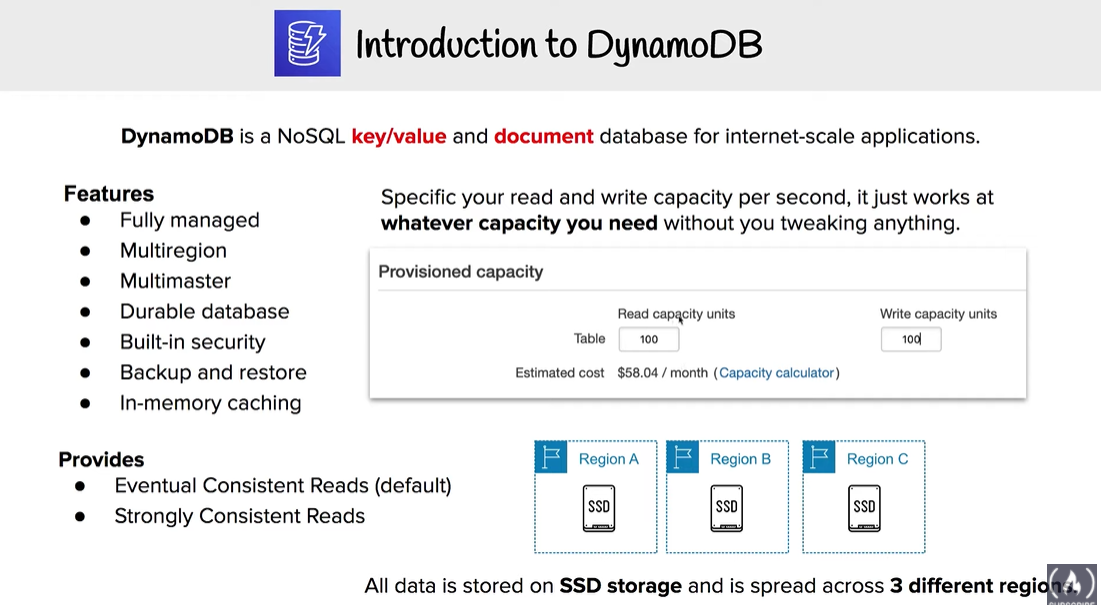
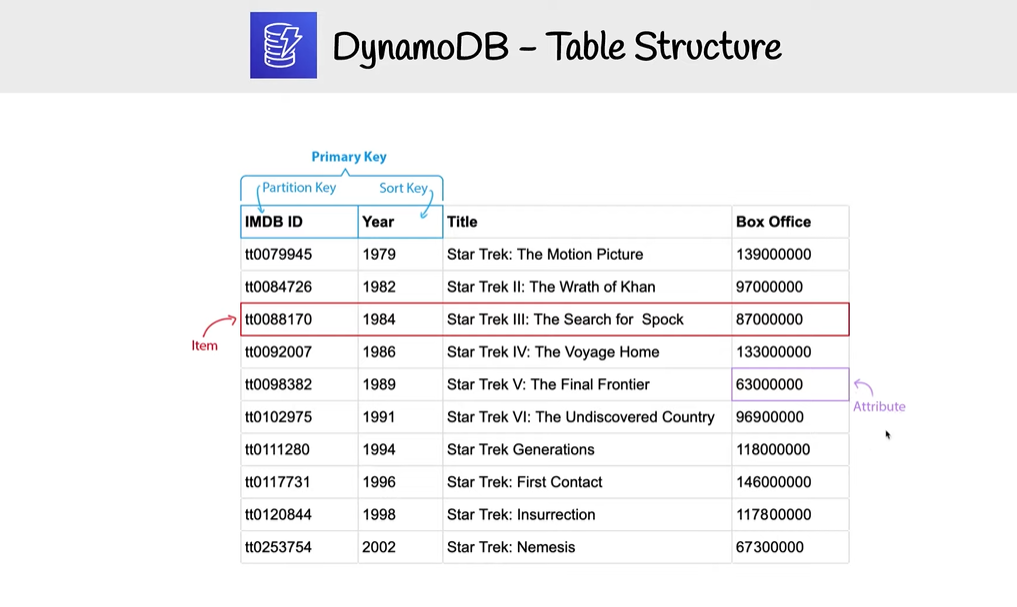
AWS DynamoDB

# 

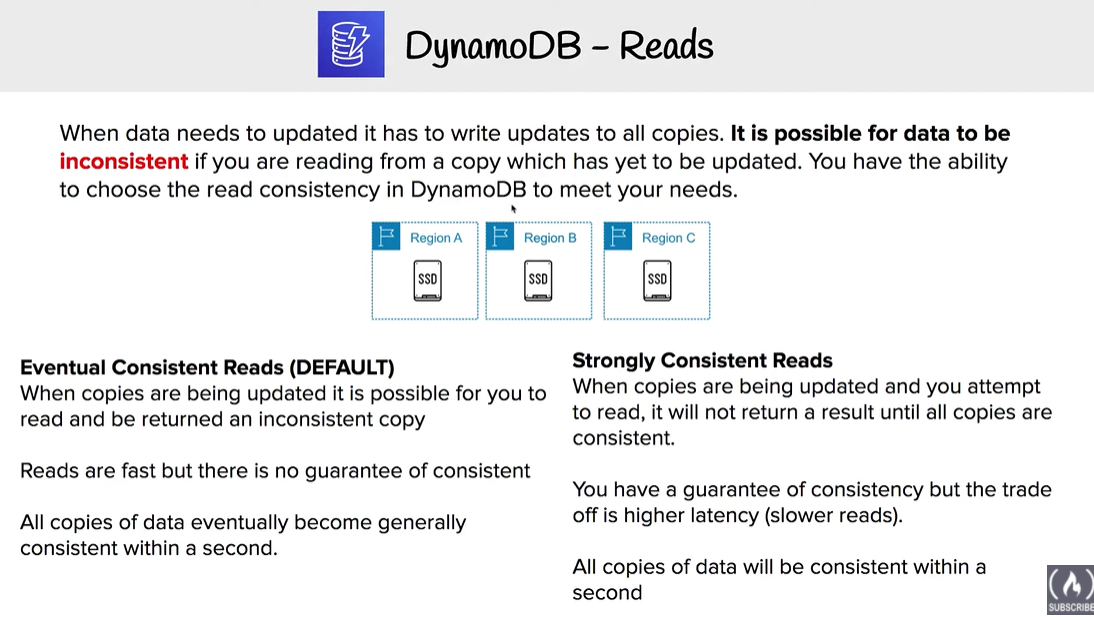




# AWS DynamoDB

* Amazon DynamoDB is a fully managed NoSQL database service that
  + makes it simple and cost-effective to store and retrieve any amount of data and serve any level of request traffic.
  + provides fast and predictable performance with seamless scalability
* DynamoDB enables customers to offload the administrative burdens of operating and scaling distributed databases to AWS, without having to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling.
* DynamoDB tables do not have fixed schemas, and table consists of items and each item may have a different number of attributes.
* DynamoDB synchronously replicates data across three facilities in an AWS Region, giving high availability and data durability.
* DynamoDB supports fast in-place updates. A numeric attribute can be incremented or decremented in a row using a single API call
* DynamoDB uses proven cryptographic methods to securely authenticate users and prevent unauthorized data access
* Durability, performance, reliability, and security are built in, with SSD (solid state drive) storage and automatic 3-way replication.
* DynamoDB supports two different kinds of primary keys:
  + **Partition Key**(previously called the **Hash key**)
    - A simple primary key, composed of one attribute
    - DynamoDB uses the partition key’s value as input to an internal hash function; the output from the hash function determine the partition where the item will be stored.
    - No two items in a table can have the same partition key value.
  + **Partition Key and Sort Key**(previously called the **Hash and Range key**)
    - A composite primary key composed of two attributes. The first attribute is the partition key, and the second attribute is the sort key.
    - DynamoDB uses the partition key value as input to an internal hash function; the output from the hash function determines the partition where the item will be stored.
    - All items with the same partition key are stored together, in sorted order by sort key value.
    - Combination of partition key and sort key must be unique
    - It is possible for two items to have the same partition key value, but those two items must have different sort key values.
* **DynamoDB Secondary indexes**
  + **add flexibility to the queries, without impacting performance**.
  + are automatically maintained as sparse objects, items will only appear in an index if they exist in the table on which the index is defined making queries against an index very efficient
* **DynamoDB throughput and single-digit millisecond latency makes it a great fit for gaming, ad tech, mobile, and many other applications**
* **ElastiCache can be used in front of DynamoDB in order to offload high amount of reads for non frequently changed data**

## DynamoDB Consistency



* Each DynamoDB table is automatically stored in the three geographically distributed locations for durability
* Read consistency represents the manner and timing in which the successful write or update of a data item is reflected in a subsequent read operation of that same item
* DynamoDB allows user to specify whether the read should be eventually consistent or strongly consistent at the time of the request
  + **Eventually Consistent Reads** (Default)
    - Eventual consistency option maximizes the read throughput.
    - Consistency across all copies is usually reached within a second
    - However, an eventually consistent read might not reflect the results of a recently completed write.
    - Repeating a read after a short time should return the updated data.
    - DynamoDB uses eventually consistent reads, by default.
  + **Strongly Consistent Reads**
    - Strongly consistent read returns a result that reflects all writes that received a successful response prior to the read
    - Strongly consistent reads are 2x the cost of Eventually consistent reads
    - Strongly Consistent Reads comes with disadvantages
      * A strongly consistent read might not be available if there is a network delay or outage. In this case, DynamoDB may return a server error (HTTP 500).
      * Strongly consistent reads may have higher latency than eventually consistent reads.
      * Strongly consistent reads are not supported on global secondary indexes.
      * Strongly consistent reads use more throughput capacity than eventually consistent reads.
* Read operations (such as GetItem, Query, and Scan) provide a ConsistentRead parameter, if set to true, DynamoDB uses strongly consistent reads during the operation.
* Query, GetItem, and BatchGetItem operations perform eventually consistent reads by default
  + Query and GetItem operations can be forced to be strongly consistent
  + Query operations cannot perform strongly consistent reads on Global Secondary Indexes
  + BatchGetItem operations can be forced to be strongly consistent on a per-table basis

## DynamoDB Throughput Capacity

Refer to [AWS DynamoDB Throughput Capacity](https://jayendrapatil.com/aws-dynamodb-throughput-capacity/) post

## DynamoDB Secondary Indexes

Refer to [AWS DynamoDB Secondary Indexes](https://jayendrapatil.com/aws-dynamodb-secondary-indexes/) post

## DynamoDB Advanced Topics

Refer to [DynamoDB Advanced](https://jayendrapatil.com/aws-dynamodb-advanced/) post, which covers

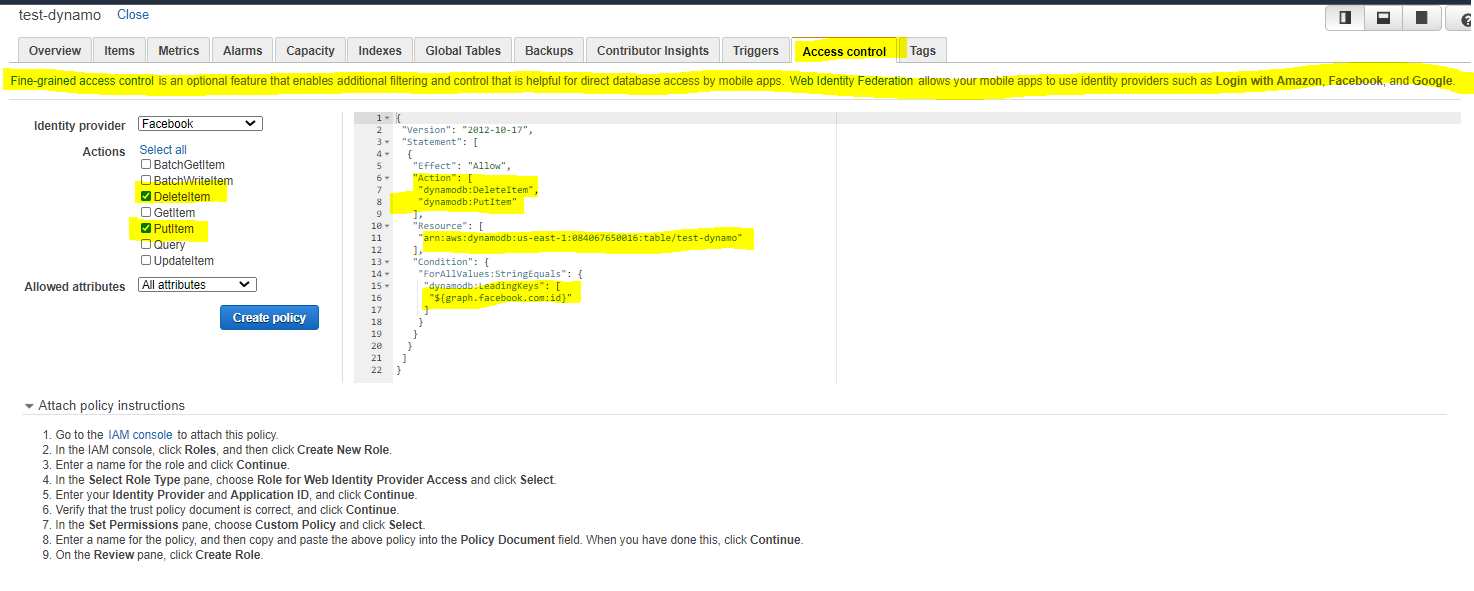
* DynamoDB Streams,
* Triggers,
* Cross Region Replication,
* [DAX](https://jayendrapatil.com/aws-dynamodb-accelerator-dax/),
* VPC Endpoints etc.

## DynamoDB Performance

* Automatically scales horizontally
* runs exclusively on Solid State Drives (SSDs).
  + SSDs help achieve the design goals of predictable low-latency response times for storing and accessing data at any scale.
  + SSDs High I/O performance enables it to serve high-scale request workloads cost efficiently, and to pass this efficiency along in low request pricing
* allows provisioned table reads and writes
  + Scale up throughput when needed
  + Scale down throughput four times per UTC calendar day
* automatically partitions, reallocates and re-partitions the data and provisions additional server capacity as the
  + table size grows or
  + provisioned throughput is increased
* Global Secondary indexes (GSI)
  + can be created upfront or added later

## DynamoDB Security

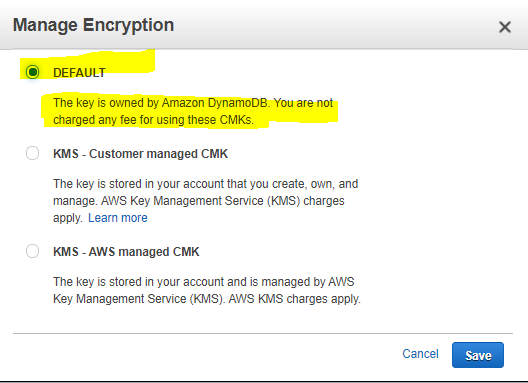
* Fine Grained Access Control (FGAC) gives a high degree of control over data in the table
* FGAC helps control *who* (caller) can access *which* items or attributes of the table and perform *what* actions (read/write capability).
* FGAC is integrated with IAM, which manages the security credentials and the associated permissions.



<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/specifying-conditions.html>

## DynamoDB Encryption

* Data in Transit Encryption
  + can be done by encrypting sensitive data on the client side or using encrypted connections (TLS)
  + All the data in DynamoDB is encrypted in transit (except the data in DAX)
  + communications to and from DynamoDB use the HTTPS protocol, which protects network traffic by using SSL/TLS encryption.
* DynamoDB supports Encryption at rest
  + Encryption at rest is enabled on all DynamoDB table data and cannot be disabled
  + Encryption at rest enables encryption for the data persisted (data at rest) in the DynamoDB tables.
  + Encryption at rest includes the base tables, primary key, local and global secondary indexes, streams, global tables, backups, and DynamoDB Accelerator (DAX) clusters
  + Encryption at rest automatically integrates with AWS KMS for managing the keys used for encrypting the tables.
  + Encryption at rest also supports AWS owned CMK, AWS managed CMK, and Customer managed CMK
  + Encryption at rest can be enabled only for a new table and not for an existing table
  + Encryption once enabled for a table, cannot be disabled
  + DynamoDB streams can be used with encrypted tables and are always encrypted with a table-level encryption key
  + On-Demand Backups of encrypted DynamoDB tables are encrypted using S3’s Server-Side Encryption
  + Encryption at rest encrypts the data using 256-bit AES encryption.



## DynamoDB Encryption Client

* DynamoDB Encryption Client is a software library that helps protect the table data before sending it to DynamoDB.
* Encrypting the sensitive data in transit and at rest helps ensure that the plaintext data isn’t available to any third party, including AWS.
* Encryption Client encrypts attribute values which can be controlled, but does not encrypt entire table, attribute names or primary key

## DynamoDB Costs

* Index Storage
  + DynamoDB is an indexed data store
    - Billable Data = Raw byte data size + 100 byte per-item storage indexing overhead
* Provisioned throughput
  + Pay flat, hourly rate based on the capacity reserved as the throughput provisioned for the table
  + one Write Capacity Unit provides one write per second for items < 1KB in size.
  + one Read Capacity Unit provides one strongly consistent read (or two eventually consistent reads) per second for items < 4KB in size.
  + Provisioned throughput charges for every 10 units of Write Capacity and every 50 units of Read Capacity.
* Reserved capacity
  + Significant savings over the normal price
  + Pay a one-time upfront fee
* DynamoDB also charges for storage, backup, replication, streams, caching, data transfer out.

Graphical user interface, application

Description automatically generated

