AWS DynamoDB Best Practices

## Primary Key Design

* Primary key uniquely identifies each item in a DynamoDB table and can be simple (a partition key only) or composite (a partition key combined with a sort key).
* Partition key portion of a table’s primary key determines the logical partitions in which a table’s data is stored, which in turn affects the underlying physical partitions.
* **Avoid hot keys and hot partitions** – a partition key design that doesn’t distribute I/O requests evenly can create “hot” partitions that result in throttling and use the provisioned I/O capacity inefficiently.
* Partition key should have many unique values
* Distribute reads / writes uniformly across partitions to avoid hot partitions
* Store hot and cold data in separate tables
* Consider all possible query patterns to eliminate use of scans and filters
* Choose sort key depending on the application’s needs

## Secondary Indexes

* Use indexes based on when the application’s query patterns
* LSIs
  + Use primary key or LSIs when strong consistency is desired
  + Watch for expanding item collections (10 GB size limit!)
* GSIs
  + Use GSIs for finer control over throughput or when your application needs to query using a different partition key
  + Can be used for eventually consistent read replicas – set up a global secondary index that has the same key schema as the parent table, with some or all of the non-key attributes projected into it.
* **Project fewer attributes** – As secondary indexes consume storage and provisioned throughput, keep the size of the index as small as possible. as it would provide greater performance.
* **Keep the number of indexes to a minimum –**don’t create secondary indexes on attributes that aren’t queried often. Indexes that are seldom used contribute to increased storage and I/O costs without improving application performance.

## Large Items and Attributes

* DynamoDB currently limits the size of each item that is stored in a table
* Use shorter (yet intuitive!) attribute names
* Keep item size small
* Use compression (GZIP)
* Split large attributes across multiple items
* Store metadata in DynamoDB and large BLOBs or attributes in S3

# Querying and Scanning Data

* Avoid scans and filters – Scan operations are less efficient than other operations in DynamoDB. A Scan operation always scans the entire table or secondary index. It then filters out values to provide the result, essentially adding the extra step of removing data from the result set.
* Use eventual consistency for reads

# Time Series Data

* Use table per day, week, month etc for storing time series data – create one table per period, provisioned with the required read and write capacity and the required indexes.
* Before the end of each period, prebuild the table for the next period. Just as the current period ends, direct event traffic to the new table. Assign names to the tables that specify the periods they have recorded.
* As soon as a table is no longer being written to, reduce its provisioned write capacity to a lower value (for example, 1 WCU), and provision whatever read capacity is appropriate. Reduce the provisioned read capacity of earlier tables as they age. Archive or delete the tables whose contents are rarely or never needed.