**AWS DynamoDB Global And Local Secondary Indexes Comparison**



Learn about AWS DynamoDB(DDB) indexes and the difference between its global and local secondary indexes.

**Why Secondary Indexes**

AWS DynamoDB being a No SQL database doesn’t support queries such as SELECT with a condition such as the following query.

SELECT \* FROM Users  
WHERE email='username@email.com';

Note that when doing the query above with an SQL database, a query optimizer evaluates available indexes to see if any index can fulfill the query.

It is possible to obtain the same query result using DynamoDB scan operation. However, scan operations access every item in a table which is slower than query operations that access items at specific indices. Imagine, you have to look for a book in a library by going through possibly all the books in the library versus you know which shelf the book is at.

Thus, there is a need for another table or data structure that stores data with different primary key and maps a subset of attributes from this base table. This other table is called a secondary index and is managed by AWS DynamoDB. When items are added, modified, or deleted in the base table, associated secondary indexes will be updated to reflect the changes.

**Global(GSI) vs Local Secondary Indexes(LSI)**

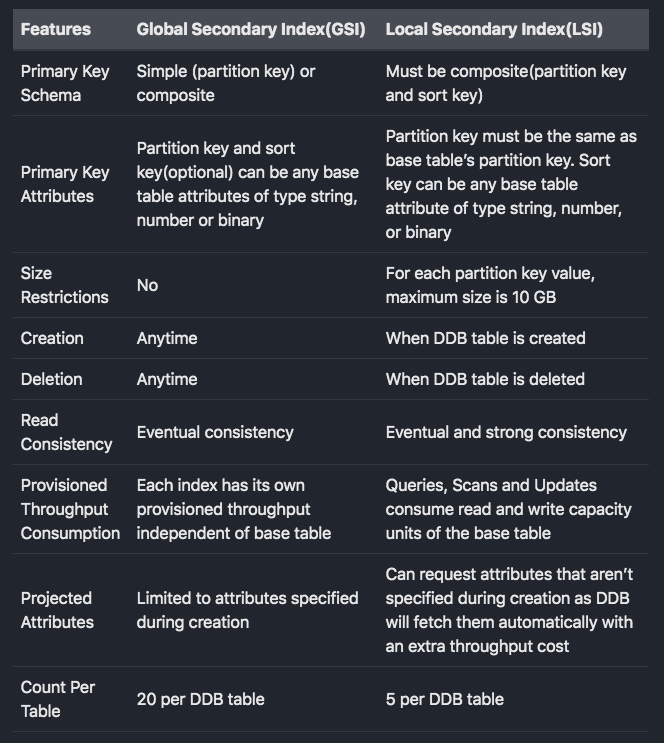
AWS DynamoDB supports two types of indexes: Global Secondary Index (GSI) and Local Secondary Index (LSI).

Global secondary index is an index that have a partition key and an optional sort key that are different from base table's primary key. It is deemed "global" because queries on the index can access the data across different partitions of the base table. It can viewed as a different table that contains attributes based on the base table.

Local secondary index is an index that must have the same partition key but a different sort key from the base table. It is considered "local" because every partition of a local secondary index is bounded by the same partition key value of the base table. It enables data query with different sorting order of the specified sort key attribute.

Local secondary index allows Query operation to retrieve several items that have the same partition key value but different sort key values AND one item with a specific partition key value and a sort key value.

**Important Difference between GSI and LSI**



**Note**

1. Since Global Secondary Indexes have their own throughput consumption, to minimize cost, I suggest project only attributes that are needed. You can always create a new index that projects more attributes and replace the existing one when your use case changes.
2. Global Secondary Indexes are sparse indexes as only specified attributes of the items in the base table appear in the index.

**Secondary Index Examples**

Check out the following GSI and LSI examples to get an idea of when to use which.

**GSI Example**

Consider this table that contains Uuid as primary key, UserId and Data attributes.

| Uuid(Partition Key) | UserId | Data |

With this base table key schema, it can answer queries to retrieve data for a uuid. However, to get all data for a user id, it would have to do a scan query and get all the items that have matching user id.

To be able to get all data for a user efficiently, you can use a global secondary index that has UserId as its primary key (partition key). Using this index, you can do a query to retrieve all data for a user.

**LSI Example**

Local Secondary Index enables different sorting order of the same list of items as LSI uses the same partition key as base table but different sort key. Consider this table that uses composite keys: UserId as partition key, ArticleName as sort key and other attributes: DateCreated and Data.

|UserId(Partition Key) | ArticleName(Sort Key) | DateCreated | Data|

With this base table key schema, it can answer queries to retrieve all the article sorted by names for a specific user(query by UserId). However, to retrieve all the articles associated with a user sorted by date created, you would have to retrieve all the articles first and sort them.

With a local secondary index that has UserId as its partition key and DateCreated as its sort key, you can retrieve a user’s articles sorted by date created.

|UserId(Partition Key) | DateCreated(Sort Key) | ArticleName | Data|

**Summary — Which One Should You Use?**

In short, use DynamoDB Global Secondary Index when you need to support querying non-primary key attribute of a table.

And, use DynamodB Local Secondary index when you need to support querying items with different sorting order of attributes.