**DynamoDB Partitions and Data Distribution**

DynamoDB stores data in partitions. A *partition* is an allocation of storage for a table, backed by solid-state drives (SSDs) and automatically replicated across multiple Availability Zones within an AWS Region. Partition management is handled entirely by DynamoDB—you never have to manage partitions yourself.

When you create a table, the initial status of the table is CREATING. During this phase, DynamoDB allocates sufficient partitions to the table so that it can handle your provisioned throughput requirements. You can begin writing and reading table data after the table status changes to ACTIVE.

DynamoDB allocates additional partitions to a table in the following situations:

* If you increase the table's provisioned throughput settings beyond what the existing partitions can support.
* If an existing partition fills to capacity and more storage space is required.

Partition management occurs automatically in the background and is transparent to your applications. Your table remains available throughout and fully supports your provisioned throughput requirements.

For more details, see [Partition Key Design](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-partition-key-design.html).

Global secondary indexes in DynamoDB are also composed of partitions. The data in a GSI is stored separately from the data in its base table, but index partitions behave in much the same way as table partitions.

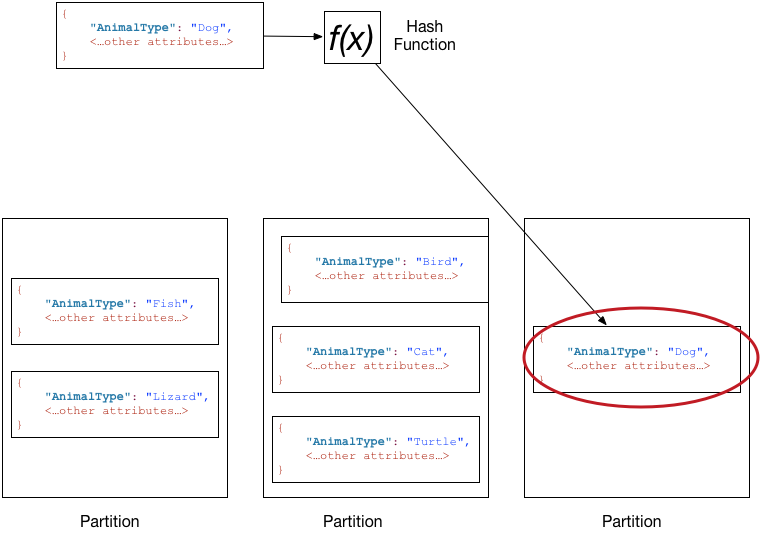
**Data Distribution: Partition Key**

If your table has a simple primary key (partition key only), DynamoDB stores and retrieves each item based on its partition key value.

To write an item to the table, DynamoDB uses the value of the partition key as input to an internal hash function. The output value from the hash function determines the partition in which the item will be stored.

To read an item from the table, you must specify the partition key value for the item. DynamoDB uses this value as input to its hash function, yielding the partition in which the item can be found.

The following diagram shows a table named *Pets*, which spans multiple partitions. The table's primary key is *AnimalType* (only this key attribute is shown). DynamoDB uses its hash function to determine where to store a new item, in this case based on the hash value of the string *Dog*. Note that the items are not stored in sorted order. Each item's location is determined by the hash value of its partition key.



**Note**

DynamoDB is optimized for uniform distribution of items across a table's partitions, no matter how many partitions there may be. We recommend that you choose a partition key that can have a large number of distinct values relative to the number of items in the table.

**Data Distribution: Partition Key and Sort Key**

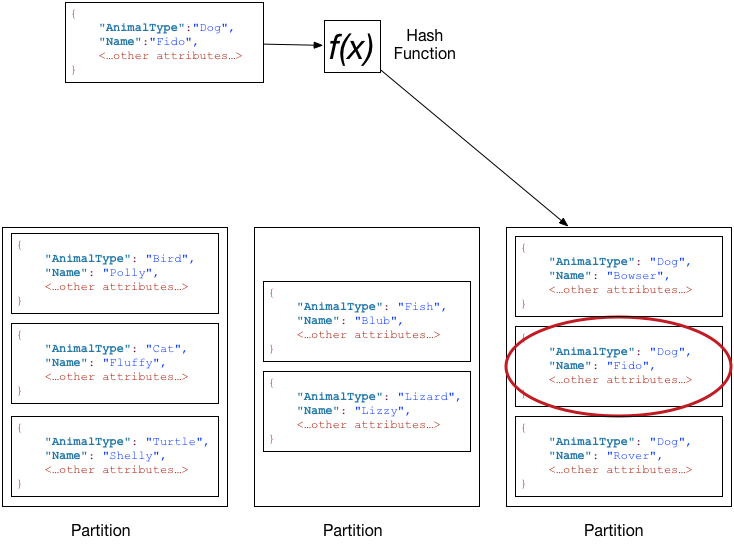
If the table has a composite primary key (partition key and sort key), DynamoDB calculates the hash value of the partition key in the same way as described in [Data Distribution: Partition Key](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.Partitions.html#HowItWorks.Partitions.SimpleKey)—but it stores all of the items with the same partition key value physically close together, ordered by sort key value.

To write an item to the table, DynamoDB calculates the hash value of the partition key to determine which partition should contain the item. In that partition, there could be several items with the same partition key value, so DynamoDB stores the item among the others with the same partition key, in ascending order by sort key.

To read an item from the table, you must specify its partition key value and sort key value. DynamoDB calculates the partition key's hash value, yielding the partition in which the item can be found.

You can read multiple items from the table in a single operation (Query), provided that the items you want have the same partition key value. DynamoDB returns all of the items with that partition key value. Optionally, you can apply a condition to the sort key so that it returns only the items within a certain range of values.

Suppose that the *Pets* table has a composite primary key consisting of *AnimalType* (partition key) and *Name*(sort key). The following diagram shows DynamoDB writing an item with a partition key value of *Dog* and a sort key value of *Fido*.



To read that same item from the *Pets* table, DynamoDB calculates the hash value of *Dog*, yielding the partition in which these items are stored. DynamoDB then scans the sort key attribute values until it finds*Fido*.

To read all of the items with an *AnimalType* of *Dog*, you can issue a Query operation without specifying a sort key condition. By default, the items are returned in the order that they are stored (that is, in ascending order by sort key). Optionally, you can request descending order instead.

To query only some of the *Dog* items, you can apply a condition to the sort key (for example, only the *Dog*items where *Name* begins with a letter that is within the range A through K).

**Note**

In a DynamoDB table, there is no upper limit on the number of distinct sort key values per partition key value. If you needed to store many billions of *Dog* items in the *Pets* table, DynamoDB automatically allocates enough storage to handle this requirement.