DynamoDB

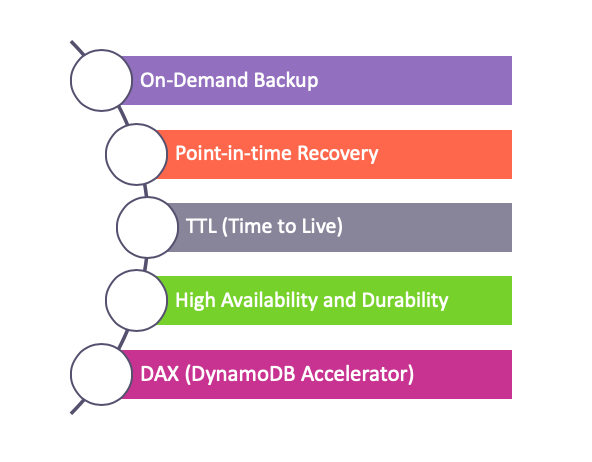
There are two types of databases, **Relational** and **Non-Relational**. As you remember from the previous lessons, we just learned the Relational database (SQL) service, RDS. Now, It's time to cover **DynamoDB** which is the **Non-Relational (NoSQL)** database solution of AWS.

Amazon DynamoDB is a **NoSQL database** service that supports **key-value** and **document** data models. It is NoSQL database, so you don't need to stick pre-determined schema. Instead of Schema, DynamoDB uses **flexible tables**. You can create database tables that can store and retrieve any amount of data and serve any level of request traffic.

Unlike RDS, Amazon DynamoDB is a **fully-managed** database. It means, there are no servers to provision, patch, or manage, and no software to install, maintain or operate. DynamoDB automatically scales tables to adjust for capacity and maintains performance with **zero administration**.

DynamoDB is designed to run high-performance, internet-scale applications that would overburden traditional relational databases like mobile, web, gaming, ad-tech, IOT, and many other applications.

### **Features of DynamoDB**



**On-Demand Backup:**

DynamoDB provides on-demand backup capability. It allows you to create full backups of your tables for long-term retention and archival for regulatory compliance needs.

**Point-in-time Recovery:**

After creating on-demand backups, you can enable point-in-time recovery for your Amazon DynamoDB tables. Point-in-time recovery helps protect your tables from accidental write or delete operations. With point-in-time recovery, you can restore that table to any point in time during the last 35 days.

**TTL(Time to Live)**

DynamoDB has also **TTL** function. DynamoDB TTL allows you to delete expired items from tables automatically so that you can reduce storage usage and the cost of storing data that is no longer relevant.

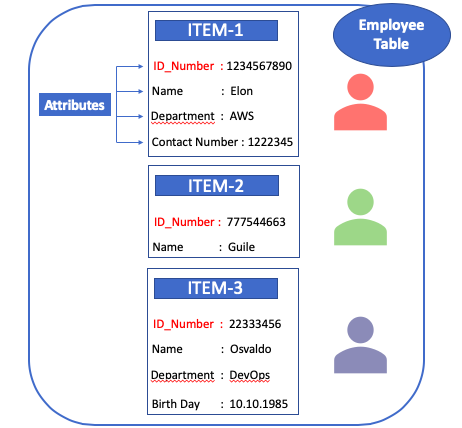
**High Availability and Durability**

In DynamoDB all of your data is stored on solid-state disks (SSDs) and is automatically replicated across **multiple Availability Zones** in an AWS Region. but also you can use **Global Tables** to keep DynamoDB tables in **multiple AWS Regions**.

**DAX (DynamoDB Accelerator)**

Although Amazon DynamoDB is designed for performance, you may need to require response times in **microseconds** for certain use cases. So, DynamoDB Accelerator (DAX) delivers fast response times for these use cases. In short, you can speed up your transactions with this feature.

### **Tables, Items and Attributes**



In DynamoDB, **tables**, **items**, and **attributes** are the key components that you work with. A table is a collection of items, and each item is a collection of attributes.

* **Tables:**

Similar to other database systems, DynamoDB stores data in tables. A table is a collection of data. As you see in the picture above, there is a table called Employee and you can see the contact information of Employee in this table.

* **Items:**

Each table contains items. An item is a group of attributes that is uniquely identifiable among all of the other items. In an Employee Table, **each item represents a person**. Items in DynamoDB are similar in many ways to **rows and records** in other database systems. In DynamoDB, there is **no limit** to the number of items you can store in a table. But each item can not exceed **400 KB**.

* **Attributes:** Each item is composed of one or more attributes. An attribute is a **key data element** for DynamoDB table. For example, an item in an Employee Table contains attributes called **PersonID, LastName, FirstName, and so on**.

### **Primary Key:**

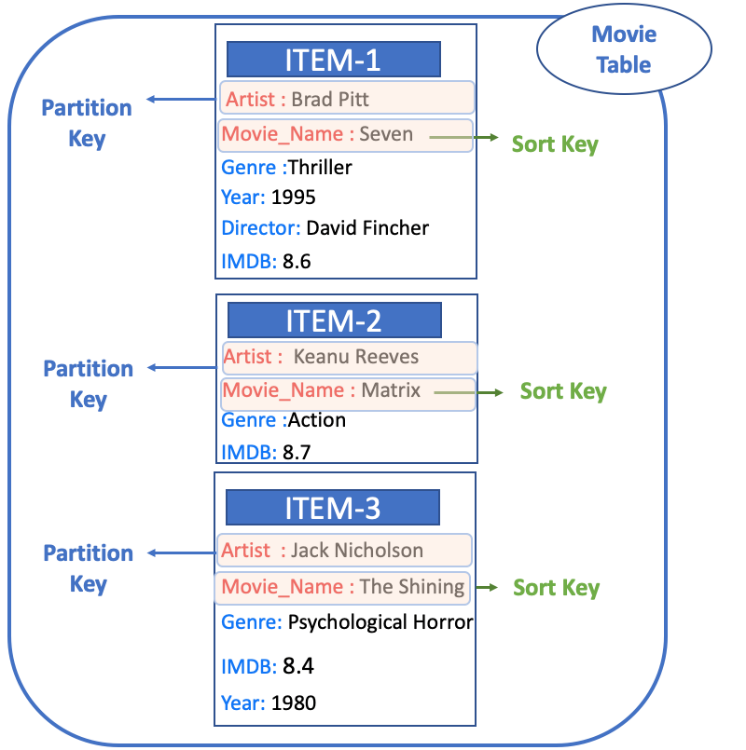
DynamoDB uses **Primary Keys** to **uniquely identify each item** in a table. When you create a table, in addition to the table name, you must specify the primary key of the table. The primary key uniquely identifies each item in the table, so that no two items can have the same key.

There are two different kinds of Primary Keys: **Partition Key** and **Partition Key&Sort Key**.

* **Partition Key:**

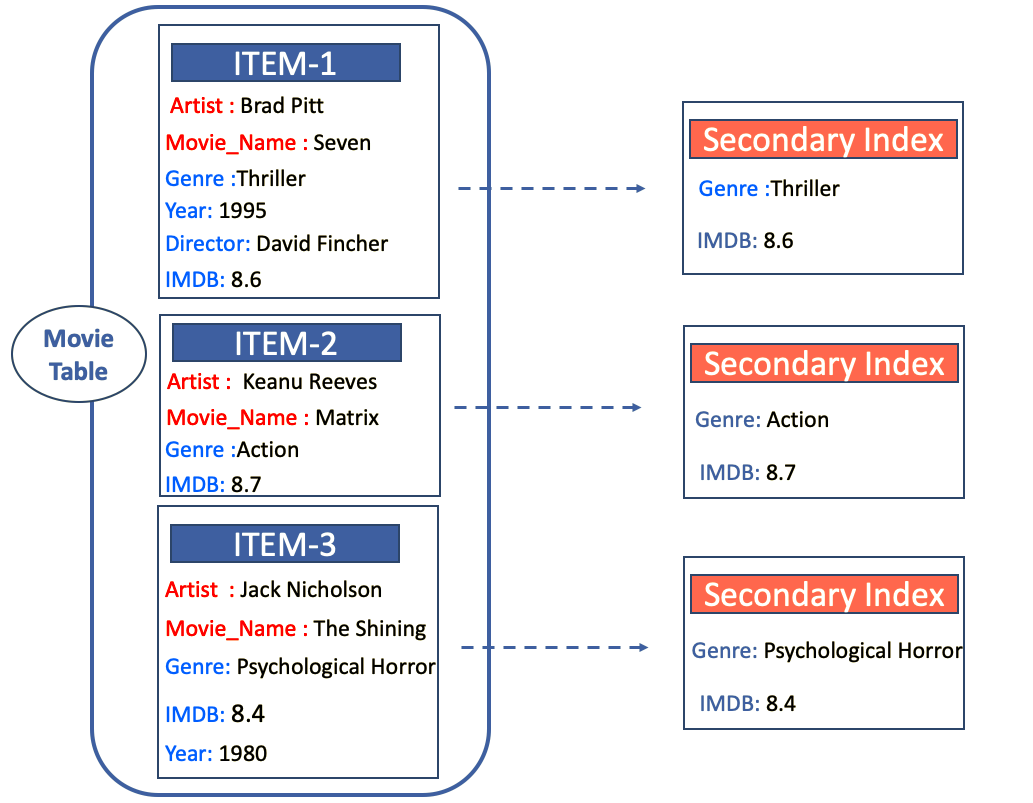
Partition Key is a simple primary key that composed of **one attribute**. If you determine one of the attributes as a Primary Key (Partition Key), there is no way to add an item into the table without Partition Key. Partition Key is **mandatory** value in the table.

As you see in the picture above, there is an **Employee Table** described in tables, items, and attributes. In this table, the primary key(Partition Key) is PersonID. All items (person) have PersonID value. You can access any item in the Employee Table directly by providing the PersonID value for that item.



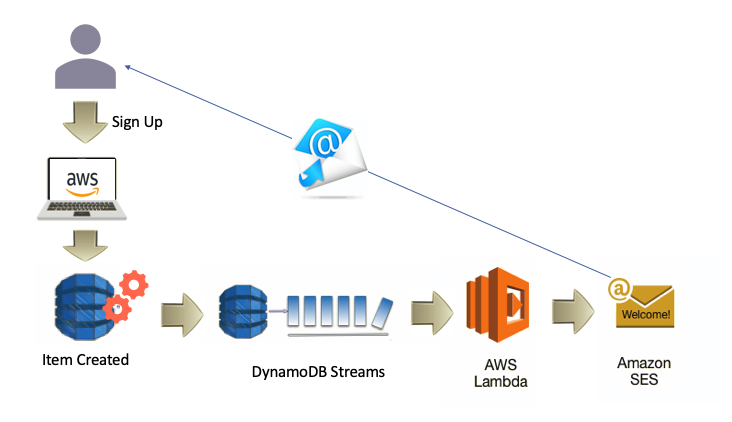
* **Partition Key and Sort Key**
  + Unlike the Primary Key, this type of key is composed of **two attributes**. The first attribute is the **partition key**, and the second attribute is the **Sort Key**.
  + In fact, **Sort Key** is an optional key. We use the Sort Key to enable rich query capabilities. If there are 2 items that have the same Partition Key, thanks to the Sort Key, you can distinguish these two items from each other.
  + As you see in the picture above, there is a Movie Table described in tables, items, and attributes. The primary key for Movie consists of two attributes (Artist and Movie\_Name). Each item in the table must have these two attributes. The combination of Artist and Movie\_Name distinguishes each item in the table from all of the others.

### **Secondary Indexes**



* DynamoDB uses **Secondary Indexes** to **enrich queries**.
* A secondary index lets you query the data in the table using an alternate key, in addition to queries against the primary key. DynamoDB doesn't require that you use indexes, but they give your applications more flexibility when querying your data.
* In other words, Secondary Indexes are a way to create an **alternative query option** with current attributes of the item.
* In the example Movie Table shown above, you can query data items by Artist (partition key) or by Artist and Movie\_Name (partition key and sort key). But, what if you also wanted to query the data by **Genre and IMDB Rate**? To do this, you can create a **Secondary Index** on Genre and IMDB, and then you can read data from the index in much the same way as you do from the Movie table.
* DynamoDB supports two kinds of 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.
* Each table in DynamoDB has a limit of **20 global secondary indexes** (default limit) and **5 local secondary indexes** per table.

### **DynamoDB Streams**



DynamoDB Streams is an optional feature that **captures data modification events** in DynamoDB tables. It is a kind of transaction record.

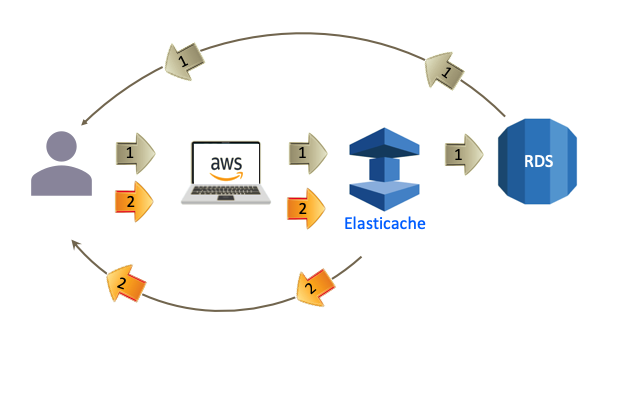
Each event is represented by a stream record. If you enable a stream on a table, DynamoDB writes a stream record whenever events such as **adding, updating or deleting an item** occurs in the table.

Stream records have a lifetime of **24 hours**; after that, they are automatically removed from the stream.

But, why do we need streams?

* **First**, you can use DynamoDB Streams **together with AWS Lambda to create a trigger**. For example, in the picture above:
  + Thanks to the stream whenever a new customer is added to the customer table, DynamoDB Streams creates a Stream Record,
  + Since we associate the stream with a Lambda function, the Lambda function will trigger Amazon Simple Email Service (Amazon SES)
  + Then Amazon Simple Email Service (Amazon SES) will send a welcome email to the customer.
* **Another reason** for using DynamoDB Streams is to **enable the Global Tables** which means Multi-Region Replication with DynamoDB

Elasticache



Before covering Elasticache, we need to understand the concept of **In-Memory Cache**. In-Memory Cache is a **temporary and fast** storage component. These components are used to reduce the workload of the main data storage device such as a database.

As you know from the conventional computer, RAM responds faster than Hard Disk to the queries. As for database infrastructure, **In-Memory Cache** responds faster than a database system that stores the data in SSD or HDD disks. So we prefer to store the **necessary** and **frequently used information** in In-Memory Cache during the time we determine beforehand.

As you see in the picture above,

In the first query of the client, **Database responds** the query, and by the way, In-Memory Cache memorizes this information.

In the second query, **In-Memory Cache responds** to the query. Thanks to the In-Memory Cache, we achieve to decrease the workload of the database and provide a faster reaction.

### **AWS Elasticache:**

The product of AWS in in-memory cache service is Amazon Elasticache. Amazon ElastiCache makes it easy to set up, manage, and scale a distributed in-memory data store or cache environment in the cloud. It provides a high-performance, scalable, and cost-effective caching solution.

At the same time, it helps remove the complexity associated with deploying and managing a distributed cache environment.

### **Features of AWS Elasticache:**

* **Redis and Memcached:**

Amazon Elasticache offers two kinds of in-memory cache options, Redis and Memcached. These are popular, open-source, in-memory data stores. Both of them have satisfactory performance for different use cases.

* **Extreme performance:**

Amazon ElastiCache works as an in-memory data store and cache to support the most demanding applications requiring **sub-millisecond response times**. Amazon Elasticache provides secure, blazing-fast performance.

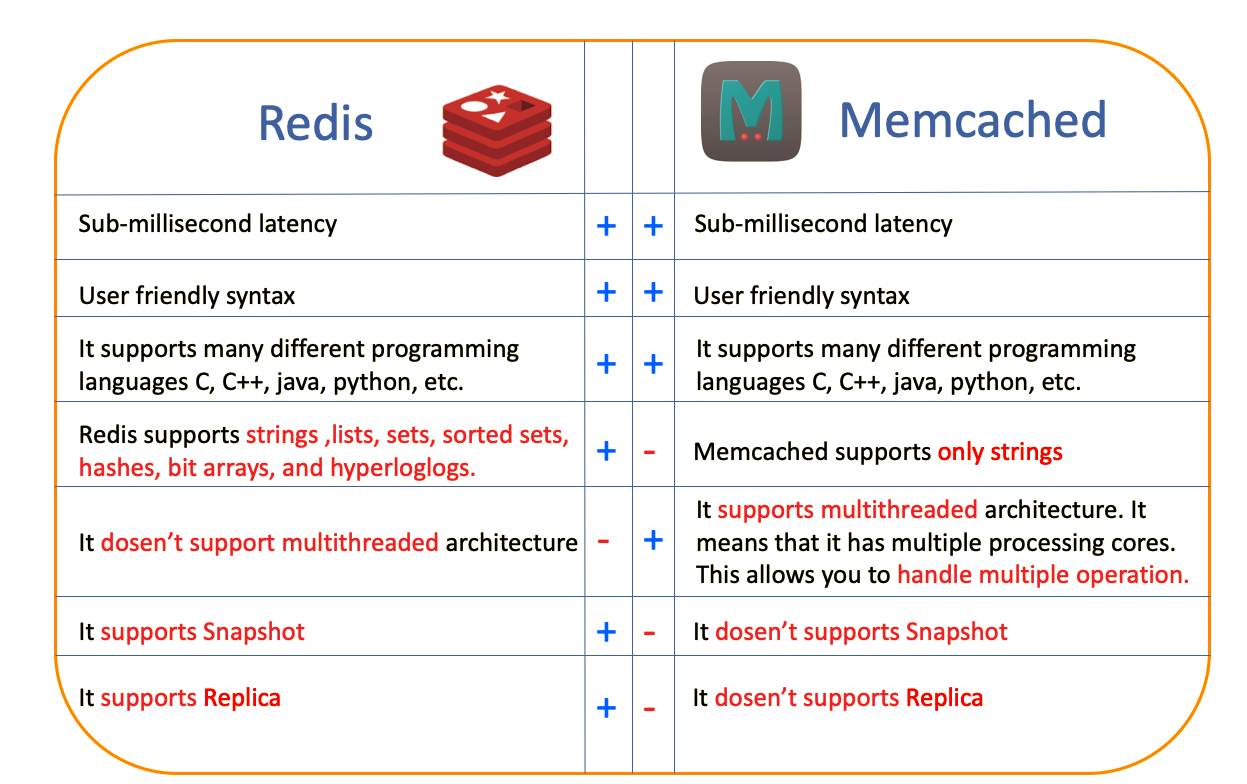
* **Fully Managed:**

Thanks to the AWS Elasticache, you don't need to perform management tasks such as hardware provisioning, software patching, setup, configuration, monitoring, failure recovery, and backups. Elasticache constantly tracks the clusters to maintain the workloads up and working so that you can concentrate on designing higher-value apps.

* **Scalability:**

Amazon ElastiCache provides to scale capacity to adapt fluctuating application requirements. In addition to write and memory scaling, Elasticache provides a read scaling via Replicas.

### **Redis vs. Memcached**



Redis and Memcached are common, open-source, in-memory stores of data. While they are both simple to use and deliver good efficiency, there are significant distinctions to remember when choosing an engine.

**Memcached** is built for **simplicity**, while **Redis** provides a rich set of features that render it useful for a **wide variety of applications**. Some differences and features are seen in the picture above.