Amazon Web Services

AWS Databases

RDS & DynamoDB

Contents

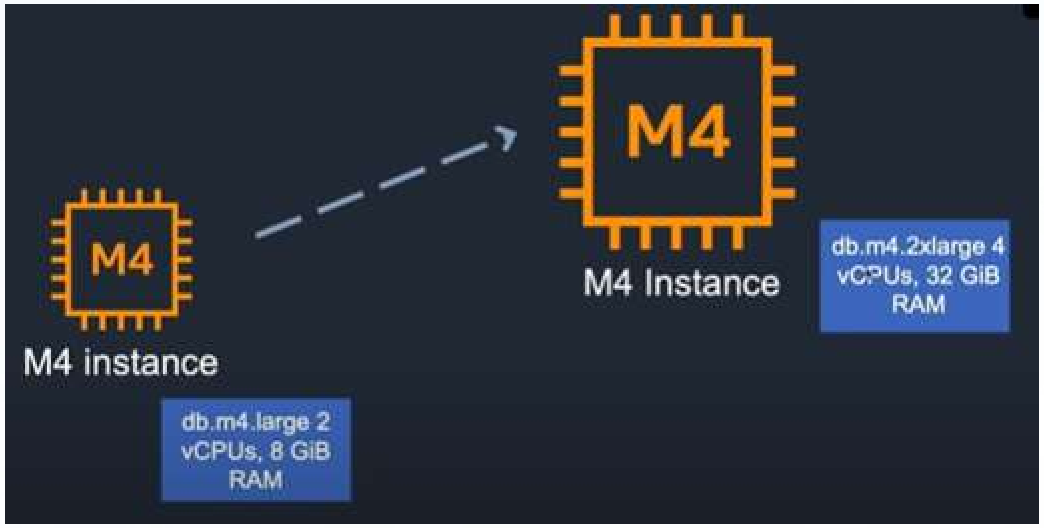
Amazon RDS

Amazon Relational Database Service (RDS)

- Amazon RDS is a managed relational database service that makes it easy to set up, operate, and scale a relational database in the cloud.
- It provides cost-efficient and resizable capacity while automating time-consuming administration tasks such as hardware provisioning, database setup, patching and backups.
- Amazon RDS is available on several database instance types - optimized for memory, performance or I/O
- Amazon RDS provides you with six familiar database engines to choose from - Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, and SQL Server.

RDS Scaling – Scale Up

- You can the change the instance type of an RDS database to scale up resources. This will incur database downtime.


M4
M4 instance
db.m4.large 2 vCPUs, 8 GiB RAM
M4
M4 Instance
db.m4.2xlarge 4 vCPUs, 32 GiB RAM

RDS Scaling - Scale Out & Disaster Recovery (DR)

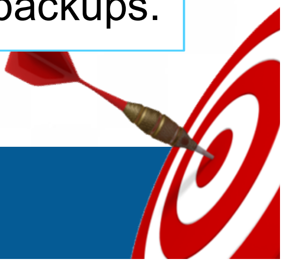
- You can scale out your RDS by creating RDS read replicas in the same AZ that are replicated asynchronously.

- These read replicas allows us to offload some of the read requests from the master there by reducing the load. Read requests can be directed to the read replicas while the write operations are performed on the master.

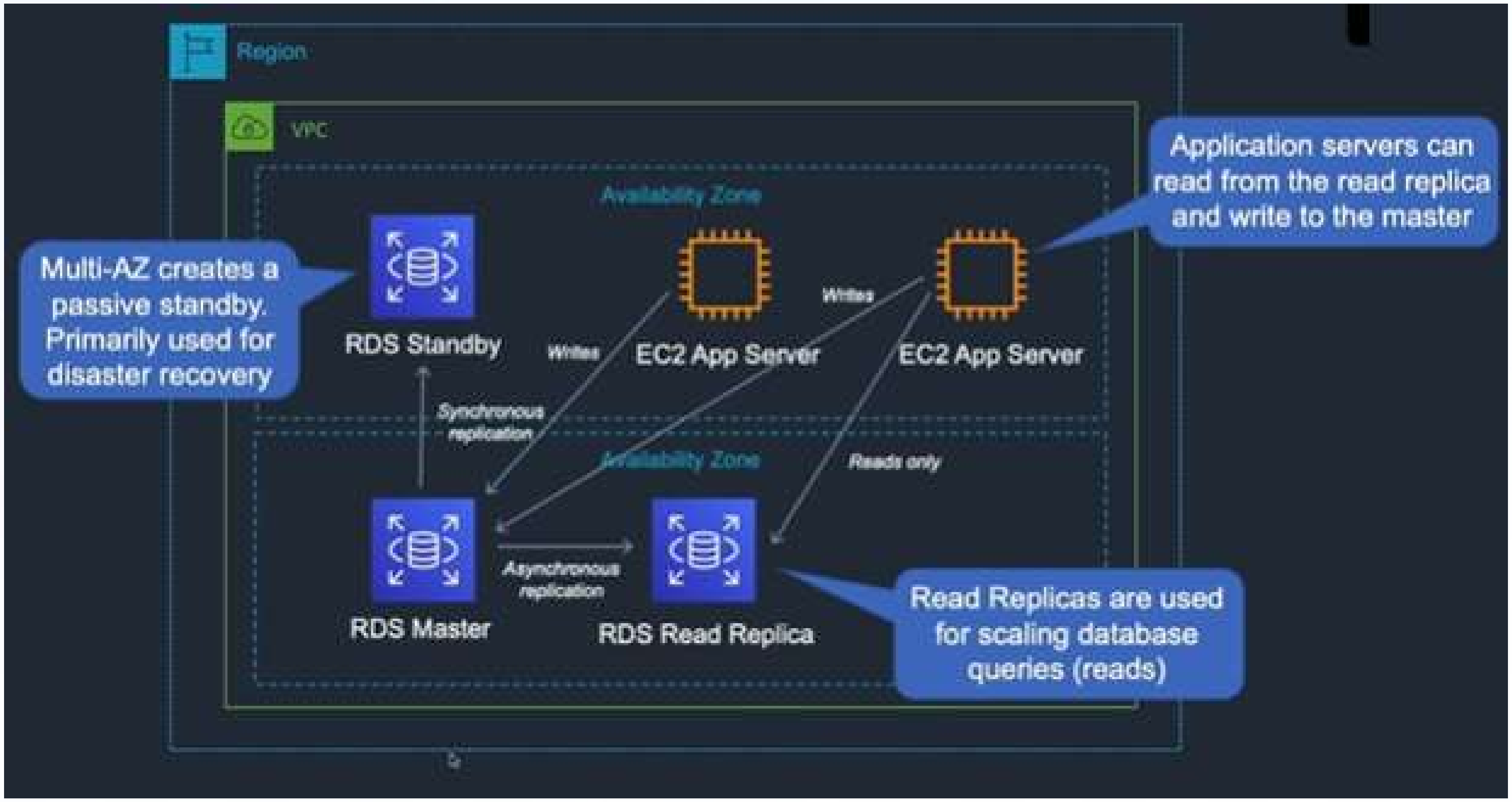
- We can create RDS Standby database on another AZ (**Multi-AZ**) which is synchronously replicated. These Standby replicas can be used for disaster recovery in case master replica fails.

- Multi-AZ Standby Replica ➡ Disaster recovery.
- Intra-AZ Read Replica ➡ To scale out read requests.

Multi-AZ creates a standby in a different Availability Zone (AZ) to provide data redundancy, eliminate I/O freezes, and minimize latency spikes during system backups.



RDS Scaling – Scale Out & Disaster Recovery (DR)


Region
VPC
Availability Zone
Availability Zone
Multi-AZ creates a passive standby. Primarily used for disaster recovery
RDS Standby
Writes
EC2 App Server
Writes
EC2 App Server
Application servers can read from the read replica and write to the master
Synchronous replication
Reads only
RDS Master
Asynchronous replication
RDS Read Replica
Read Replicas are used for scaling database queries (reads)

Create an RDS database

- Go to Services >> Database >> RDS and click on **Create Database** button.

- Create Database Options:
  - Database Creation Method – Select Standard create
  - Engine Options – Select MySQL (& select the default edition)
  - Templates – Select Free tier
  - Settings:
    - DB instance identifier – type database name (ex: ctsdemodb)
    - Credentials Settings
      - Master username (ex: ctsdemo); Master password (ex: ctsdemo123)
  - DB instance class – select default (burstable – db.t2.micro)
  - Storage
    - Storage Type, Allocated Storage, Storage autoscaling
  - Availability & Durability
  - Connectivity
    - Public access: Yes
  - Database Authentication (select password authentication)
  - Additional Configurations
    - Initial Database (ctsdemodb1)

- Click on **Create Database** button

Creating a Multi-AZ Standby

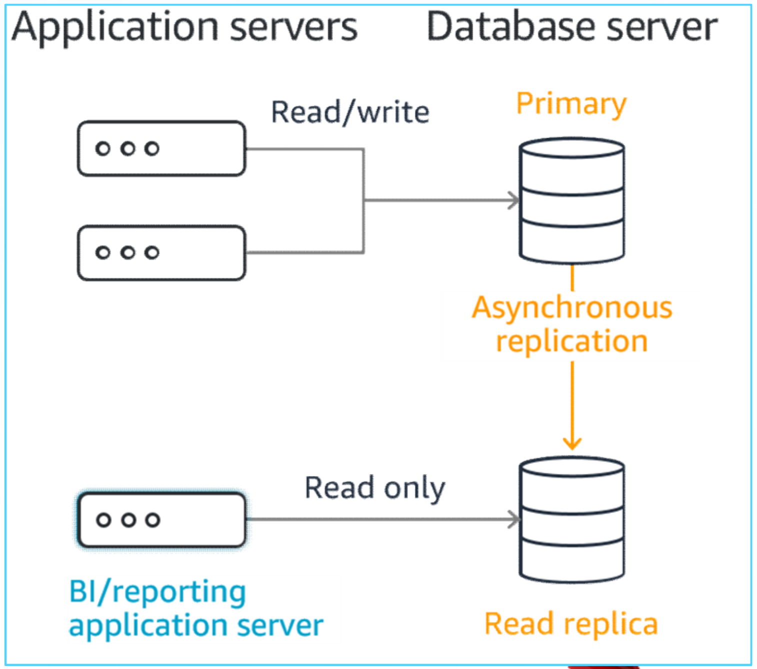
A Multi-AZ standby of an RDS database is a synchronous replica that acts as a failover for disaster recovery. The standby will have the same connection endpoint and it will automatically reroute the connection to standby in case of master failure.

- Open your DFS database instance
- Click on **Modify** option
- Under **Availability & durability,** select **Multi-AZ deployment**
- Under **Scheduling of modifications,** select **Apply immediately**
- Click on **Modify DB instance** option

Creating a Read-Replica

A read-replica of an RDS database is a separate database instance with a different connection endpoint. Applications can connect to this read-replica for read options there by offloading some traffic from master. We can setup a Multi-AZ standby for read replicas also, if required.

- Open your DFS database instance.
- Click on **Actions** and select **Read replica** option.
- Note that this is a separate database instance and you have to fill all details as you would for a new RDS database.



Amazon DynamoDB

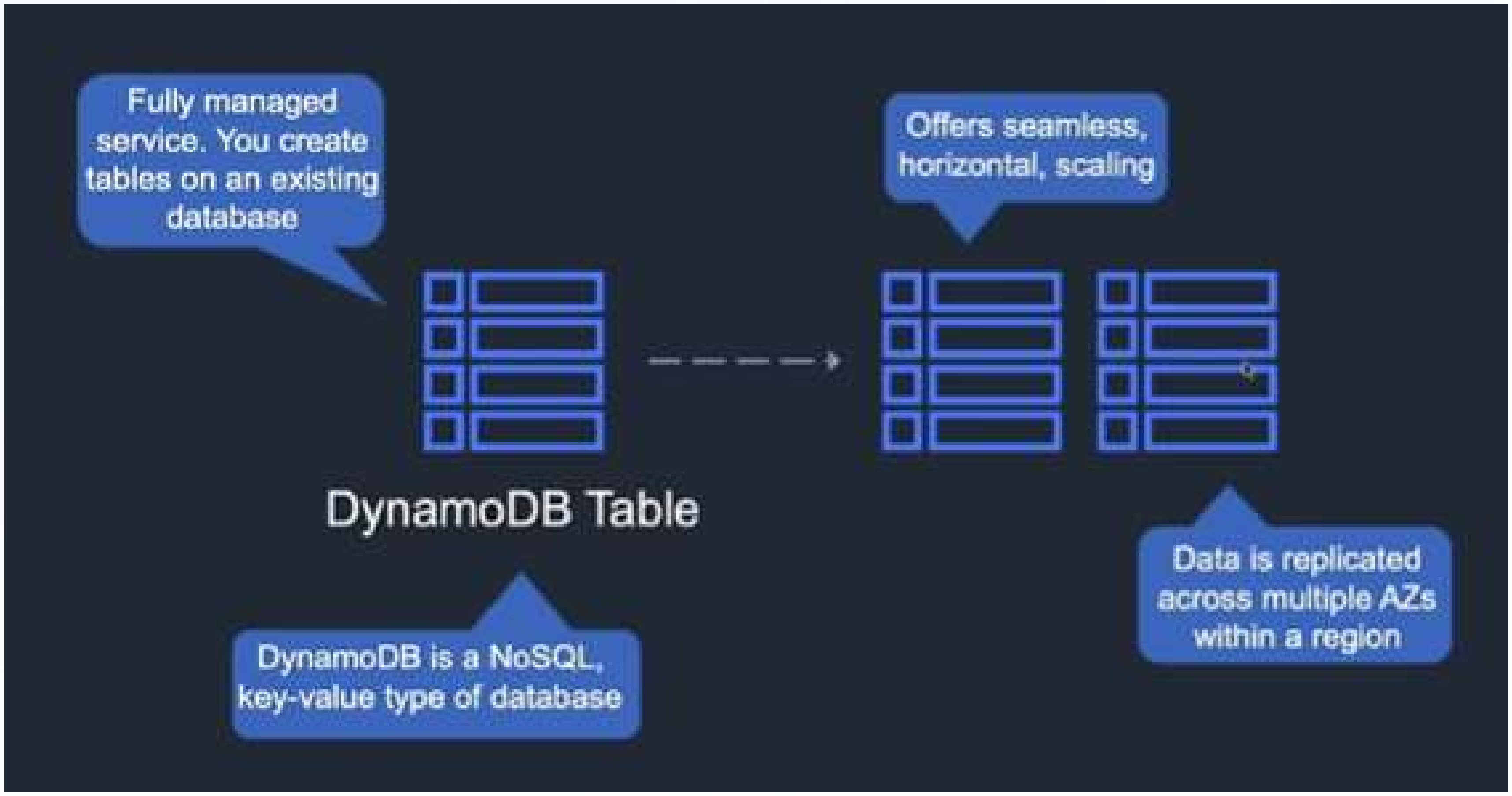
Amazon DynamoDB

- Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability.
- DynamoDB lets you offload the administrative burdens of operating and scaling a distributed database so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling.
- DynamoDB also offers encryption at rest, which eliminates the operational burden and complexity involved in protecting sensitive data.

Amazon DynamoDB

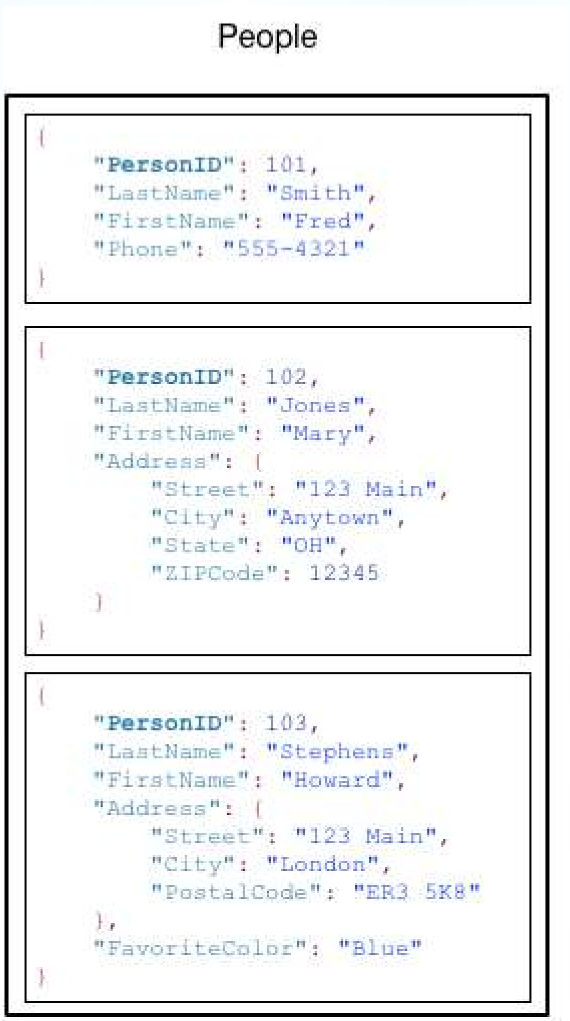
- With DynamoDB, you can create database tables that can store and retrieve any amount of data and serve any level of request traffic.
- You can scale up or scale down your tables' throughput capacity without downtime or performance degradation.
- You can use the AWS Management Console to monitor resource utilization and performance metrics.
- 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.

Amazon DynamoDB


Fully managed service. You create tables on an existing database
Offers seamless, horizontal, scaling
DynamoDB Table
DynamoDB is a NoSQL, key-value type of database
Data is replicated across multiple AZs within a region

DynamoDB Components

- **Tables, Items & Attributes**
  - **Table**: A *table* is a collection of data.
  - **Item**: An *item* is a group of attributes that is uniquely identifiable among all of the other items. ***(each JSON inside the table)***
  - **Attributes**: Each item is composed of one or more attributes. An *attribute* is a fundamental data element, something that does not need to be broken down any further.
- **Primary Key**
  - Primary Key is mandatory for every table.
  - The primary key uniquely identifies each item in the table, so that no two items can have the same key. *(PersonId in the example)*



People

```
{
    "PersonID": 101,
    "LastName": "Smith",
    "FirstName": "Fred",
    "Phone": "555-4321"
}
```

```
{
    "PersonID": 102,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
        "Street": "123 Main",
        "City": "Anytown",
        "State": "OH",
        "ZIPCode": 12345
    }
}
```

```
{
    "PersonID": 103,
    "LastName": "Stephens",
    "FirstName": "Howard",
    "Address": {
        "Street": "123 Main",
        "City": "London",
        "PostalCode": "ER3 5K8"
    },
    "FavoriteColor": "Blue"
}
```

DynamoDB Components - Primary Key

- DynamoDB supports two different kinds of primary keys:
  - **Partition key**
    - A simple primary key, composed of one attribute known as the *partition key*.
    - Partition key's value is used as input to an internal hash function which determines the partition (physical storage internal to DynamoDB) in which the item will be stored.
    - In a table that has only a partition key, no two items can have the same partition key value.
  - **Partition key and sort key**
    - Is a *composite primary key with* two attributes - *partition key* and *sort key*.
    - In a table that has a partition key and a sort key, it's possible for multiple items to have the same partition key value. However, those items must have different sort key values.
- These keys can only be of **String**, **Binary** or **Number** data types.

DynamoDB Components - Secondary Indexes

- You can create one or more secondary indexes on a table. A *secondary index* lets you query the data in the table using an alternate key, in addition to queries against the primary key.

- 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 quota of 20 global secondary indexes (default quota) and 5 local secondary indexes.

Creating a table

- Go to Services >> Database >> DynamoDB
- Click on **Create Table** command to create a table
- Add Table details:
  - Table name: Emp
  - Partition key: EmpId (mandatory primary key)
  - Sort key: FirstName (optional)
- Click on **Create Table** button

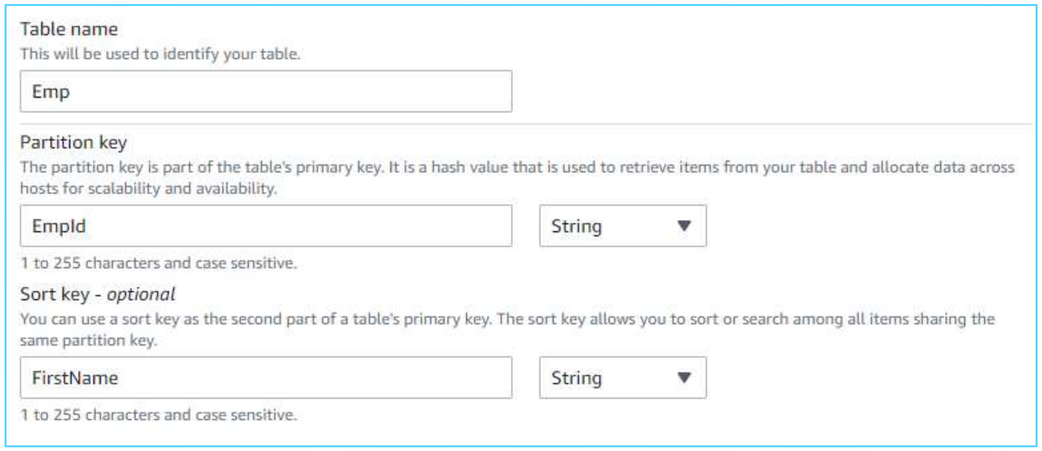


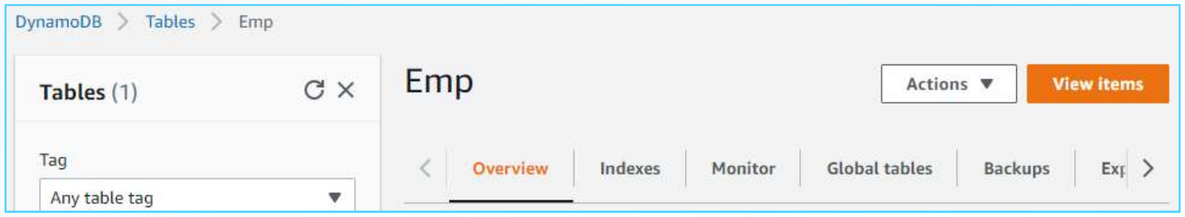
Table name
This will be used to identify your table.
Emp

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
EmpId
String
1 to 255 characters and case sensitive.

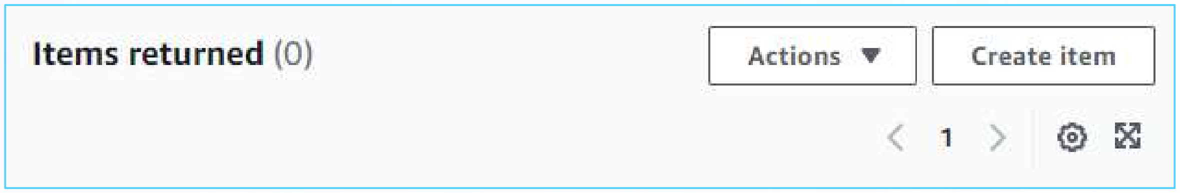
Sort key - *optional*
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
FirstName
String
1 to 255 characters and case sensitive.

Adding Items to the Table

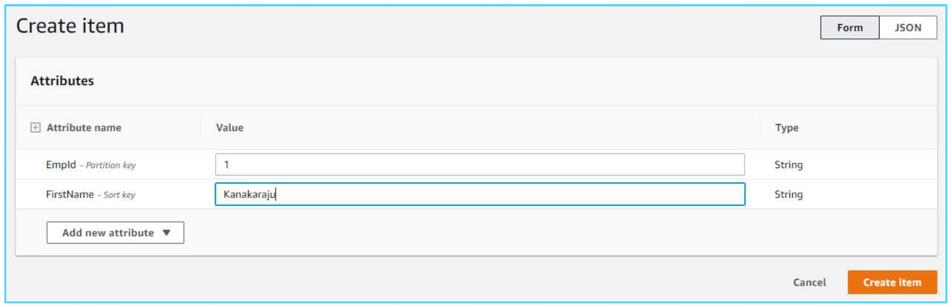
- Click on the table that is created to go to details page.
- Click on **View Items** button.



- Click on **Create Item** button under **Items returned.**



- Add the values for mandatory attributed. Add additional attributes as required.


Create item

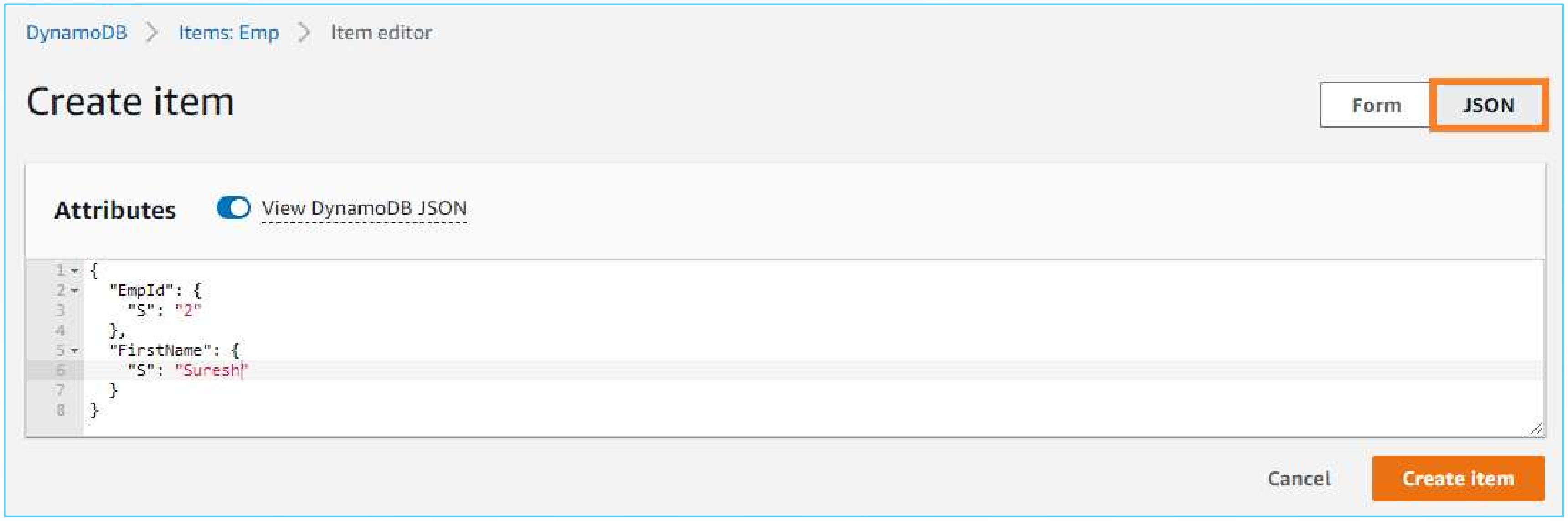
Form | JSON

Attributes

Attribute name	Value	Type
EmpId - *Partition key*	1	String
FirstName - *Sort key*	Kanakaraju	String

Adding Items to the Table - Using JSON

- Go to the **Create Item** page *(refer previous slide)*
- Click on JSON Tab
- Enter the **JSON** content and click on **Create Item** Button



```
{
  "EmpId": {
    "S": "2"
  },
  "FirstName": {
    "S": "Suresh"
  }
}
```

Fetch Items from a Table

- Open the details page of a table and click on **View Items** button
- Click on **Query** Tab and enter the query criteria.
- Click on **Run** button

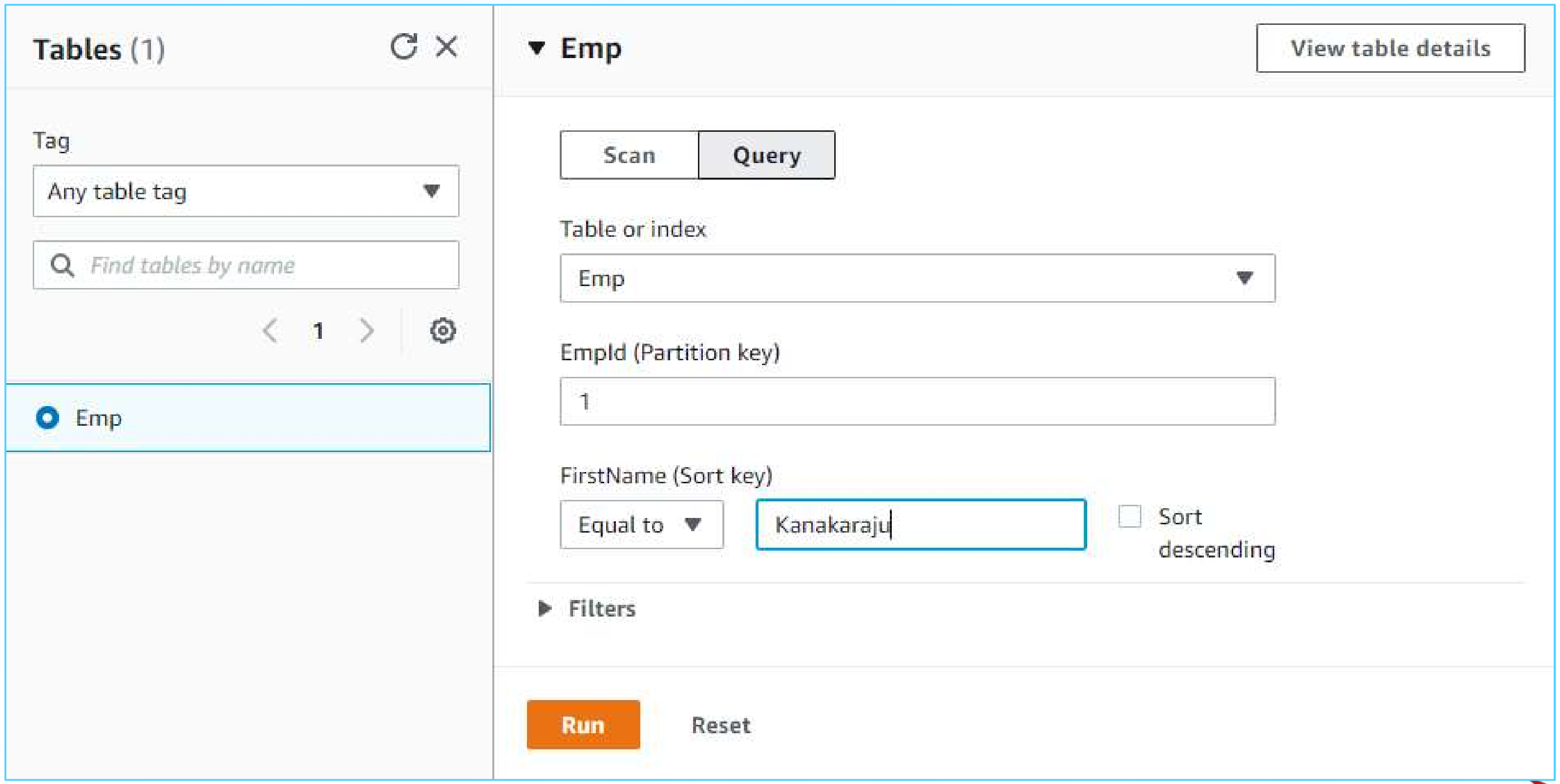
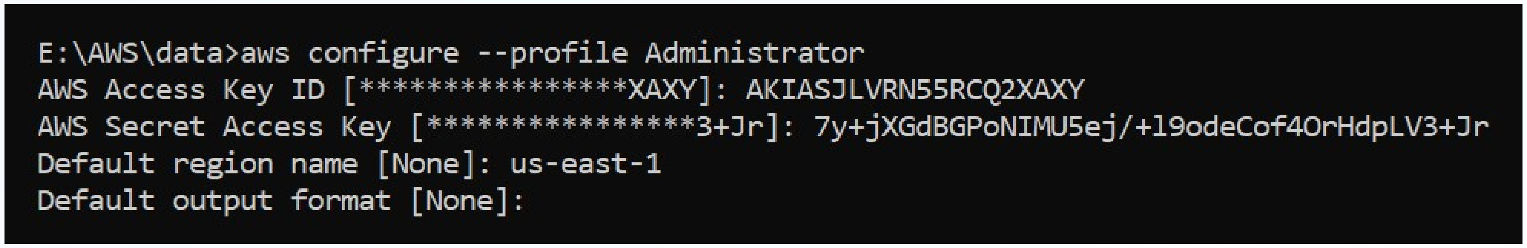


Table or index: Emp
EmpId (Partition key): 1
FirstName (Sort key): Equal to Kanakaraju
☐ Sort descending

Exporting items to a Table from AWS CLI

- Make sure you installed AWS CLI
- Create an AWS user profile using Access keys: (specify the region)



```
E:\AWS\data>aws configure --profile Administrator
AWS Access Key ID [****************XAXY]: AKIASJLVRN55RCQ2XAXY
AWS Secret Access Key [****************3+Jr]: 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
Default region name [None]: us-east-1
Default output format [None]:
```

- Enter the following command on the CLI to export to a DynamoDB table.

```
aws dynamodb batch-write-item --request-items file://mystore.json
```

  - The above command exports the items from the JSON file.

Working with DynamoDB from AWS CLI

```
aws dynamodb batch-write-item --request-items file://mystore.json

aws dynamodb scan --table-name mystore

aws dynamodb query  --table-name mystore --key-conditions '{ "clientid":{
"ComparisonOperator":"EQ", "AttributeValueList": [ {"S":
"chris@example.com"} ] } }`

aws dynamodb query --table-name mystore --key-condition-expression
"clientid = :name" --expression-attribute-values
'{":name":{"S":"chris@example.com"}}'
```

THANK YOU