**AWS Dynamo DB**

Amazon DynamoDB is a fully managed NoSQL database service offered by Amazon Web Services (AWS). It is designed to provide a highly scalable, low-latency, and high-performance database solution for applications that require fast and predictable performance with seamless scalability.

**Key features:**

**1. NoSQL Database:** DynamoDB is a NoSQL database, which means it doesn't rely on traditional relational database structures like tables with fixed schemas. Instead, it uses a schema-less format where each item can have different attributes, allowing for flexible data modeling.

**2. Managed Service:** DynamoDB is a fully managed service, which means AWS handles tasks like provisioning, patching, scaling, and backups. This enables developers to focus more on their application logic and less on database management.

**3. Scalability:** DynamoDB is designed to handle high-traffic and highly-scalable workloads. It automatically scales both read and write capacity to accommodate changes in demand. You can specify the desired read and write capacity units, and DynamoDB will manage the scaling for you.

**4. Primary Key:** Each item in DynamoDB must have a primary key, which consists of one or two attributes: the partition key (required) and an optional sort key. The combination of partition key and sort key allows you to uniquely identify items and also perform range queries.

**5. Global Secondary Indexes (GSI) and Local Secondary Indexes (LSI):** These are indexing mechanisms that allow you to query data based on attributes other than the primary key. Global Secondary Indexes can be created on any attribute, while Local Secondary Indexes can only be created on attributes that are part of the primary key's sort key.

**6. Data Types:** DynamoDB supports various data types, including strings, numbers, binary data, boolean, null, lists, sets, and maps.

**7. Billing:** DynamoDB pricing is based on provisioned capacity, on-demand usage, and additional features like global tables, streams, etc. It's important to understand the pricing model to optimize costs.

**NoSQL (which stands for "Not Only SQL")** is a category of databases that diverges from traditional relational databases (SQL databases) in terms of data modeling, storage, and retrieval. NoSQL databases are designed to handle large volumes of unstructured or semi-structured data, making them well-suited for applications that require high scalability, performance, and flexibility. They are commonly used for web applications, real-time analytics, and other scenarios where the data schema may evolve rapidly.

NoSQL databases depart from the rigid structure of SQL databases, where data is stored in tables with predefined schemas and relationships. Instead, NoSQL databases use various data models to store and manage data.

**1. Document Stores:**

Document-oriented databases store data in documents, typically in formats like JSON or BSON. Each document can have a different structure, allowing for flexible and dynamic data schemas. Examples of document stores include:

- MongoDB: Stores data in JSON-like documents and is suitable for a wide range of applications.

**2. Key-Value Stores:**

Key-value stores store data as a collection of key-value pairs. They are very efficient for simple data retrieval but lack complex querying capabilities.

- Redis: A high-performance, in-memory key-value store often used for caching and real-time analytics.

**3. Column-Family Stores:**

Column-family stores organize data into column families, which are groups of related data. Each column family can have a different set of columns, allowing for sparse data storage.

- Apache Cassandra: Designed for scalability and fault tolerance, it's used in applications that require high availability and distribution across multiple nodes.

**4. Graph Databases:**

Graph databases represent data using nodes, edges, and properties. They excel at managing data with complex relationships, like social networks and recommendation engines.

- Neo4j: A popular graph database used for querying and analyzing complex connected data.

**Examples of NoSQL databases:**

1. Amazon DynamoDB: A managed NoSQL database service offered by Amazon Web Services (AWS) that provides high availability and scalability, making it suitable for various applications.
2. Couchbase: A NoSQL database that combines the flexibility of a document store with the distributed architecture of a key-value store.
3. Cassandra: An open-source distributed database system that can handle large amounts of data across many commodity servers.
4. Elasticsearch: While primarily used for full-text search, Elasticsearch also stores and retrieves structured data efficiently.
5. Riak: A distributed key-value store that focuses on high availability and fault tolerance.

**Java AWS SDK v2 DynamoDB:**

**1. Setting up AWS SDK:**

First, you need to set up the AWS SDK for Java in your project. You can do this by including the SDK as a dependency in your build configuration (e.g., Maven or Gradle). Here's an example for Maven:

<dependency>

<groupId>software.amazon.awssdk</groupId>

<artifactId>dynamodb</artifactId>

<version>2.17.49</version>

</dependency>

**2. Configuring AWS Credentials:**

Before using any AWS service, you need to provide your AWS credentials (Access Key ID and Secret Access Key) to the SDK. You can do this using environment variables, system properties, or AWS configuration files. The preferred way is using the `DefaultCredentialsProvider`:

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;

DefaultCredentialsProvider credentialsProvider = DefaultCredentialsProvider.create();

**3. Creating DynamoDB Client:**

Once you have the credentials set up, you can create a DynamoDB client to interact with the DynamoDB service:

import software.amazon.awssdk.services.dynamodb.DynamoDbClient;

DynamoDbClient dynamoDbClient = DynamoDbClient.builder()

.credentialsProvider(credentialsProvider)

.region(Region.YOUR\_REGION) // Replace with your desired region

.build();

4. Basic Operations:

Here's how you can perform basic CRUD operations using DynamoDB:

**- Creating an Item:**

import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;

import software.amazon.awssdk.services.dynamodb.model.AttributeValue;

Map<String, AttributeValue> itemValues = new HashMap<>();

itemValues.put("primaryKey", AttributeValue.builder().s("someValue").build());

itemValues.put("attributeName", AttributeValue.builder().s("someData").build());

PutItemRequest request = PutItemRequest.builder()

.tableName("yourTableName")

.item(itemValues)

.build();

dynamoDbClient.putItem(request);

**- Reading an Item:**

import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;

import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;

GetItemRequest request = GetItemRequest.builder()

.tableName("yourTableName")

.key(Map.of("primaryKey", AttributeValue.builder().s("someValue").build()))

.build();

GetItemResponse response = dynamoDbClient.getItem(request);

Map<String, AttributeValue> item = response.item();

**- Updating an Item:**

Similarly, you can use `UpdateItemRequest` to update an existing item.

**- Deleting an Item:**

Use `DeleteItemRequest` to delete an item.

AWS SDK for Java v2 Documentation <https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/home.html>