Amazon DynamoDB is a fast and flexible **NoSQL database** service for all applications that need consistent, single-digit millisecond latency at any scale. It is a fully managed cloud database and supports both document and key-value store models. Its flexible data model, reliable performance and automatic scaling of throughput capacity, makes it a great fit for mobile, web, gaming, ad tech, IoT and many other applications.

NoSQL is a term used to describe high-performance, non-relational databases. NoSQL databases utilize a variety of data models, including document, graph, key-value, and columnar. NoSQL databases are widely recognized for ease of development, scalable performance, high availability, and resilience.

Document Databases

Document databases are designed to store semi-structured data as documents, typically in JSON or XML format. Unlike traditional relational databases, the schema for each non-relational (NoSQL) document can vary, giving you more flexibility in organizing and storing application data and reducing storage required for optional values.

In-Memory Key-Value Stores

In-memory key-value stores are NoSQL databases optimized for read-heavy application workloads (such as social networking, gaming, media sharing and Q&A portals) or compute-intensive workloads (such as a recommendation engine). In-memory caching improves application performance by storing critical pieces of data in memory for low-latency access

SQL vs NoSQL Terminology

SQL	MongoDB	DynamoDB
Table	Collection	Table
Row	Document	Item
Column	Field	Attribute
Primary Key	ObjectId	Primary Key
Index	Index	Secondary Index
View	View	Global Secondary Index
Nested Table or Object	Embedded Document	Map
Array	Array	List

DECCANSOFT

SQL or NoSQL?

Today's applications have more demanding requirements than ever before. For example, an online game might start out with just a few users and a very small amount of data. However, if the game becomes successful, it can easily outstrip the resources of the underlying database management system. It is not uncommon for web-based applications to have thousands or millions of concurrent users, with gigabytes or more of new data generated per day. Databases for such applications must handle thousands of reads and writes per second.

Amazon DynamoDB is well-suited for these kinds of workloads. As a developer, we can start with a small amount of provisioned throughput and gradually increase it as our application becomes more popular. DynamoDB scales seamlessly to handle very large amounts of data and very large numbers of users.

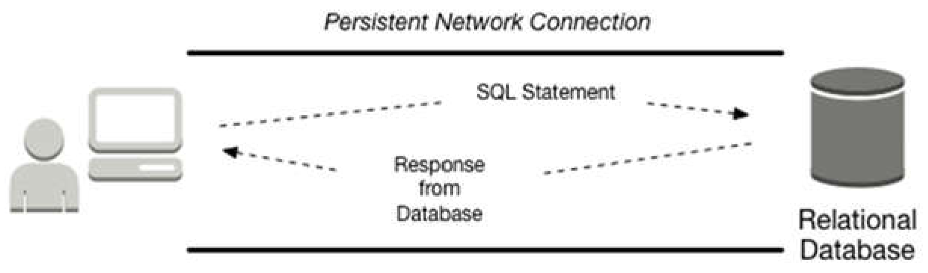
The following table shows some high-level differences between a relational database management system (RDBMS) and DynamoDB:

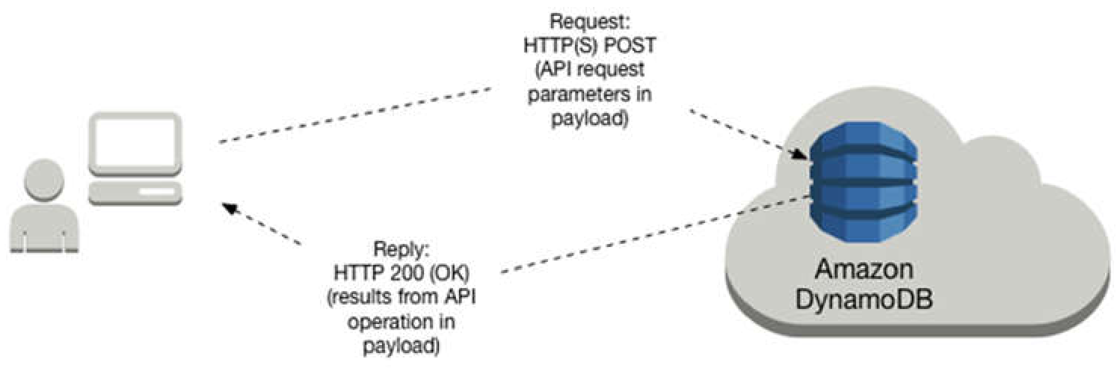
Characteristic	Relational Database Management System (RDBMS)	Amazon DynamoDB
Optimal Workloads	Ad hoc queries; data warehousing; OLAP (online analytical processing).	Web-scale applications, including social networks, gaming, media sharing, and IoT (Internet of Things).
Data Model	The relational model requires a well-defined schema, where data is normalized into tables, rows and columns. In addition, all of the relationships are defined among tables, columns, indexes, and other database elements.	DynamoDB is schema-less. Every table must have a primary key to uniquely identify each data item, but there are no similar constraints on other non-key attributes. DynamoDB can manage structured or semi-structured data, including JSON documents.
Data Access	SQL (Structured Query Language) is the standard for storing and retrieving data. Relational databases offer a rich set of tools for simplifying the development of database-driven applications, but all of these tools use SQL.	We can use the AWS Management Console or the AWS CLI to work with DynamoDB and perform ad hoc tasks. Applications can leverage the AWS software development kits (SDKs) to work with DynamoDB using object-based, document-centric, or low-level interfaces.

DECCANSOFT

Performance	Relational databases are optimized for storage, so performance generally depends on the disk subsystem. Developers and database administrators must optimize queries, indexes, and table structures in order to achieve peak performance.	DynamoDB is optimized for compute, so performance is mainly a function of the underlying hardware and network latency. As a managed service, DynamoDB insulates us and our applications from these implementation details, so that we can focus on designing and building robust, high-performance applications.
Scaling	It is easiest to scale up with faster hardware. It is also possible for database tables to span across multiple hosts in a distributed system, but this requires additional investment. Relational databases have maximum sizes for the number and size of files, which imposes upper limits on scalability.	DynamoDB is designed to scale out using distributed clusters of hardware. This design allows increased throughput without increased latency. Customers specify their throughput requirements, and DynamoDB allocates sufficient resources to meet those requirements. There are no upper limits on the number of items per table, nor the total size of that table.

The following diagram shows client interaction with a relational database, and with DynamoDB.




Request: HTTP(S) POST (API request parameters in payload)
Reply: HTTP 200 (OK) (results from API operation in payload)
Amazon DynamoDB

The following table has more details about client interaction tasks:

Characteristic	Relational Database Management System (RDBMS)	Amazon DynamoDB
Tools for Accessing the Database	Most relational databases provide a command line interface (CLI), so that you can enter ad hoc SQL statements and see the results immediately.	In most cases, you write application code. You can also use the AWS Management Console or the AWS Command Line Interface (AWS CLI) to send requests to DynamoDB and view the results.
Connecting to the Database	An application program establishes and maintains a network connection with the database. When the application is finished, it terminates the connection.	DynamoDB is a web service, and interactions with it are stateless. Applications do not need to maintain persistent network connections. Instead, interaction with DynamoDB occurs using HTTP(S) requests and responses.
Authentication	An application cannot connect to the database until it is authenticated. The RDBMS can perform the authentication itself, or it can offload this task to the host operating system or a directory service.	Every request to DynamoDB must be accompanied by a cryptographic signature, which authenticates that particular request. The AWS SDKs provide all of the logic necessary for creating signatures and signing requests.
Authorization	Applications can only perform actions for which they have been authorized. Database administrators or application owners can use the SQL GRANT and REVOKE statements to control access to database objects (such as tables), data (such as rows within a table), or the ability to issue certain SQL statements.	In DynamoDB, authorization is handled by AWS Identity and Access Management (IAM). You can write an IAM policy to grant permissions on a DynamoDB resource (such as a table), and then allow IAM users and roles to use that policy. IAM also features fine-grained access control for individual data items in DynamoDB tables.
Sending a Request	The application issues a SQL statement for every database operation that it wants to perform. Upon receipt of the SQL statement, the RDBMS checks its syntax, creates a plan for performing the operation, and then executes the plan.	The application sends HTTP(S) requests to DynamoDB. The requests contain the name of the DynamoDB operation to perform, along with parameters. DynamoDB executes the request immediately.
Receiving a Response	The RDBMS returns the results from the SQL statement. If there is an error, the RDBMS returns an error status and message.	DynamoDB returns an HTTP(S) response containing the results of the operation. If there is an error, DynamoDB returns an HTTP error status and message(s).

DECCANSOFT

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

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

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

DynamoDB automatically spreads the data and traffic for your tables over a sufficient number of servers to handle your throughput and storage requirements, while maintaining consistent and fast performance. All of our data is stored on solid state disks (SSDs) and automatically replicated across multiple Availability Zones in an AWS region, providing built-in high availability and data durability.

DECCANSOFT

DynamoDB Core Components

In DynamoDB, tables, items and attributes are the core components that we work with. A *table* is a collection of *items* and each item is a collection of *attributes*. DynamoDB uses primary keys to uniquely identify each item in a table and secondary indexes to provide more querying flexibility. You can use DynamoDB Streams to capture data modification events in DynamoDB tables.

The following are the basic DynamoDB components:

- **Tables** – Similar to other database systems, DynamoDB stores data in tables. A *table* is a collection of data. For example, see the example table called *People* that we could use to store personal contact information about friends, family, or anyone else of interest. We could also have a *Cars* table to store information about cars that people drive.

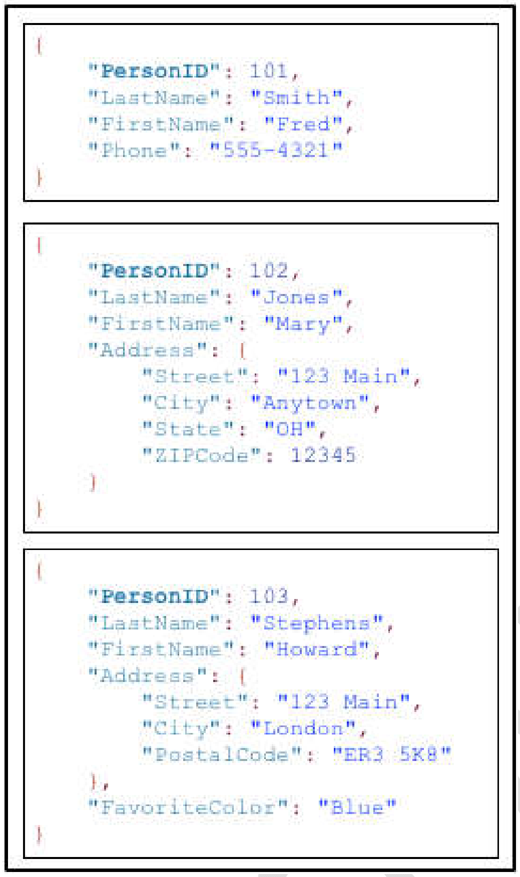
- **Items** – Each table contains multiple items. An *item* is a group of attributes that is uniquely identifiable among all of the other items. In a *People* table, each item represents a person. For a *Cars* table, each item represents one car. Items in DynamoDB are similar in many ways to rows, records, or tuples in other database systems. In DynamoDB, there is no limit to the number of items we can store in a 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. For example, an item in a *People* table contains attributes called *PersonID*, *LastName*, *FirstName*, and so on. For a *Department* table, an item might have attributes such as *DepartmentID*, *Name*,*Manager*, and so on. Attributes in DynamoDB are similar in many ways to fields or columns in other database systems.

DECCANSOFT

The following diagram shows a table named *People* with some example items and attributes.

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"
}
```

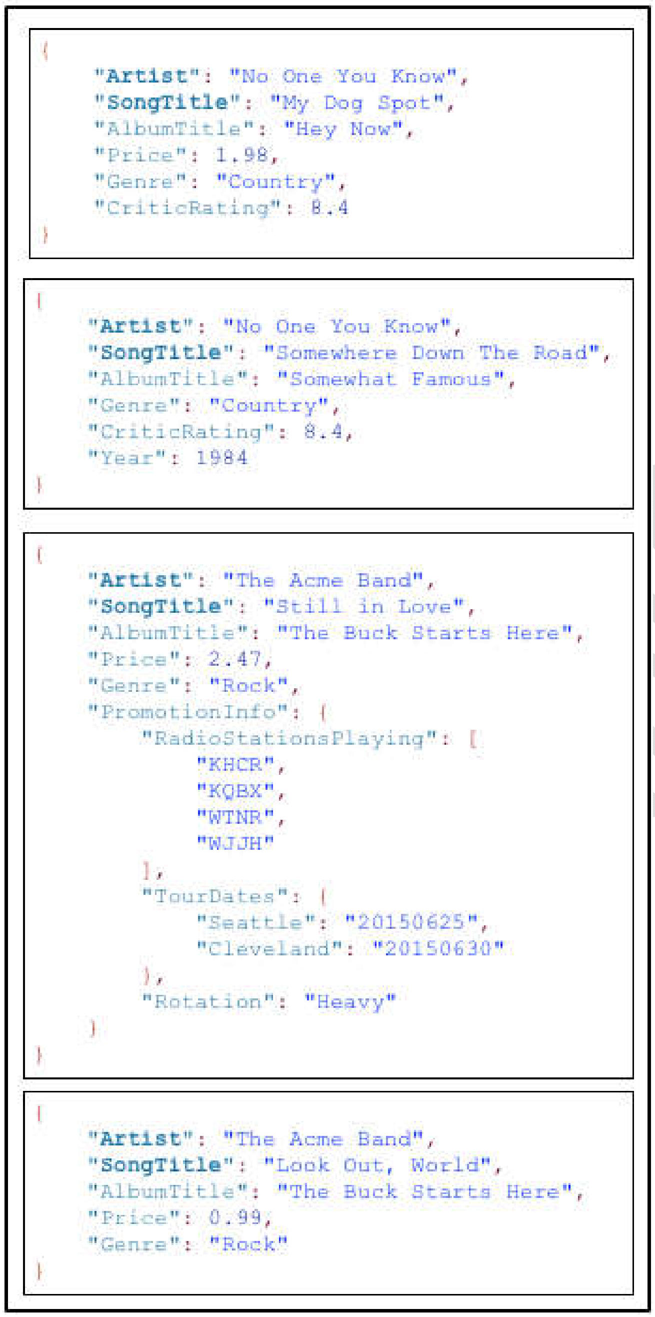
DECCANSOFT

Note the following about the *People* table:

- Each item in the table has a unique identifier, or primary key, that distinguishes the item from all of the others in the table. In the *People* table, the primary key consists of one attribute (*PersonID*).
- Other than the primary key, the *People* table is schemaless, which means that neither the attributes nor their data types need to be defined beforehand. Each item can have its own distinct attributes.
- Most of the attributes are *scalar*, which means that they can have only one value. Strings and numbers are common examples of scalars.
- Some of the items have a nested attribute (*Address*). DynamoDB supports nested attributes up to 32 levels deep.

The following is another example table named *Music* that we could use to keep track of your music collection.

Music



```
{
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot",
    "AlbumTitle": "Hey Now",
    "Price": 1.98,
    "Genre": "Country",
    "CriticRating": 8.4
}
```

```
{
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The Road",
    "AlbumTitle": "Somewhat Famous",
    "Genre": "Country",
    "CriticRating": 8.4,
    "Year": 1984
}
```

```
{
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 2.47,
    "Genre": "Rock",
    "PromotionInfo": {
        "RadioStationsPlaying": [
            "KHCR",
            "KQBX",
            "WTNR",
            "WJJH"
        ],
        "TourDates": {
            "Seattle": "20150625",
            "Cleveland": "20150630"
        },
        "Rotation": "Heavy"
    }
}
```

```
{
    "Artist": "The Acme Band",
    "SongTitle": "Look Out, World",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 0.99,
    "Genre": "Rock"
}
```

Note the following about the *Music* table:

- The primary key for *Music* consists of two attributes (*Artist* and *SongTitle*). Each item in the table must have these two attributes. The combination of *Artist* and *SongTitle* distinguishes each item in the table from all of the others.
- Other than the primary key, the *Music* table is schemaless, which means that neither the attributes nor their data types need to be defined beforehand. Each item can have its own distinct attributes.
- One of the items has a nested attribute (*PromotionInfo*), which contains other nested attributes. DynamoDB supports nested attributes up to 32 levels deep.

DECCANSOFT