DynamoDB

DynamoDB

- Amazon DynamoDB is a key-value and document database that delivers single-digit millisecond performance at any scale. It's a fully managed, multiregional, multimaster, durable database with built-in security, backup and restore, and in-memory caching for internet-scale applications. DynamoDB can handle more than 10 trillion requests per day and can support peaks of more than 20 million requests per second
- Hundreds of thousands of AWS customers have chosen DynamoDB as their key-value and document database for mobile, web, gaming, ad tech, IoT, and other applications that need low-latency data access at any scale. Create a new table for your application and let DynamoDB handle the rest.

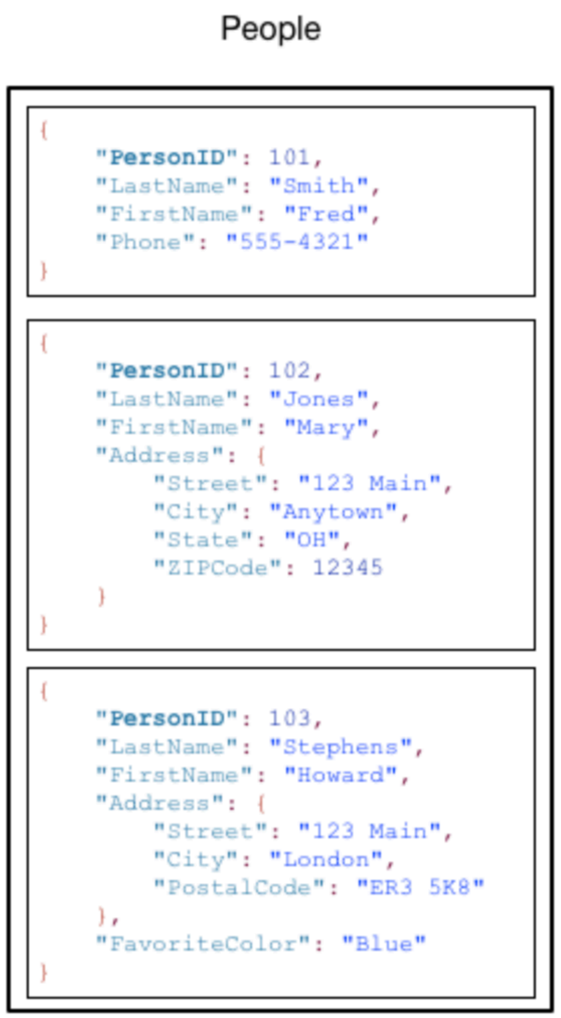
DynamoDB - Applications

- Build powerful web applications that automatically scale up and down. You don't need to maintain servers, and your applications have automated high availability.
- Use DynamoDB and AWS AppSync to build interactive mobile and web apps with real-time updates, offline data access, and data sync with built-in conflict resolution.
- Build flexible and reusable microservices using DynamoDB as a serverless data store for consistent and fast performance.

DynamoDB

- **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 you could use to store personal contact information about friends, family, or anyone else of interest. You could also have a Cars table to store information about vehicles that people drive.
- **Items** – Each table contains zero or more 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 vehicle. 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 you 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.

DynamoDB



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

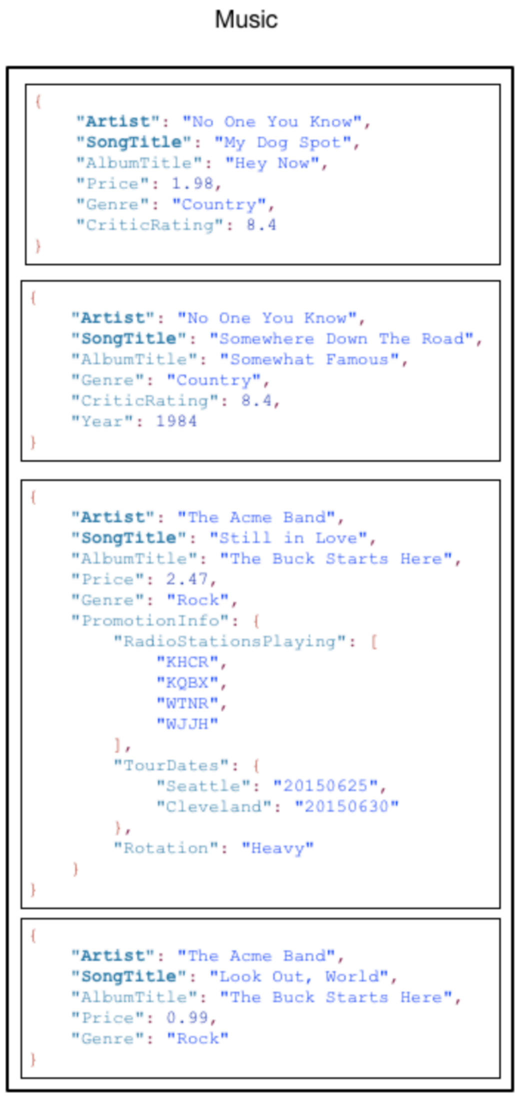
```
{
    "PersonID": 102,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
        "Street": "123 Main",
        "City": "Anytown",
        "State": "OH",
        "ZIPCode": 12345
    }
}
```

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

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).

Partition key – A simple primary key, composed of one attribute known as the partition key.
DynamoDB uses the partition key's value as input to an internal hash function. The output from the hash function 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.

DynamoDB



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"
}
```

Partition key and sort key – Referred to as a *composite 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*.
DynamoDB uses the partition key value as input to an internal hash function. The output from the hash function determines the partition (physical storage internal to DynamoDB) in which the item will be stored. All items with the same partition key value are stored together, in sorted order by sort key value.
In a table that has a partition key and a sort key, it's possible for two items to have the same partition key value. However, those two items must have different sort key values.

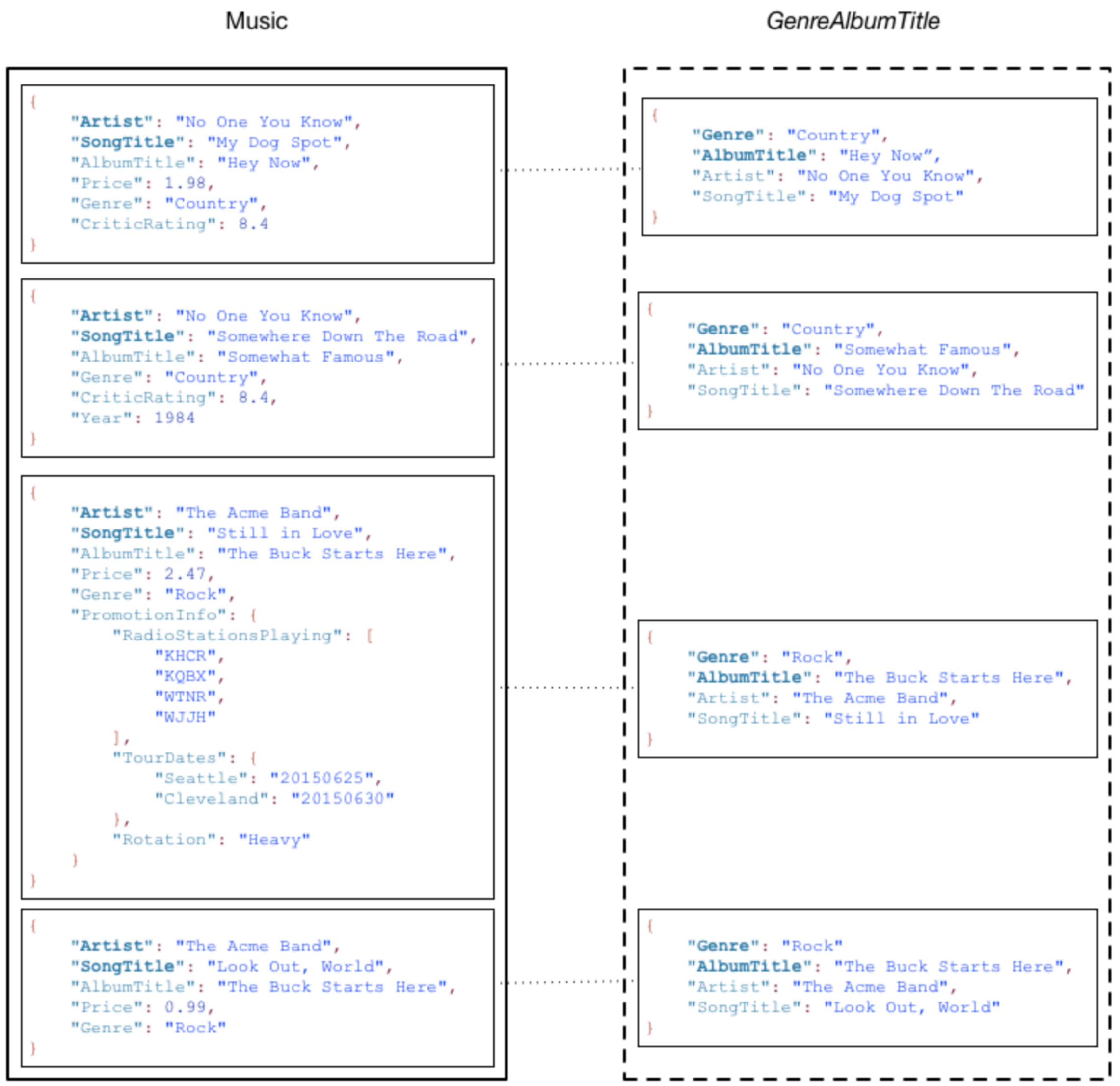
DynamoDB - Indexes

- 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 doesn't require that you use indexes, but they give your applications more flexibility when querying your data. After you create a secondary index on a table, you can read data from the index in much the same way as you do from the 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 - Indexes



```
{
    "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"
}
```

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

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

```
{
    "Genre": "Rock",
    "AlbumTitle": "The Buck Starts Here",
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love"
}
```

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

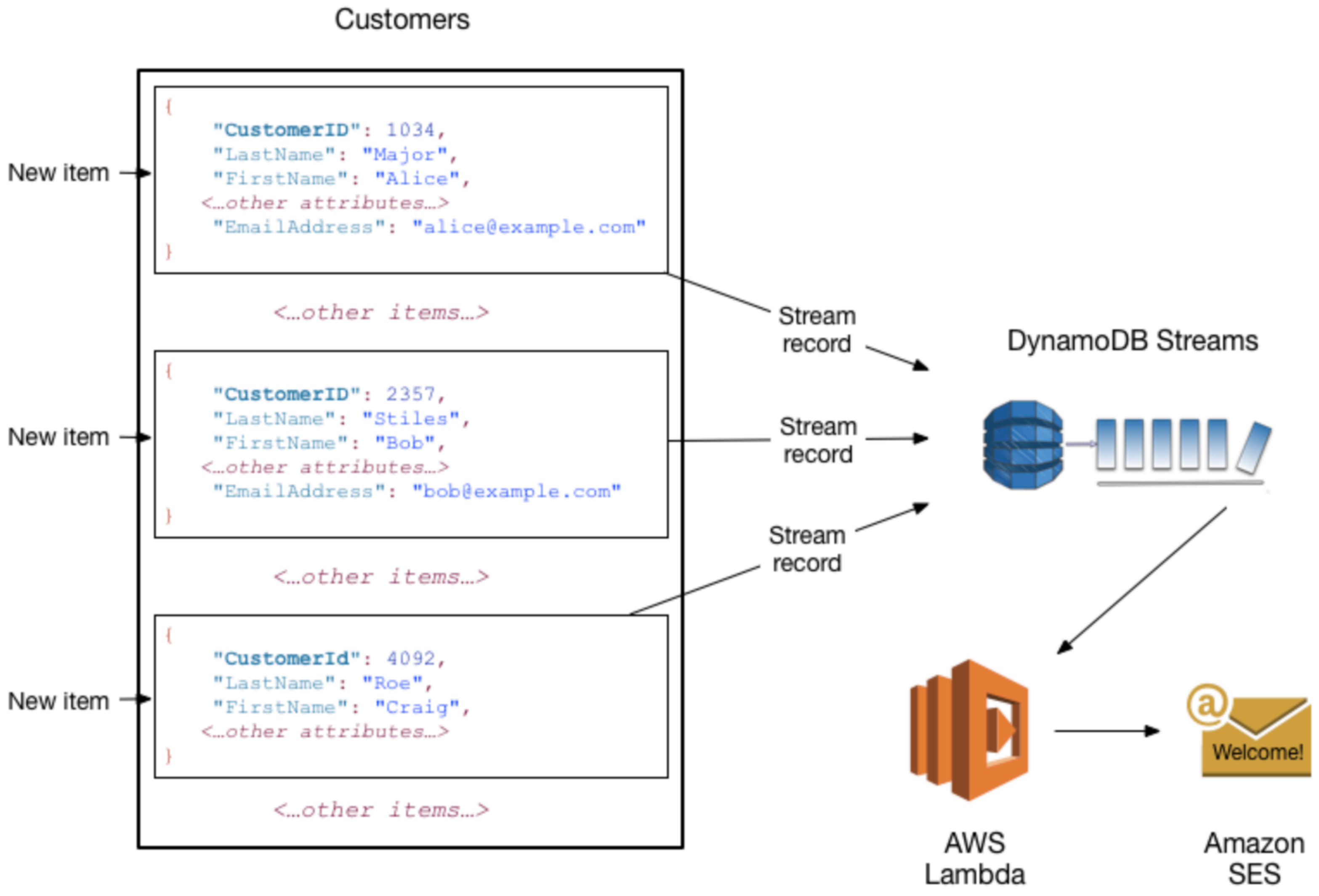
DynamoDB - Streams

- DynamoDB Streams is an optional feature that captures data modification events in DynamoDB tables. The data about these events appear in the stream in near-real time, and in the order that the events occurred.

Each event is represented by a stream record. If you enable a stream on a table, DynamoDB Streams writes a stream record whenever one of the following events occurs:

- A new item is added to the table: The stream captures an image of the entire item, including all of its attributes.
- An item is updated: The stream captures the "before" and "after" image of any attributes that were modified in the item.
- An item is deleted from the table: The stream captures an image of the entire item before it was deleted.

DynamoDB - Streams



```
{
   "CustomerID": 1034,
   "LastName": "Major",
   "FirstName": "Alice",
 <…other attributes…>
   "EmailAddress": "alice@example.com"
}
```

```
{
   "CustomerID": 2357,
   "LastName": "Stiles",
   "FirstName": "Bob",
 <…other attributes…>
   "EmailAddress": "bob@example.com"
}
```

```
{
   "CustomerId": 4092,
   "LastName": "Roe",
   "FirstName": "Craig",
 <…other attributes…>
}
```

DynamoDB – Global Tables

- A global table is a collection of one or more replica tables, all owned by a single AWS account.
- A replica table (or replica, for short) is a single DynamoDB table that functions as a part of a global table. Each replica stores the same set of data items. Any given global table can only have one replica table per AWS Region.

To keep your table data in sync, global tables automatically create the following attributes for every item:

- aws:rep:deleting
- aws:rep:updatetime
- aws:rep:updateregion

Do not modify these attributes or create attributes with the same name.

DynamoDB – Consistency Models

- While reading data from DynamoDB, user can specify whether they want the read to be eventually or strongly consistent, these are the two consistency model in DynamoDB.

  Eventually Consistent Reads (Default) – the eventual consistency option is used to maximize the read throughput. Consistency across all copies of data is usually reached within a second. Repeating a read after a short time should return the updated data.

  Strongly Consistent Reads — in addition to eventual consistency, DynamoDB also gives user the flexibility and control to request a strongly consistent read when your application, or an element of your application, requires it. A strongly consistent is used to read and return a result that reflects all the writes that was received as a successful response prior to the read.

DynamoDB - DAX

- Amazon DynamoDB Accelerator (DAX) is a fully managed, highly available, in-memory cache for DynamoDB that delivers up to a 10x performance improvement – from milliseconds to microseconds – even at millions of requests per second. DAX does all the heavy lifting required to add in-memory acceleration to your DynamoDB tables, without requiring developers to manage cache invalidation, data population, or cluster management. Now you can focus on building great applications for your customers without worrying about performance at scale. You do not need to modify application logic, since DAX is compatible with existing DynamoDB API calls. You can enable DAX with just a few clicks in the AWS Management Console or using the AWS SDK.

DynamoDB – Read and Write Units

- For example, suppose that you create a table with 10 provisioned read capacity units. This allows you to perform 10 strongly consistent reads per second, or 20 eventually consistent reads per second, for items up to 4 KB.
- For example, a strongly consistent read of an item that is 8 KB (4 KB × 2) consumes 2 read capacity units. An eventually consistent read on that same item consumes only 1 read capacity unit.
- For example, suppose that you create a table with 10 write capacity units. This allows you to perform 10 writes per second, for items up to 1 KB in size per second.