



AWS DynamoDB

Traditional Architecture



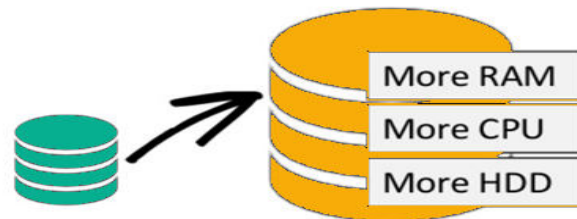
- Traditional application leverage RDBMS databases
- These databases have the SQL query language
- Strong requirements about how the data should be modeled (tables, rows, columns, joins, indexes, etc)
- Ability to do join, aggregations, computations
- Vertical scaling (means usually getting a more powerful CPU/RAM/IO)

NoSQL databases

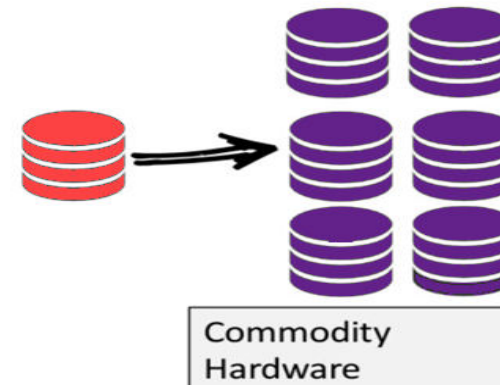


- The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data.
- The system response time becomes slow when you use RDBMS for massive volumes of data.
- To resolve this problem, we could "scale up" our systems by upgrading our existing hardware. This process is expensive.
- The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as "scaling out."

Scale-Up (*vertical scaling*):



Scale-Out (*horizontal scaling*):



NoSQL databases



- NoSQL means, Not Only SQL.
- NoSQL databases are non-relational databases(no fixed columns etc) and are distributed(horizontal scaling).
- NoSQL databases include MongoDB, DynamoDB etc
- NoSQL databases do not support joins
- All the data that is needed for a query is present in one row
- NoSQL databases don't perform aggregations such as "SUM".
- NoSQL databases scale horizontally

NoSQL databases

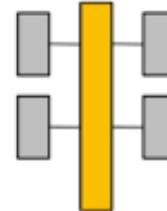


SQL Database

Relational



Analytical (OLAP)



NoSQL Database

Column-Family



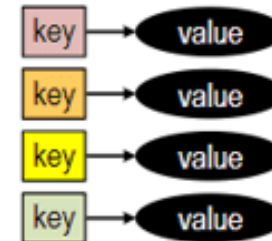
Graph



Document



Key-Value




NoSQL databases



Types of NoSQL Databases

Key Value




Example:
Riak, Tokyo Cabinet, Redis
server, Memcached,
Scalaris

Document-Based



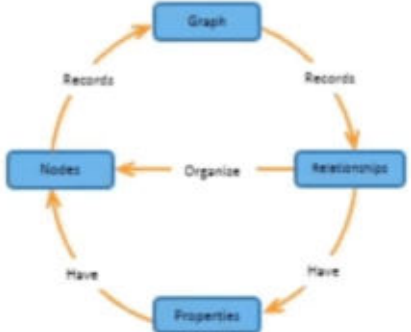
Example:
MongoDB, CouchDB,
OrientDB, RavenDB

Column-Based



Example:
BigTable, Cassandra,
Hbase,
Hypertable

Graph-Based



Example:
Neo4J, InfoGrid, Infinite
Graph, Flock DB

DynamoDB vs RDS



- DynamoDB offers “push button” scaling , meaning that you can scale your database on the fly, without any down time.
- DynamoDB is Serverless
- RDS is not so easy and you usually have to use a bigger instance size or to add a read replica.

What is DynamoDB ?



- Amazon DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale.
- DynamoDB allows you to create a database table that can store and retrieve any amount of data and serve any level of request traffic.
- It is fully managed database and supports both document and key-value data models.
- Its flexible data model and reliable performance make it s great fit for mobile, web, gaming, ad-tech, IoT, and many other applications.
- Transactions provide atomicity, consistency, isolation, and durability (ACID) in DynamoDB *.
- The data in DynamoDB is stored in JSON format. *

Basics of DynamoDB



- Fully Managed, Highly available with replication across 3 AZ
- NoSQL database – not a relational database.
- Scales to massive workloads, distributed database
- Millions of requests per second, trillions of rows, 100s of TB storage
- Fast and Consistent in Performance(low latency on retrieval)
- Integrated with IAM for security, authorization and administration

Basics of DynamoDB



- Stored on SSD storage
- Spread across 3 geographically distinct data centers
- Eventual Consistent Reads(Default)
- Strongly Consistent Reads

Dynamo DB Components



DynamoDB comprises of core basic components that include *Tables*, *Items* and *Attributes*. A Table is a collection of Items, each of which is a collection of one or more attributes. DynamoDB uses primary keys to uniquely identify each item in a table and secondary indexes to provide more querying flexibility.

- **Tables** – DynamoDB stores data in tables which are a collection of data; for example Students Table to store student information data
- **Items** – Each table contains multiple items, where an item is a group of attributes that uniquely identifies it when compared to other items. Items can be considered similar to rows or records in a relational database system
- **Attributes** – Each item is composed of one or more attributes. Attributes are similar to fields or columns. For example, an item in the Students table could be a student's record, that has attributes including StudentID, FirstName, LastName etc.

Dynamo DB Components



- DynamoDB is made of **Tables**
- Each table has a **Primary Key** (must be decided at creation time)
- Each table can have an infinite number of **items**(=rows)
- Each item has **attributes** (can be added over time – can be null)
- Maximum size of a item is 400KB

Read Consistency of DynamoDB



Eventual Consistent Reads

- 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

- A strongly consistent read returns a result that reflects all writes that received a successful response prior to the read.

Read Consistency- Transactions



Transactions

- New Feature from November 2018
- Transaction = Ability to Create / Update / Delete multiple rows in different tables at the same time
- Its an “all or nothing” type of operations
- Write Modes: Standard, Transactional
- Read Modes: Eventual Consistency, Strong Consistency, Transactional
- Consume 2 X of WCU / RCU

Read Consistency- Transactions



- AccountBalance Table

Account_id	balance	Last_transaction_ts
Acct_21	230	1562503085
Acct_45	120	1562503085

- BankTransactions Table

Transaction_id	Transaction_time	From_account_id	From_account_id	value
Tx_12345	1561483349	Acct_45	Acct_21	45
Tx_23456	1562503085	Acct_21	Acct_45	100

A transaction is a write to both table or None !

Read Consistency



When reading an item from DynamoDB, the response may not reflect the results of recently completed writes. This is because DynamoDB maintains multiple copies of the data across multiple availability zones and so it offers eventual consistent reads which are cheaper than strongly consistent reads.

- **Eventual Consistent Reads** – With Event Consistent Read operations, the response might not reflect the results of a recently completed write operation. The response might include some stale data. However, generally, the lag is no more than one second. If you repeat your read request after a short time, the response should return the latest data.
- **Strongly Consistent Reads** – You can request strongly consistent reads and DynamoDB will respond with the most up-to-date data, reflecting the updates from all prior write operations that were successful.

Data Types



Unlike traditional relational databases, where you need to specify the columns, their names as well as data types that will be contained in them,

DynamoDB only requires specifying a Primary Key field to start with.

You do not need to specify all attributes ahead of time of an item; you can add columns on the fly. This gives you the flexibility to expand the schema as required over time.

Data Types



There are three categories of data types:

- **Scalar Types:** String, Number, Binary, Boolean, NULL
- **Set Data Types :** String Set, Number Set, Binary Set
- **Document Data Types :** List, Map

Primary Key



- When you create a table, you need to specify a Primary key which uniquely identifies every item in a database.
DynamoDB support two types of private keys:
- **Partition Key**
- **Partition Key and Sort Key (Composite key)**

Partition Key



- **Partition Key** – The primary key is defined with a **single attribute** and is known as the partition key.
- DynamoDB uses the partition key's value to build an unordered hash index which is used to identify the partition in the which the item will be stored.
- The partition key of an item is also known as its **hash attribute**
- **Note** that if you are only using only a partition key, you cannot have two items on the same table using the same partition key

Partition Key



- Example: `user_id` for a `users` table

user_id

12broiu45

dfi7503df

Partition key
(unique)

First Name

Age

John

46

Katie

31

attributes

This looks like traditional table

Partition Key and Sort Key

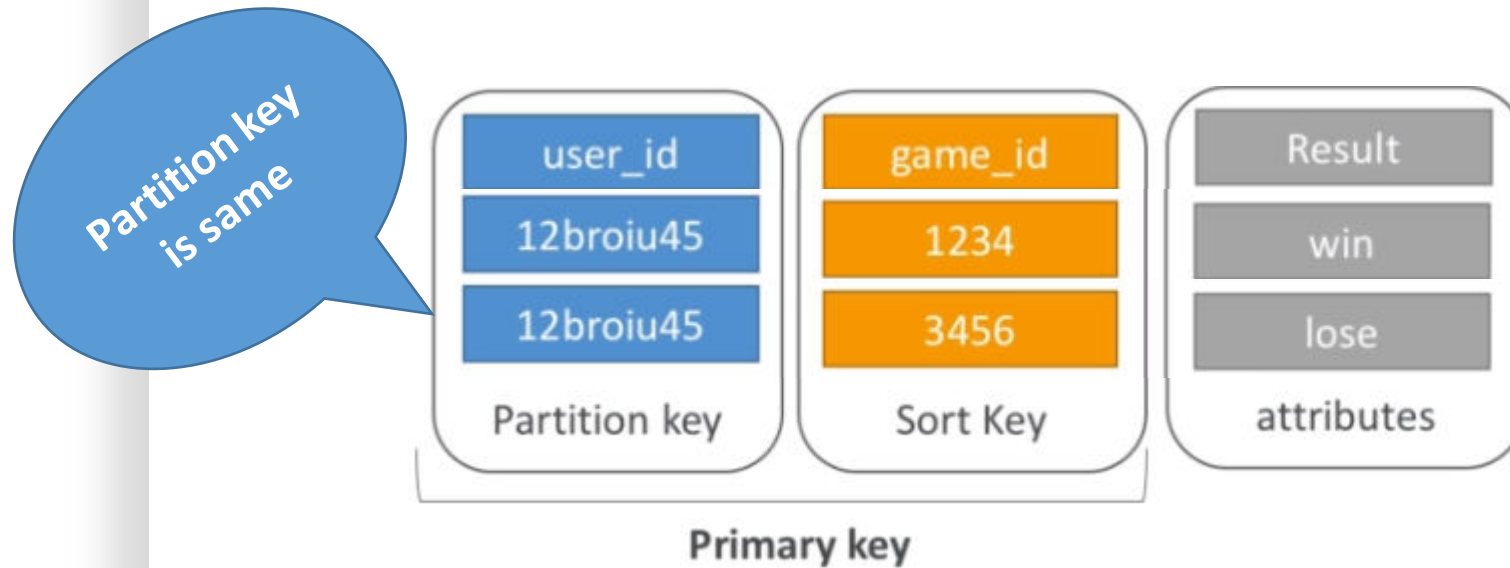


- **Partition Key and Sort Key** – This is known as a **composite primary** key and is made up of two attributes, namely the primary (partition) key and the sort (range) key.
- The **sort key** of an item is known as a **range attribute**
- You can uniquely identify an item if you provide both the partition key and sort key. Note that you can have multiple items with the same partition key if they have different sort keys.

Partition Key + Sort Key



- Example: user_id for a users table



Partition key has duplicate attributes.

However, The combination of Partition key and Sort Key should be unique

Secondary Indexes



- Amazon DynamoDB enables you to query the data in a table using an optionally defined alternative key known as a Secondary Index.
- **Local Secondary Index**
- **Global Secondary Index**

Local & Global Secondary Index



- **Local Secondary Index** – This is an index that has the same partition key as the table, but a different sort key. **These can only be created when the table is created.** Furthermore, you cannot modify or delete a Local Secondary Index once created.
- **Global Secondary Index** – this is an index with a partition key and sort key that can both be different from those on the table. **Global secondary indexes can be created or deleted on a table at any time**
- LSI(Local secondary indexes) = Partition Key + Any Column as sort key
- GSI (Global Secondary Indexes) = Any Column as PK + Any Column as SK

Provisioning Capacity



- DynamoDB enables you to write and read from the tables you create a Database. You can create, update and delete individual items. In addition, you can use multiple querying options to search for data in your tables.
- To ensure high availability and low latency responses, you are required to specify you read and write throughput values when you create a table.
- DynamoDB uses this information to reserve sufficient hardware resources and appropriately partitions your data over multiple servers to meet your throughput requirements. When you create a table, you need to specify the following capacity units

Provisioning Capacity



- **Read Capacity Units (throughput for reads)**
 - Readers are rounded to increments of 4KB in size
 - In *Eventual Consistent Reads*, one read capacity unit is 2 reads per second for items up to 4KB
 - For *Strongly Consistent Reads*, one read capacity unit consist of 1 read per second of up to 4KB in size
- **Write Capacity Units (throughputs for writes)**
 - Number of 1KB writes per second

Read Capacity Units

- For ECR , 1 RCU = 2 reads per second for 4KB size
- For SCR, 1 RCU = 1 read per second for 4KB size

Write Capacity Units

- 1 WCU = 1 write per second



- ***Remember*** – One read capacity unit represents one strongly consistent read per second, or two eventually consistent reads per second, for items up to 4 KB in size
- ***Remember*** – One write capacity unit represents one write per second for items up to 1 KB in size.

Pricing



- DynamoDB simply asks you to specify the target utilization rate and minimum to maximum capacity that you want for your table. DynamoDB handles the provisioning of resources to achieve your target utilization of read and write capacity, then auto scales your capacity based on usage.
- Optionally you can directly specify read and write capacity if you prefer manual.
 - Provisioned Throughput(Write) – One write capacity unit(WCU) provides up to one write per second, enough for 2.5 million writes per month. As low as \$0.47 per WCU
 - Provisioned Throughput(Read) – One read capacity unit (RCU) provides up to two reads per second, enough for 5.2 million reads per month as low as \$0.09 per RCU
 - Indexed Data Storage – DynamoDB charges an hourly rate per GB of disk space that your table consumes table throughput

Provisioning Capacity



Example

Read Capacity Requirements

- If you have a table and you want to read 100 items per second with strongly consistent reads and your items are 8KB in size, you would calculate the required provisioned capacity as follows;
- $8\text{KB}/4\text{KB} = 2$ capacity units
- 2 read capacity units per item \times 100 reads per second $= 200$ read capacity units
- **Note:** Eventual Consistent Reads would require $200/2 = 100$ read capacity units

Write Capacity Requirements

- If you have a table and you want to write 50 items per second and your items are 4KB in size, you would calculate the required provisioned capacity as follows;
- $4\text{KB}/1\text{KB} = 4$ capacity units
- 4 write capacity units per item \times 50 writes per second $= 200$ write capacity units

Provisioning Capacity



Example Read Capacity Requirements

- 10 strongly consistent reads per seconds of 4 KB each
We need $10 * 4KB / 4KB = 10$ RCU (For SCR, 1 RCU= 1 read per second for 4KB)
- 16 Eventually consistent reads per seconds of 12 KB each
We need $16/2 * 12KB / 4KB = 24$ RCU (For ECR, 1 RCU = 2 reads per second for 4KB size)
- 10 Strongly consistent reads per seconds of 6 KB each
We need $10 * 8KB / 4KB = 20$ RCU (For ECR, 1 RCU = 2 reads per second for 4KB size, 6 round off to 8KB because each is 4KB and next size is 8KB)

Searching / Scan Items



- You can use a Query or a Scan to search for items in a DynamoDB table.
- Queries are primary search operations to help you search for items in a table or a secondary index using the primary key attribute values.
- You need to provide partition key name and value to search for and you can also provide a sort key name and value and use a comparison operator to refine the search results.

Amazon DynamoDB Streams

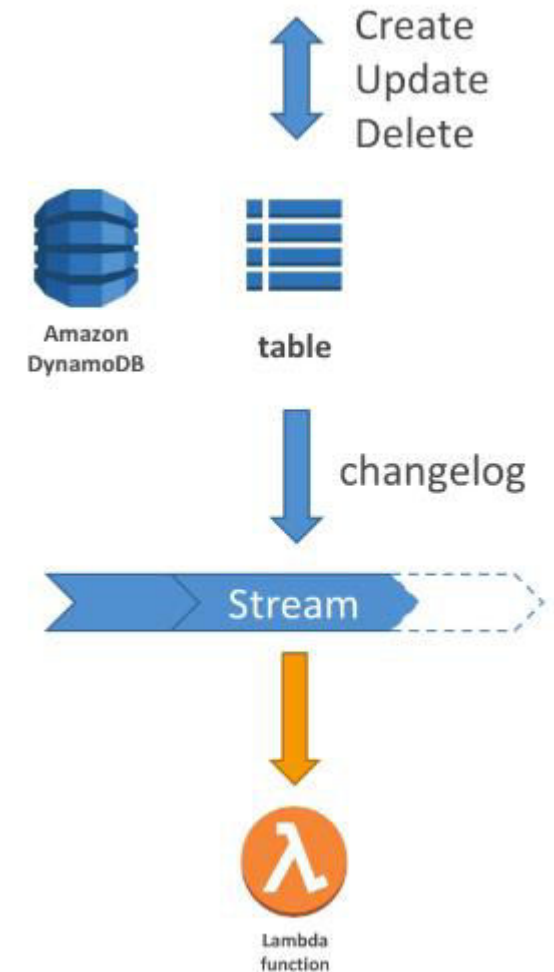


- Applications can be designed to keep track of recent changes and they perform some action on the changed record sets.
- This method of streaming data is a feature available on DynamoDB and enables you to get a list of item changes for a **24-hour** period.
- The stream is essentially ordered flow of information about changes to items in an Amazon DynamoDB table. Once enabled, DynamoDB captures information about every modification to data items in the table.

Amazon DynamoDB Streams



- Changes in DynamoDB (Create, Update, Delete) can end up in a DynamoDB Stream
- This Stream can be read by AWS Lambda, and we can do
 - React to changes in real time(welcome emails).
 - Analytics
 - Insert into ElasticSearch and lot more
- Cloud Implement cross region replication using streams
- Stream has 24 hours of data retention



Amazon DynamoDB TTL



- TTL = automatically delete an item after an expiry date / time
- TTL is provided at no extra cost, deletions do not use WCU /RCU
- TTL is a background task operated by the DynamoDB service itself
- Helps reduce storage and manage the table size over time
- TTL is enabled per row (you define a TTL column, and add a date there)
- DynamoDB typically deletes expired items within 48 hours of expiration
- Deleted items due to TTL are also deleted in GSI / LSI
- DynamoDB streams can help to recover expired items

DynamoDB API's – Writing Data



- **PutItem** : Write data to DynamoDB(Create data or full replace). It consumes WCU.
- **UpdateItem** : Update data in DynamoDB (partial update of attributes).
- **Conditional Writes**: Accept a write/update only if conditions are respected otherwise reject
 - Helps with concurrent access to items
 - No performance impact

DynamoDB API's – Deleting Data



- **DeleteItem:**
 - Delete an individual row
 - Ability to perform a conditional delete
- **DeleteTable**
 - Delete a whole table and all its items
 - Much quicker deletion than calling DeleteItem on all items

DynamoDB API's – Batching Writes



- **BatchWriteItem:**
 - Up to 25 PutItem and / or DeleteItem in one call
 - Up to 16 MB of data written
 - Up to 400 KB of data per item
- Batching allows you to save in latency by reducing the number of calls done against DynamoDB:
- Operations are done in parallel for better efficiency
- Its possible for part of a batch to fail, in which case we have to try the failed item(using exponential back-off algorithm)

DynamoDB API's – Reading Data



- **GetItem:**
 - Read based on Primary Key
 - Primary key = HASH or HASH-RANGE
 - Eventually consistent read by default
 - Option to use strongly consistent reads
- **BatchGetItem**
 - Up to 100 items
 - Up to 16 MB of data
 - Items are retrieved in parallel to minimize latency

Features



- Amazon DynamoDB Accelerator (DAX) is a fully managed, highly available, in-memory cache that can reduce Amazon DynamoDB response times from milliseconds to microseconds, even at millions of request per second.
- Document Data Model Support- DynamoDB supports storing, querying and updating documents
- Key-Value Data Model Support
- Seamless scaling
- High availability
- Cross-region replication is now supported
- Session Management * or Storing Session data

You can do the following on Dynamodb



- Monitor recent alerts, total capacity, service health, and the latest DynamoDB news on the DynamoDB dashboard.
- Create, update, and delete tables. The capacity calculator provides estimates of how many capacity units to request based on the usage information you provide.
- Manage streams.
- View, add, update, and delete items that are stored in tables. Manage Time To Live (TTL) to define when items in a table expire so that they can be automatically deleted from the database.
- Query and scan a table.
- Set up and view alarms to monitor your table's capacity usage. View your table's top monitoring metrics on real-time graphs from CloudWatch.
- Modify a table's provisioned capacity.
- Create and delete global secondary indexes.
- Create triggers to connect DynamoDB streams to AWS Lambda functions.
- Apply tags to your resources to help organize and identify them.
- Purchase reserved capacity.

Table Overview



- **Overview** – View stream and table details, and manage streams and Time To Live (TTL).
- **Items** – Manage items and perform queries and scans.
- **Metrics** – Monitor CloudWatch metrics.
- **Alarms** – Manage CloudWatch alarms.
- **Capacity** – Modify a table's provisioned capacity.
- **Indexes** – Manage global secondary indexes.
- **Global tables** – Cross region replication
- **Triggers** – Manage triggers to connect DynamoDB streams to Lambda functions.
- **Access control** – Set up fine-grained access control with web identity federation.
- **Tags** – Apply tags to your resources to help organize and identify them.