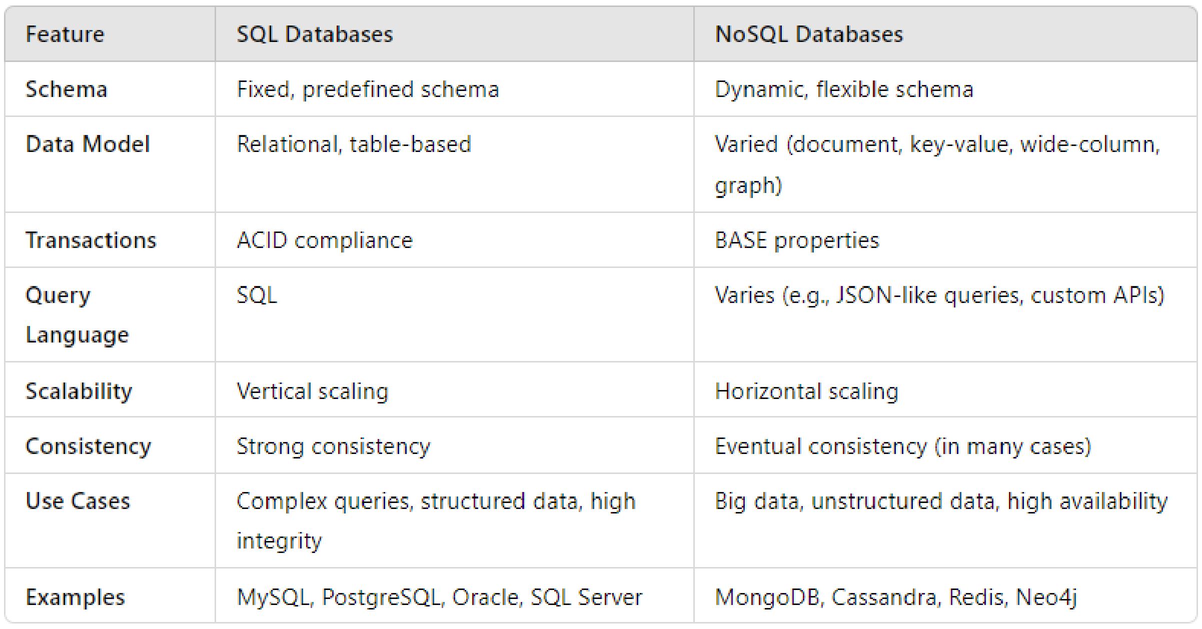
AWS DynamoDB

Before we learn about DynamoDB, we need to understand the difference between SQL and NoSQL Databases

SQL (Structured Query Language) and NoSQL (***Not Only or Non*** SQL) databases serve different needs and use cases.



Feature	SQL Databases	NoSQL Databases
Schema	Fixed, predefined schema	Dynamic, flexible schema
Data Model	Relational, table-based	Varied (document, key-value, wide-column, graph)
Transactions	ACID compliance	BASE properties
Query Language	SQL	Varies (e.g., JSON-like queries, custom APIs)
Scalability	Vertical scaling	Horizontal scaling
Consistency	Strong consistency	Eventual consistency (in many cases)
Use Cases	Complex queries, structured data, high integrity	Big data, unstructured data, high availability
Examples	MySQL, PostgreSQL, Oracle, SQL Server	MongoDB, Cassandra, Redis, Neo4j

Choose SQL if:

- You need complex queries and transactions.
- Data integrity and ACID compliance are critical.
- Your data structure is well-defined and unlikely to change.

ACID (Atomicity, Consistency, Isolation, Durability)

Choose NoSQL if:

- You need to handle large volumes of unstructured or semi-structured data.
- Your application requires high availability and horizontal scaling.
- The data model is flexible or likely to evolve over time.
- You are building distributed systems with low-latency requirements.

BASE (Basically Available, Soft state, Eventual consistency)

Both SQL and NoSQL databases have their strengths and are suitable for different types of applications. The choice depends on your specific use case, data requirements, and scalability needs.

NoSQL Databases

- They are non-relational databases and are distributed
- They do not support the query joins
- All the data that is needed for a query is present in a row
- They don't perform aggregation such as SUM, AVG
- Scaled horizontally

Now we know the SQL and NoSQL Differences ...! Let's get into our Dynamo DB

Amazon Dynamo DB

- Fully Managed, highly available with replication across multiple AZs
- NoSQL database – not a relational database
- Scales to massive workloads, distributed database
- Millions of requests per second, trillions for row, 100s of TB Storage
- Fast & Consistent in performance
- Integrated with IAM for security, authorization and administration
- Enables event driven programming with DynamoDB Streams
- Low cost and auto scaling capabilities
- Standard & Infrequent Access (IA) Table Class

How it looks like

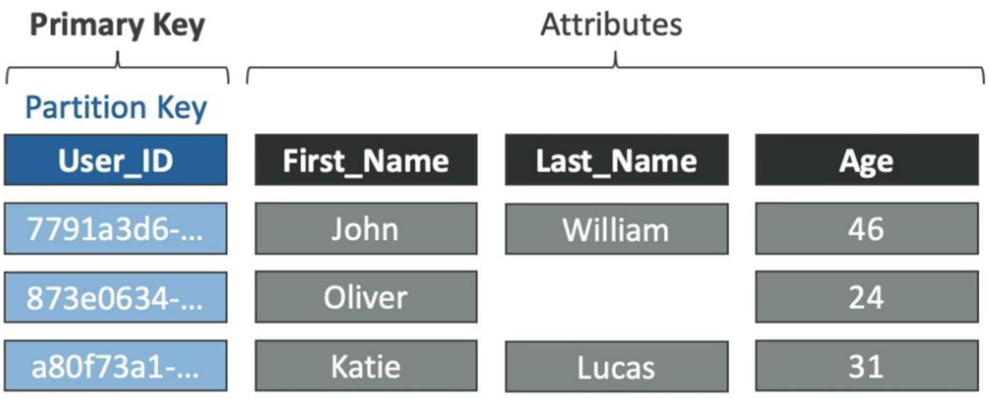
- DynamoDB is made of Tables
- Each table has a Primary Key (must be added at the creation time)
- Each Table can have an infinite number of items (We can call it as Rows)
- Each item has attributes (Can be added over time – can be null)
- Maximum size of ***an item is 400 KB***
- Data types supported

  1) **Scalar Types** – String, Number, Binary, Boolean & Null
  2) **Document Types** – List, Map
  3) **Set Types** - String set, Number set and Binary set

Primary Keys

Choosing one: Partition Key (HASH)

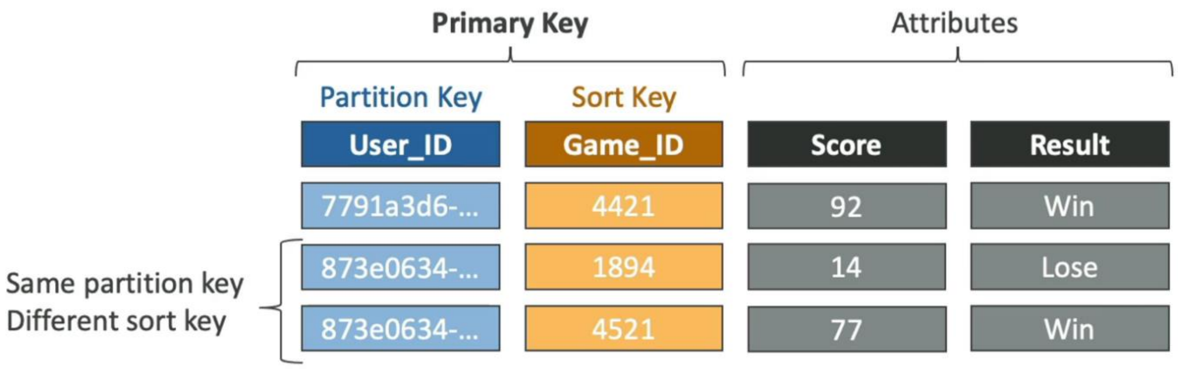
- Partition key must be unique for each item
- Partition Key must be diverse so that data is distributed
- Example: Student_ID for the student's table



Primary Key	Attributes		
Partition Key			
User_ID	First_Name	Last_Name	Age
7791a3d6-...	John	William	46
873e0634-...	Oliver		24
a80f73a1-...	Katie	Lucas	31

Choosing Two: Partition Key + Sort Key (HASH+RANGE)

- The combination must be unique for each item
- Data is grouped by partition key
- Example: students and sports, Student_ID for the partition key and sport_ID for the sort key



	Primary Key		Attributes	
	Partition Key	Sort Key		
	User_ID	Game_ID	Score	Result
	7791a3d6-...	4421	92	Win
Same partition key Different sort key	873e0634-...	1894	14	Lose
	873e0634-...	4521	77	Win

Partition key should be a column in the table that has highest cardinality to maximize the number of partitions (data distribution). Partition key should also be highly diverse to ensure that the data is distributed equally across partitions.

If the partition (hash) key is not highly diverse (only few unique values), add a suffix to the partition key to make the partition key diverse. The suffix can be generated either randomly or calculated using a hashing algorithm.

As a basic fundamental – now we know how DynamoDB looks like so, now we need to understand, what size we need and all other things related to the Dynamo DB

We need to learn few terms before we step into the Introvert Path of the DynamoDB

Capacity Planning

Capacity planning in Amazon DynamoDB refers to the process of estimating and provisioning the read and write capacity units (RCUs and WCUs, respectively) that your DynamoDB tables will require to handle the expected workload efficiently.

Throughput

In Amazon DynamoDB, throughput refers to the measure of the amount of read and write activity that a DynamoDB table can handle per unit of time. Throughput capacity in DynamoDB is provisioned and measured in terms of Read Capacity Units (RCUs) and Write Capacity Units (WCUs).

DynamoDB – Read/Write Capacity Modes

Control how you manage your table's capacity (read/write throughput)

Provisioned Mode

- You specify the number of reads/writes per second
- You need to plan capacity beforehand
- Pay for provisioned read & write capacity units

On-Demand Mode

- Read/Writes automatically scales up/down with your workloads
- No Capacity planning needed
- Pay for what you use, more & more expensive

😉***We can switch between modes once every 24 hours***

Read/Write Capacity Modes – Provisioned

- Table must have provisioned read and write capacity units
- **Read Capacity Units (RCU) – throughput for reads**
- **Write Capacity Reads (WCU) – throughput for writes**
- Option to setup auto-scaling of throughput to meet demand
- Throughput can be exceeded temporarily using the burst capacity
- If Burst Capacity has been consumed, we will get a "ProvisionedThroughputExceededException"
- It's then advised to do an exceptional backoff retry

Before we get into Read and Write Capacity Units calculation, we might need to learn about few terms here – those are

Eventually Consistent Read: (Default)

An eventually consistent read may not reflect the latest write data but will eventually converge to the most up-to-date data. DynamoDB reaches consistency across all copies of data within typically a second.

- Low latency
- May get stale data (if the replication hasn't happened yet)

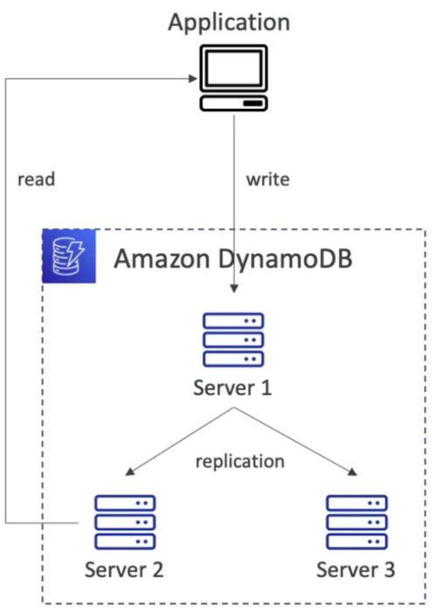
Strongly Consistent Read:

A strongly consistent read returns the most up-to-date data, reflecting all writes that were completed before the read.

- More latency
- Set "ConsistentRead" Parameter to True in API Calls

Transactional Read:

Transactional reads are part of DynamoDB transactions, ensuring that the read operations within a transaction see a consistent snapshot of the data.



Standard Write: (Default)

Standard write operations are atomic, meaning each write either fully succeeds or fails. However, they do not offer cross-item or cross-table atomicity guarantees.

Transactional Write:

Transactional write operations ensure all operations within a transaction (including writes across multiple items or tables) either fully succeed or fail together. This provides ACID (Atomicity, Consistency, Isolation, Durability) guarantees.

Read Capacity Units (RCUs)

One RCU represents the read throughput capacity for strongly consistent reads of items up to 4 KB in size, or two eventually consistent reads per second.

General Formula:

- **Eventually Consistent** Read: ***1 RCU = 2 reads per second*** for items up to 4 KB.
- **Strongly Consistent** Read: ***1 RCU = 1 read per second*** for items up to 4 KB.
- **Transactional** Read: ***1 RCU = 0.5 read per second*** for items up to 4 KB.

For items larger than 4 KB, RCUs are calculated by rounding up the item size to the nearest 4 KB block.

$$\text{RCUs} = \left\{ \frac{\textbf{Item Size}}{\textbf{4 KB}} \right\} \times \left\{ \frac{\textbf{Read Request Rate}}{\textbf{Consistency factor}} \right\}$$

- Consistency Factor = **2** for Eventually Consistent Reads
- Consistency Factor = **1** for Strongly Consistent and
- Consistency Factor = **0.5** for Transactional Reads

Example:

If you need to read 80 items per second, each 6 KB in size, using eventually consistent reads:

- Item size = 6 KB
- Read request rate = 80 reads/second
- Consistency factor = 2

$$\text{RCUs} = \left(\frac{6\,\text{KB}}{4\,\text{KB}}\right) \times \left(\frac{80}{2}\right) = 2 \times 40 = 80\ \text{RCUs}$$

Write Capacity Units (WCU)

One WCU represents the write throughput capacity for writes of items up to 1 KB in size.

General Formula:

- **Standard** Write: **1 WCU = 1 write** per second for items up to 1 KB.
- **Transactional** Write: **1 WCU = 0.5** write per second for items up to 1 KB.

For items larger than 1 KB, WCUs are calculated by rounding up the item size to the nearest 1 KB block.

- Consistency Factor = ***1*** for Standard Writes
- Consistency Factor = ***0.5*** for Transactional Writes

$$\text{WCUs} = \left\{ \frac{\textbf{Item Size}}{\textbf{1 KB}} \right\} \times \left\{ \frac{\textbf{Write Request Rate}}{\textbf{Consistency factor}} \right\}$$

Example:

If you need to write 50 items per second, each 2.5 KB in size:

- Item size = 2.5 KB
- Write request rate = 50 writes/second

$$\text{WCUs} = \left(\frac{2.5\ \text{KB}}{1\ \text{KB}}\right) \times 50 = 3 \times 50 = 150\ \text{WCUs}$$

Batches: When performing batch operations, sum the total size of the items in the batch and then apply the formula.

Indexes: Each secondary index will consume additional RCUs and WCUs based on the size and access patterns of the indexed attributes.

Just to practice

1. **Reading 100 items per second, each 8 KB, with strong consistency:**

   Item size = 8 KB

   $$\text{RCUs} = \left(\frac{8\text{ KB}}{4\text{ KB}}\right) \times 100 = 2 \times 100 = 200\text{ RCUs}$$

2. **Writing 200 items per second, each 0.5 KB:**

   Item size = 0.5 KB

   $$\text{WCUs} = \left(\frac{0.5\text{ KB}}{1\text{ KB}}\right) \times 200 = 1 \times 200 = 200\text{ WCUs}$$

3. **Reading 150 items per second, each 12 KB, with eventual consistency:**

   Item size = 12 KB

   $$\text{RCUs} = \left(\frac{12\text{ KB}}{4\text{ KB}}\right) \times \left(\frac{150}{2}\right) = 3 \times 75 = 225\text{ RCUs}$$