DynamoDb

NoSQL databases

- NoSQL databases are non-relational databases and are **distributed**
- NoSQL databases include MongoDB, DynamoDB, …
- NoSQL databases do not support query joins (or just limited support)
- All the data that is needed for a query is present in one row
- NoSQL databases don't perform aggregations such as "SUM", "AVG", …
- **NoSQL databases scale horizontally**

- There's no "right or wrong" for NoSQL vs SQL, they just require to model the data differently and think about user queries differently

Amazon DynamoDB

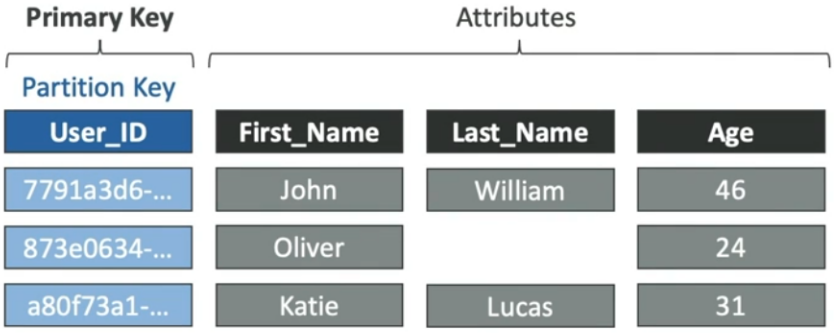
- Fully managed, highly available with replication across multiple AZs
- NoSQL database - not a relational database
- Scales to massive workloads, distributed database
- Millions of requests per seconds, trillions of row, 100s of TB of storage
- Fast and consistent in performance (low latency on retrieval)
- 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

DynamoDB - Basics

- 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 an item is **400KB**
- Data types supported are:
  - **Scalar Types** – String, Number, Binary, Boolean, Null
  - **Document Types** – List, Map
  - **Set Types** – String Set, Number Set, Binary Set

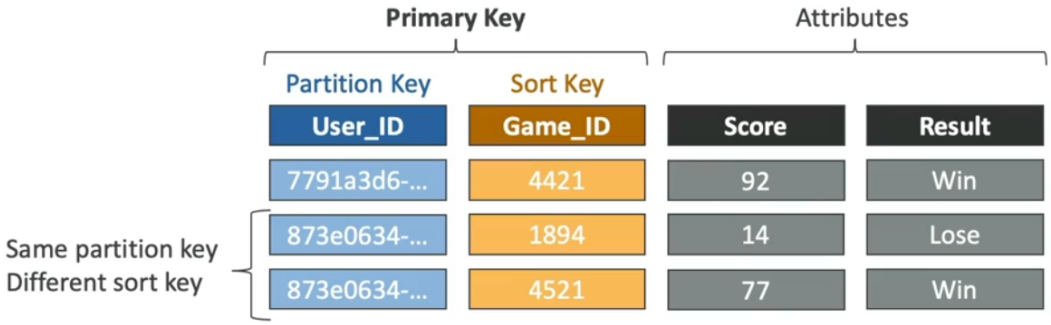
DynamoDB – Primary Keys

- Option 1: Partition Key (HASH)
  - Partition key must be unique for each item
  - Partition key must be "diverse" so that the data is distributed
  - Example: "User_ID" for a users table

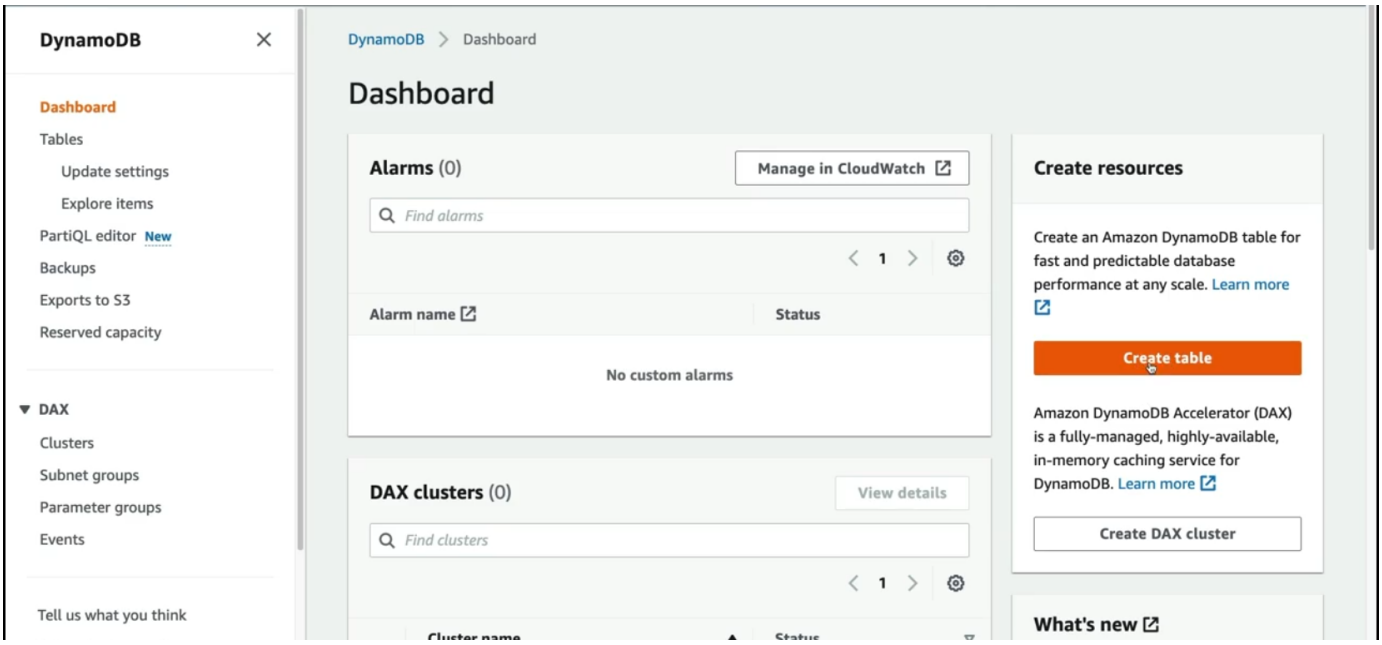


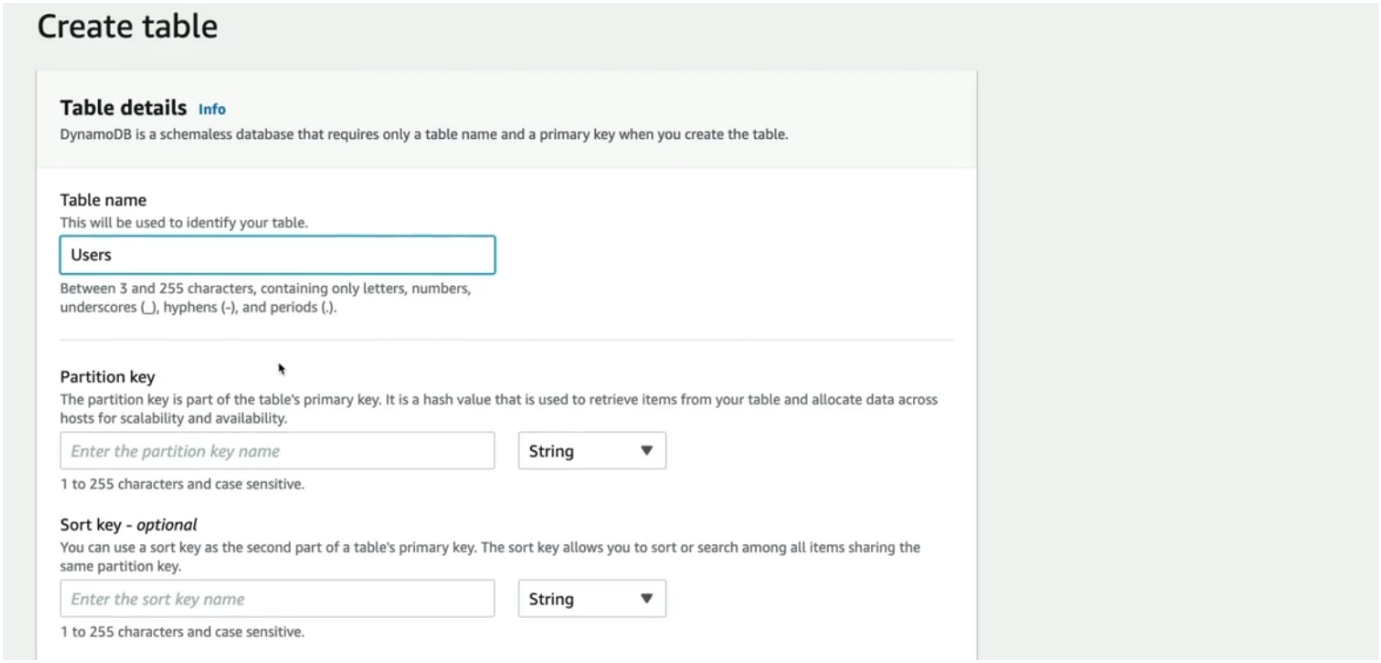
DynamoDB – Primary Keys

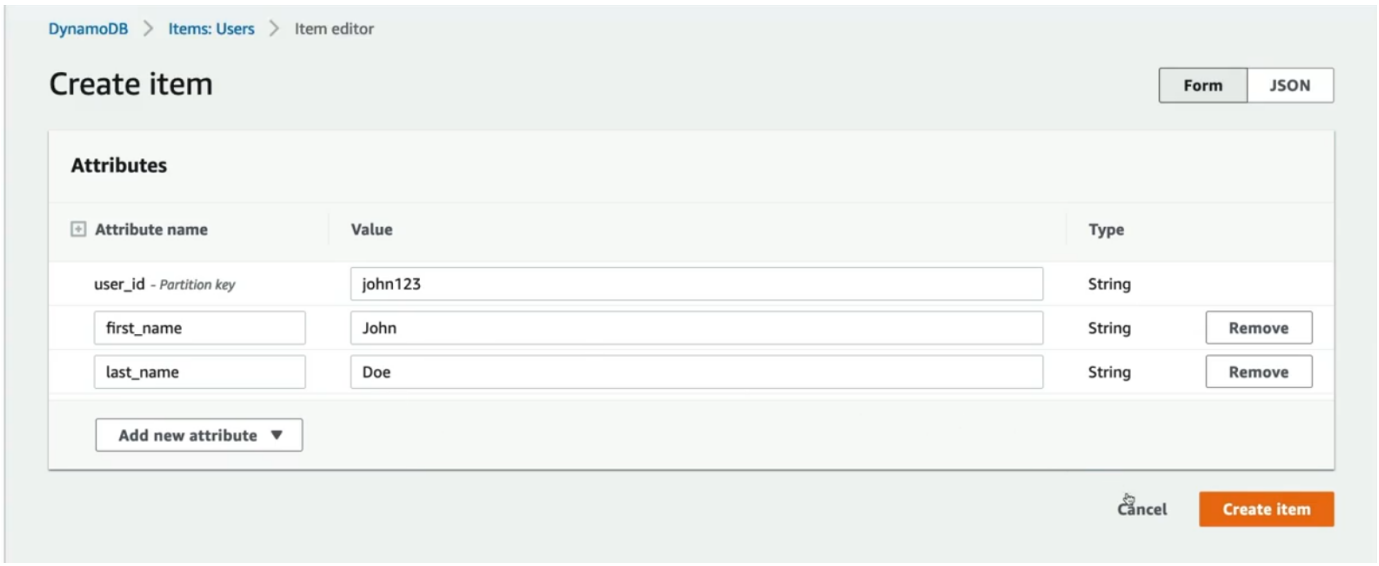
- Option 2: Partition Key + Sort Key (HASH + RANGE)
  - The combination must be unique for each item
  - Data is grouped by partition key
  - Example: users-games table, "User_ID" for Partition Key and "Game_ID" for Sort Key



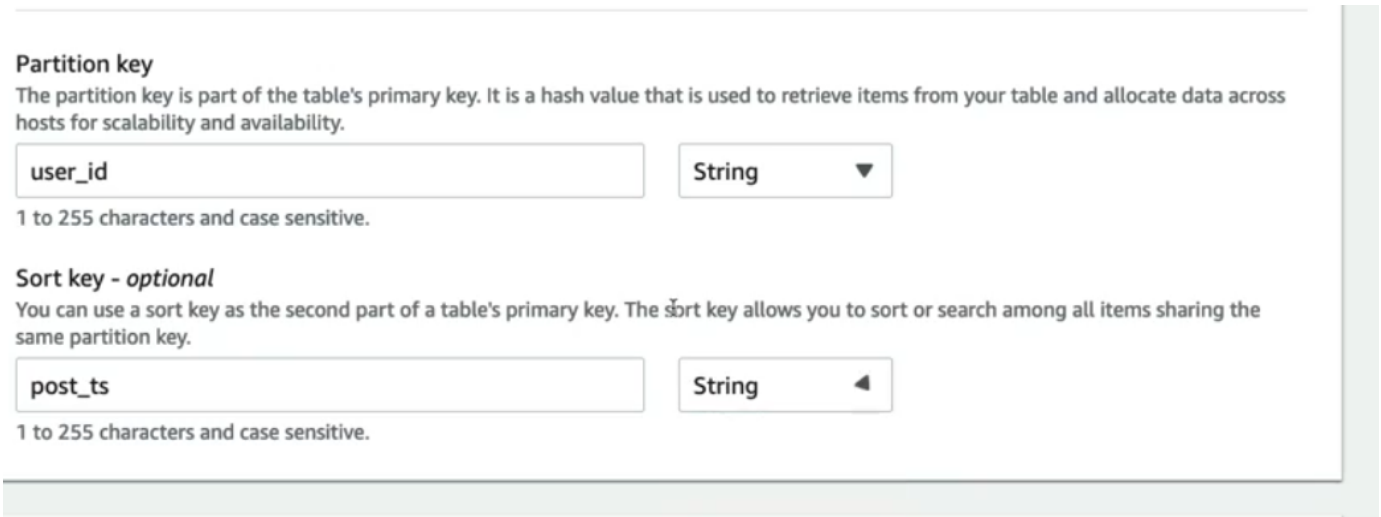
hands on dynamo db

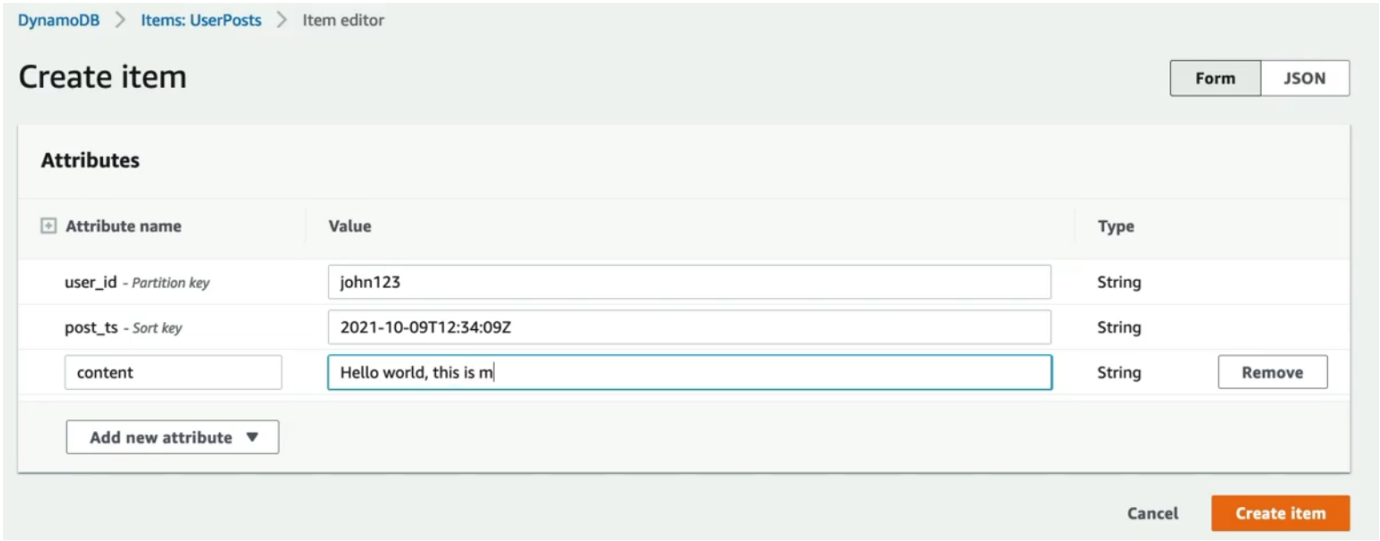


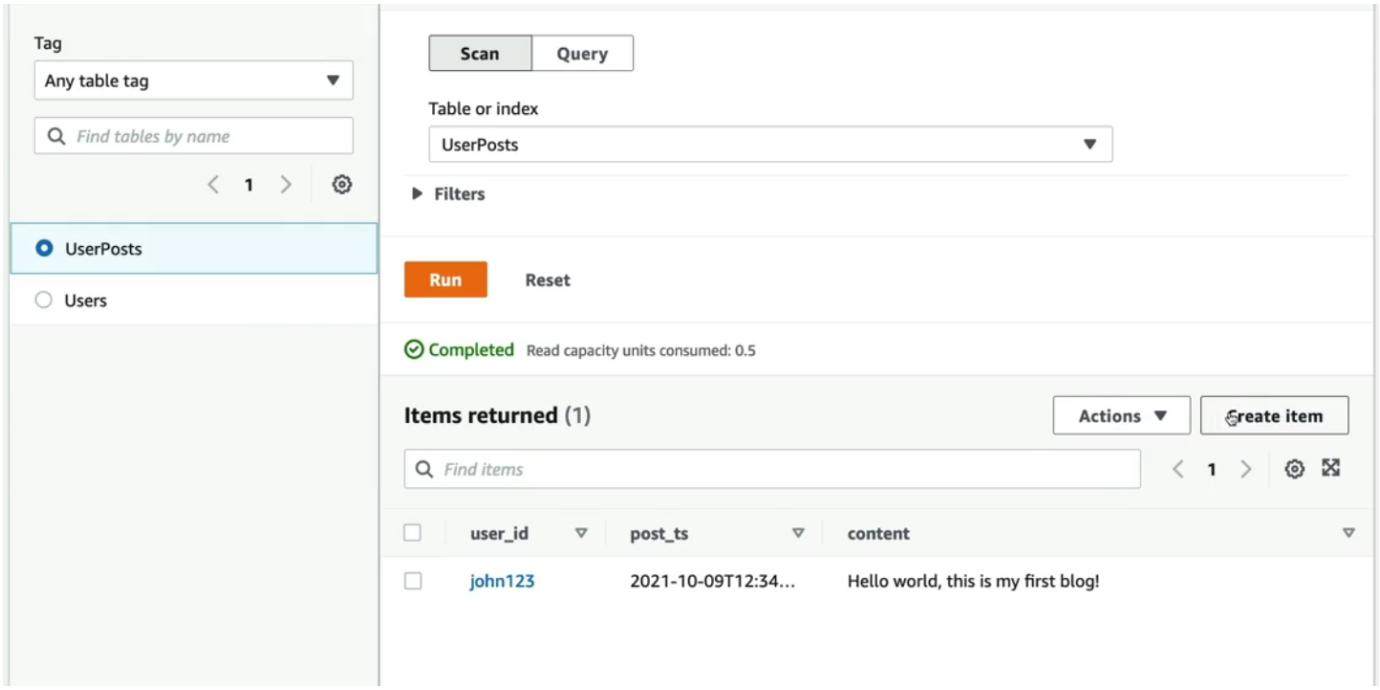




you can add short key better way to search most be unique







where we use dynamoDB

DynamoDB in Big Data

- Common use cases include:
  - Mobile apps
  - Gaming
  - Digital ad serving
  - Live voting
  - Audience interaction for live events
  - Sensor networks
  - Log ingestion
  - Access control for web-based content
  - Metadata storage for Amazon S3 objects
  - E-commerce shopping carts
  - Web session management
- Anti Pattern
  - Prewritten application tied to a traditional relational database: use RDS instead
  - Joins or complex transactions
  - Binary Large Object (BLOB) data: store data in S3 & metadata in DynamoDB
  - Large data with low I/O rate: use S3 instead

DynamoDB – Read/Write Capacity Modes

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

- Provisioned Mode (default)
  - 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 scale up/down with your workloads
  - No capacity planning needed
  - Pay for what you use, more expensive ($$$)

- You can switch between different modes once every 24 hours

DynamoDB – Read/Write Capacity Modes

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

- Provisioned Mode (default)
  - 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 scale up/down with your workloads
  - No capacity planning needed
  - Pay for what you use, more expensive ($$$)

- You can switch between different modes once every 24 hours

R/W Capacity Modes – Provisioned

- Table must have provisioned read and write capacity units
- **Read Capacity Units (RCU)** – throughput for reads
- **Write Capacity Units (WCU)** – throughput for writes
- Option to setup auto-scaling of throughput to meet demand
- Throughput can be exceeded temporarily using **"Burst Capacity"**
- If Burst Capacity has been consumed, you'll get a **"ProvisionedThroughputExceededException"**

DynamoDB – Write Capacity Units (WCU)

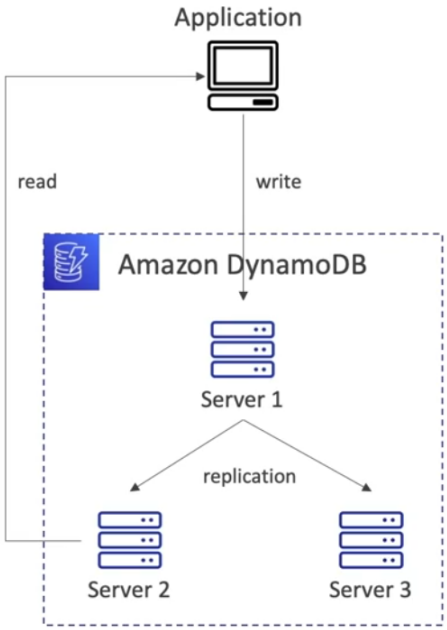
- One *Write Capacity Unit (WCU)* represents one write per second for an item up to **1 KB** in size
- If the items are larger than 1 KB, more WCUs are consumed

- **Example 1:** we write 10 items per second, with item size 2 KB
  - We need $\mathbf{10 * (\frac{2\ KB}{1\ KB}) = 20\ WCUs}$
- **Example 2:** we write 6 items per second, with item size 4.5 KB
  - We need $\mathbf{6 * (\frac{5\ KB}{1\ KB}) = 30\ WCUs}$ (4.5 gets rounded to the upper KB)

Strongly Consistent Read vs. Eventually Consistent Read

- Eventually Consistent Read (default)
  - If we read just after a write, it's possible we'll get some stale data because of replication

- Strongly Consistent Read
  - If we read just after a write, we will get the correct data
  - Set "ConsistentRead" parameter to True in API calls (GetItem, BatchGetItem, Query, Scan)
  - Consumes twice the RCU



DynamoDB – Read Capacity Units (RCU)

- One *Read Capacity Unit (RCU)* represents **one Strongly Consistent Read** per second, or **two Eventually Consistent Reads** per second, for an item up to **4 KB** in size
- If the items are larger than 4 KB, more RCUs are consumed

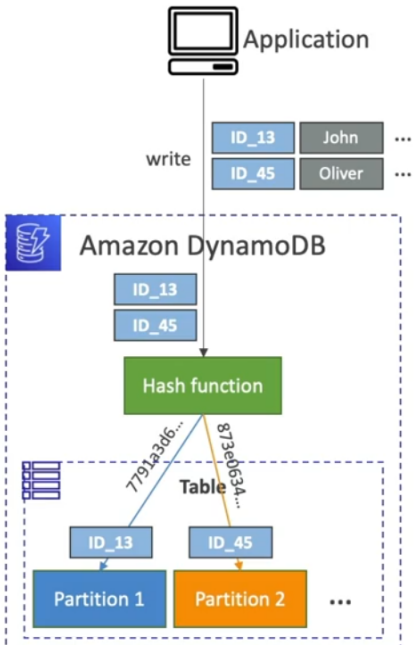
- **Example 1:** 10 Strongly Consistent Reads per second, with item size 4 KB
  - We need $\mathbf{10} * \left(\frac{4\,KB}{4\,KB}\right) = \mathbf{10}\ \boldsymbol{RCUs}$
- **Example 2:** 16 Eventually Consistent Reads per second, with item size 12 KB
  - We need $\left(\frac{16}{2}\right) * \left(\frac{12\,KB}{4\,KB}\right) = \mathbf{24}\ \boldsymbol{RCUs}$
- **Example 3:** 10 Strongly Consistent Reads per second, with item size 6 KB
  - We need $\mathbf{10} * \left(\frac{8\,KB}{4\,KB}\right) = \mathbf{20}\ \boldsymbol{RCUs}$ (we must round up 6 KB to 8 KB)

DynamoDB – Partitions Internal

- Data is stored in partitions
- Partition Keys go through a hashing algorithm to know to which partition they go to

- To compute the number of partitions:
  - $\#\ of\ partitions_{by\ capacity} = \left(\frac{RCUs_{Total}}{3000}\right) + \left(\frac{WCUs_{Total}}{1000}\right)$
  - $\#\ of\ partitions_{by\ size} = \frac{Total\ Size}{10\ GB}$
  - $\#\ of\ partitions =$ $\text{ceil}(\max(\#\ of\ partitions_{by\ capacity}, \#\ of\ partitions_{by\ size}))$

- WCUs and RCUs are spread evenly across partitions

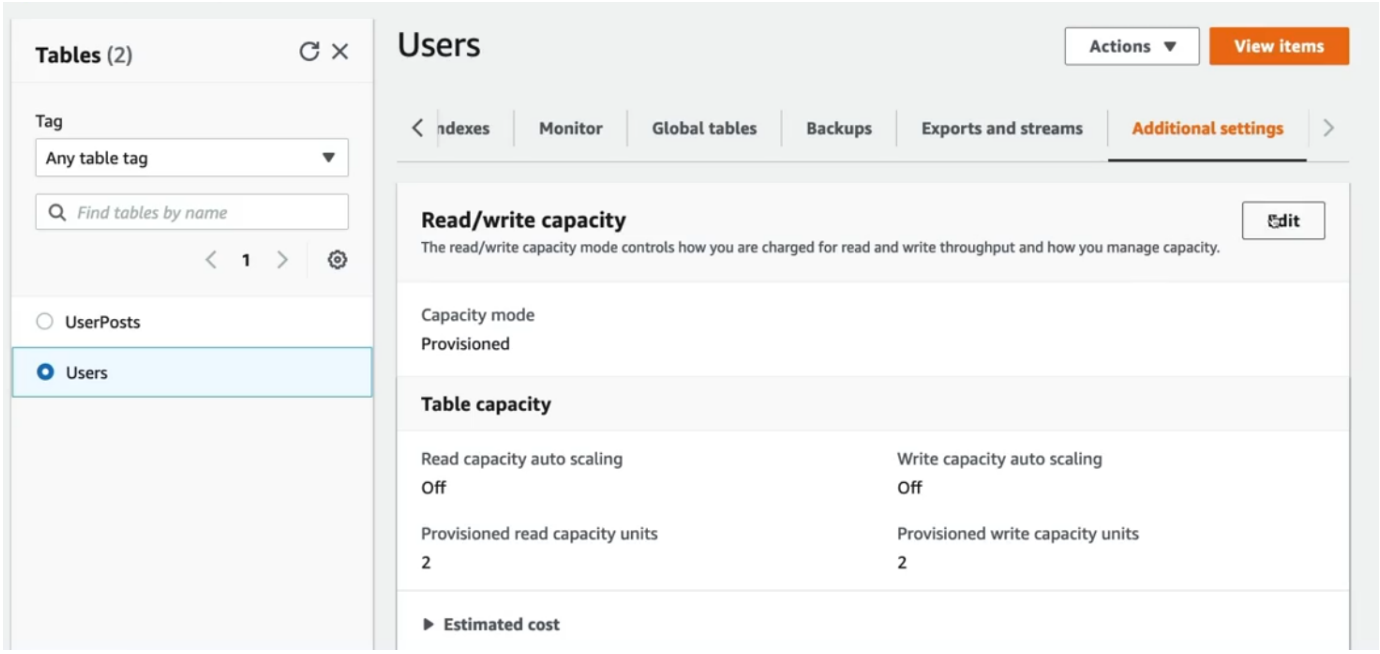


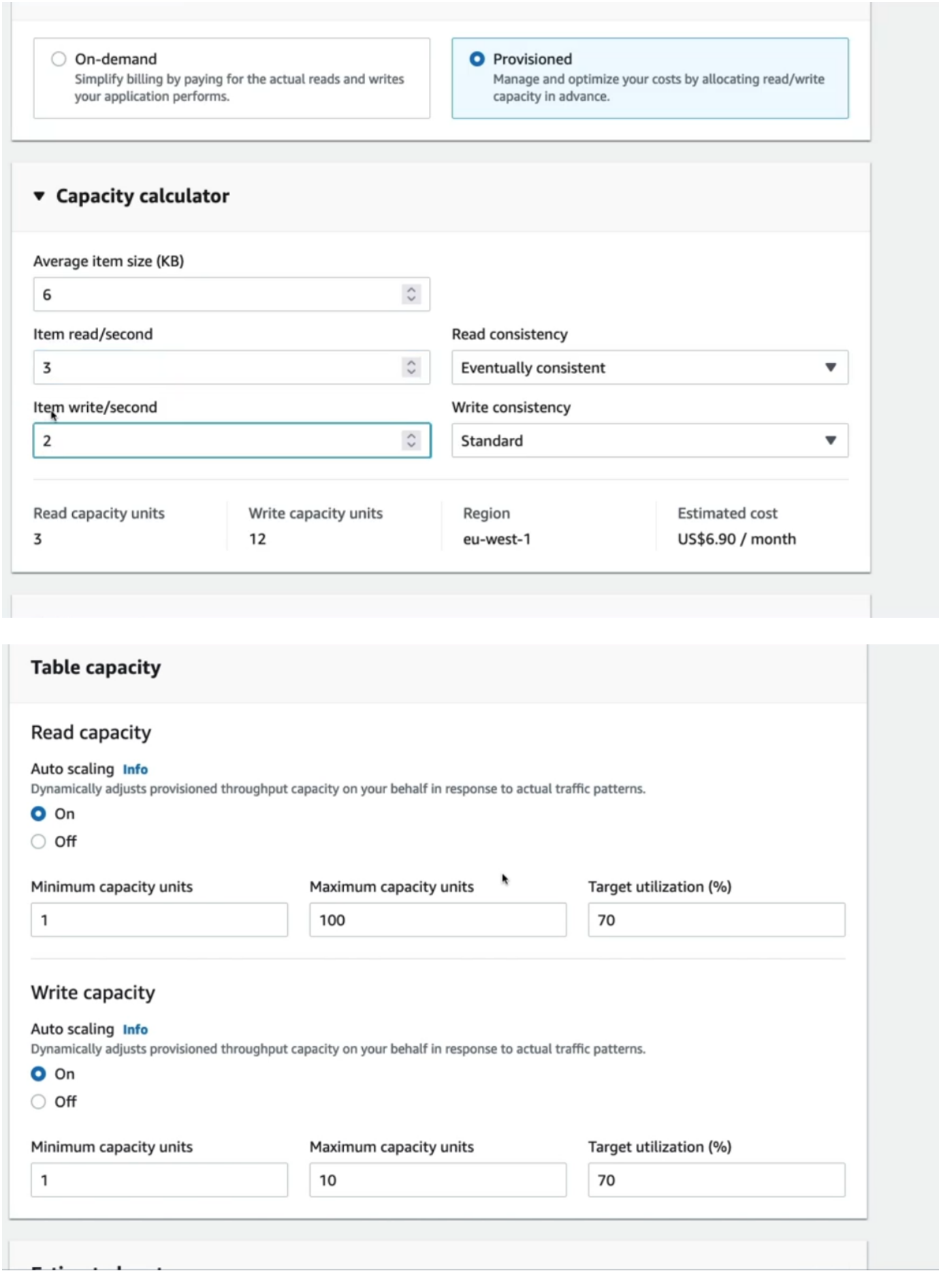
DynamoDB – Throttling

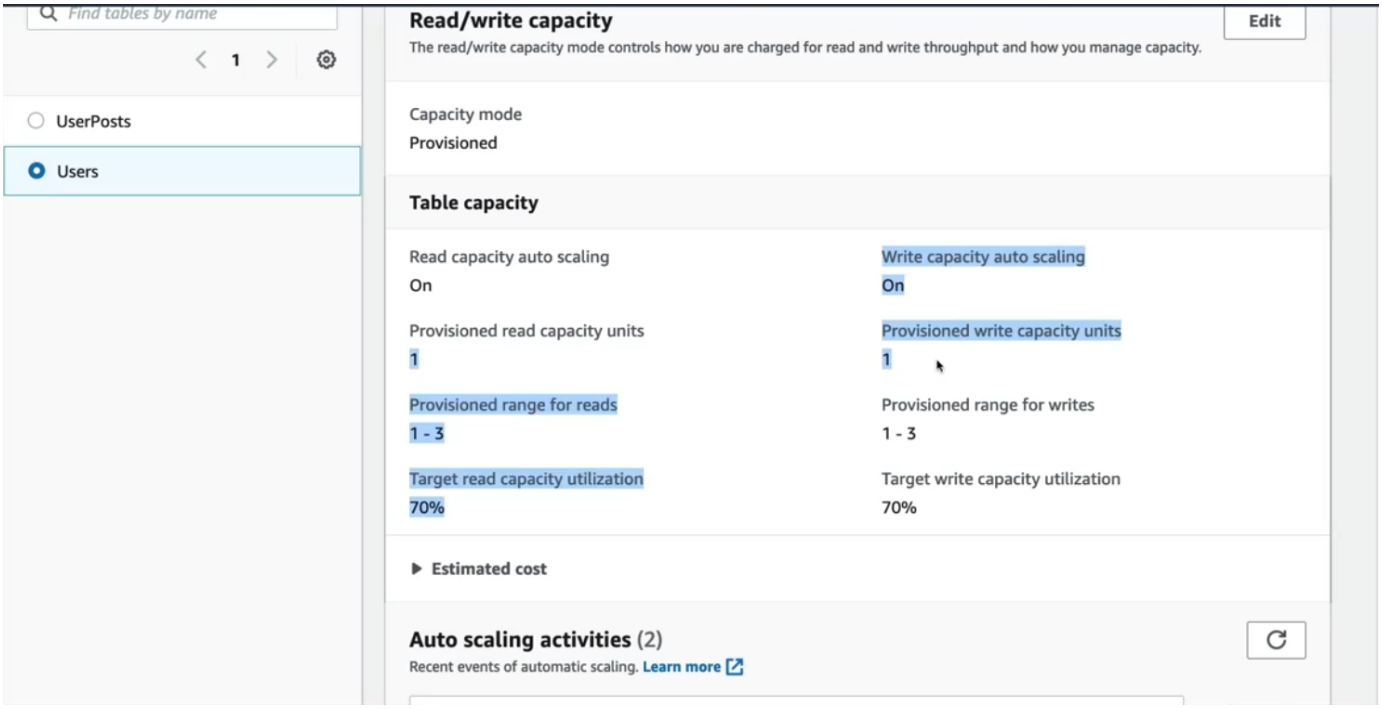
- If we exceed provisioned RCUs or WCUs, we get "ProvisionedThroughputExceededException"
- Reasons:
  - **Hot Keys** – one partition key is being read too many times (e.g., popular item)
  - **Hot Partitions**
  - **Very large items,** remember RCU and WCU depends on size of items
- Solutions:
  - **Exponential backoff** when exception is encountered (already in SDK)
  - **Distribute partition keys** as much as possible
  - If RCU issue, we can **use DynamoDB Accelerator (DAX)**

R/W Capacity Modes – On-Demand

- Read/writes automatically scale up/down with your workloads
- No capacity planning needed (WCU / RCU)
- Unlimited WCU & RCU, no throttle, more expensive
- You're charged for reads/writes that you use in terms of RRU and WRU
- Read Request Units (RRU) – throughput for reads (same as RCU)
- Write Request Units (WRU) – throughput for writes (same as WCU)
- 2.5x more expensive than provisioned capacity (use with care)
- Use cases: unknown workloads, unpredictable application traffic, …




On-demand
Simplify billing by paying for the actual reads and writes your application performs.
Provisioned
Manage and optimize your costs by allocating read/write capacity in advance.
Capacity calculator
Average item size (KB)
6
Item read/second
3
Read consistency
Eventually consistent
Item write/second
2
Write consistency
Standard
Read capacity units
3
Write capacity units
12
Region
eu-west-1
Estimated cost
US$6.90 / month
Table capacity
Read capacity
Auto scaling Info
Dynamically adjusts provisioned throughput capacity on your behalf in response to actual traffic patterns.
On
Off
Minimum capacity units
1
Maximum capacity units
100
Target utilization (%)
70
Write capacity
Auto scaling Info
Dynamically adjusts provisioned throughput capacity on your behalf in response to actual traffic patterns.
On
Off
Minimum capacity units
1
Maximum capacity units
10
Target utilization (%)
70



DynamoDB – Writing Data

- PutItem
  - Creates a new item or fully replace an old item (same Primary Key)
  - Consumes WCUs
- UpdateItem
  - Edits an existing item's attributes or adds a new item if it doesn't exist
  - Can be used to implement **Atomic Counters** – a numeric attribute that's unconditionally incremented
- Conditional Writes
  - Accept a write/update/delete only if conditions are met, otherwise returns an error

DynamoDB – Reading Data

- GetItem
  - Read based on Primary key
  - Primary Key can be **HASH** or **HASH+RANGE**
  - Eventually Consistent Read (default)
  - Option to use Strongly Consistent Reads (more RCU - might take longer)
  - **ProjectionExpression** can be specified to retrieve only certain attributes

DynamoDB – Reading Data (Query)

- **Query** returns items based on:
  - **KeyConditionExpression**
    - Partition Key value (**must be = operator**) – required
    - Sort Key value (=, <, <=, >, >=, Between, Begins with) – optional
  - **FilterExpression**
    - Additional filtering after the Query operation (before data returned to you)
    - Use only with non-key attributes (does not allow HASH or RANGE attributes)
- Returns:
  - The number of items specified in **Limit**
  - Or up to 1 MB of data
- Ability to do pagination on the results
- Can query table, a Local Secondary Index, or a Global Secondary Index

DynamoDB – Reading Data (Scan)

- **Scan** the entire table and then filter out data (inefficient)
- Returns up to 1 MB of data – use pagination to keep on reading
- Consumes a lot of RCU
- Limit impact using **Limit** or reduce the size of the result and pause
- For faster performance, use **Parallel Scan**
  - Multiple workers scan multiple data segments at the same time
  - Increases the throughput and RCU consumed
  - Limit the impact of parallel scans just like you would for Scans
- Can use **ProjectionExpression** & **FilterExpression** (no changes to RCU)

DynamoDB – Deleting Data

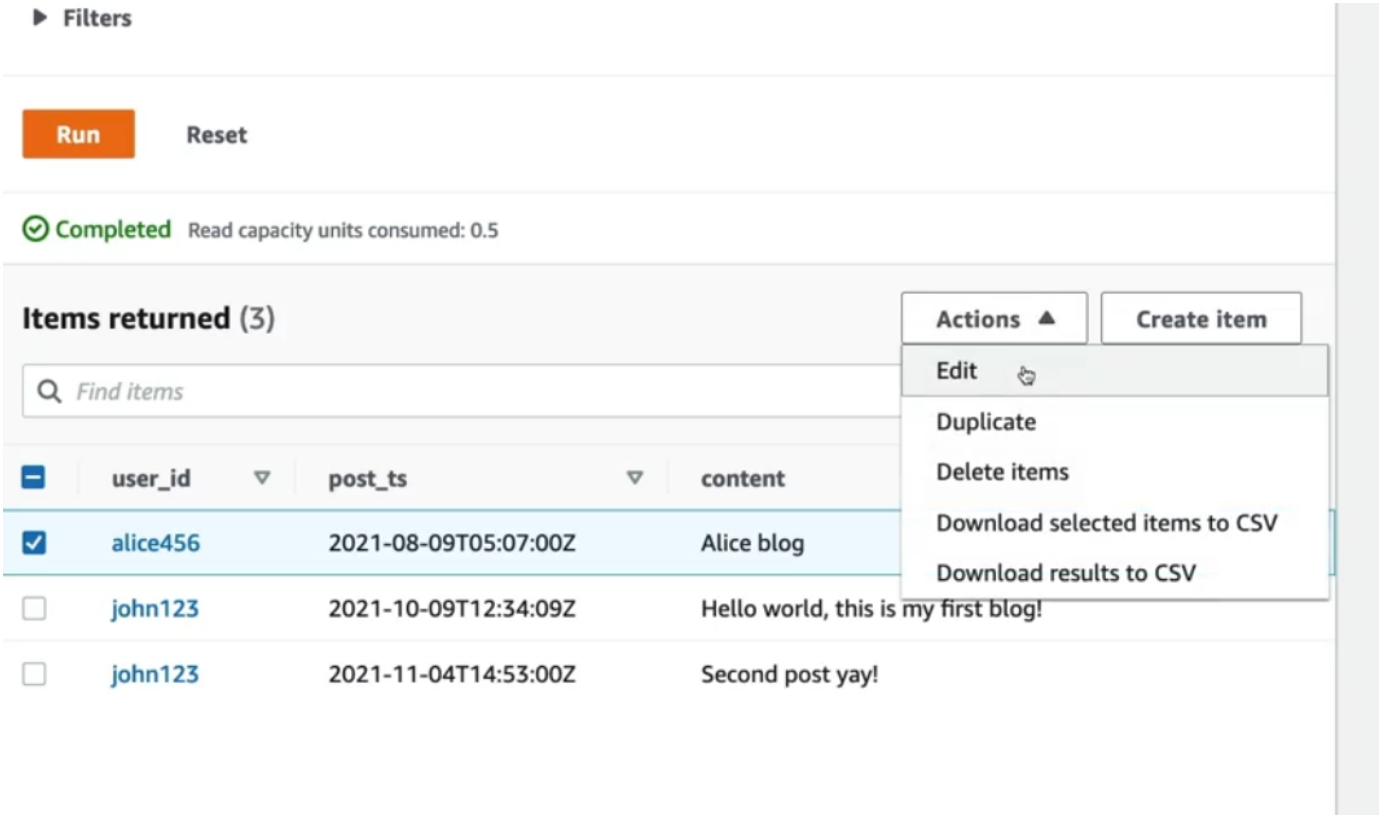
- DeleteItem
  - Delete an individual item
  - Ability to perform a conditional delete
- DeleteTable
  - Delete a whole table and all its items
  - Much quicker deletion than calling **DeleteItem** on all items

DynamoDB – Batch Operations

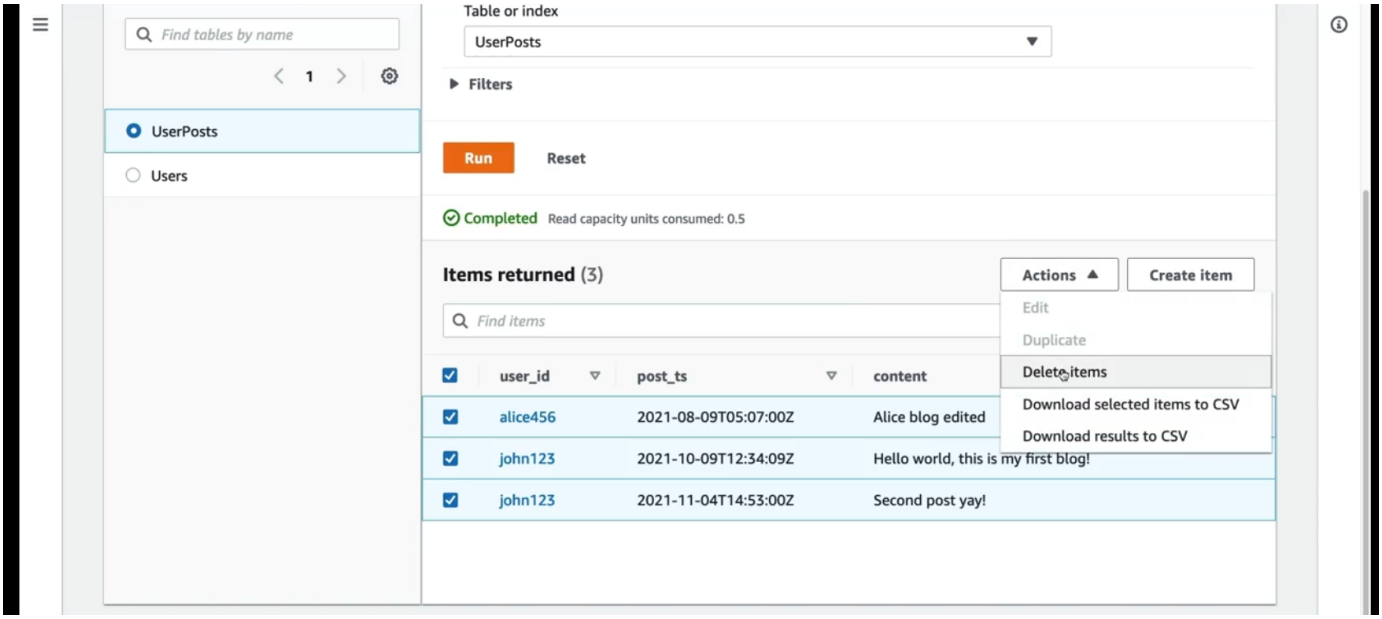
- Allows you to save in latency by reducing the number of API calls
- Operations are done in parallel for better efficiency
- Part of a batch can fail; in which case we need to try again for the failed items
- BatchWriteItem
  - Up to 25 **PutItem** and/or **DeleteItem** in one call
  - Up to 16 MB of data written, up to 400 KB of data per item
  - Can't update items (use **UpdateItem**)
- BatchGetItem
  - Return items from one or more tables
  - Up to 100 items, up to 16 MB of data
  - Items are retrieved in parallel to minimize latency

you add the data means it was puts data

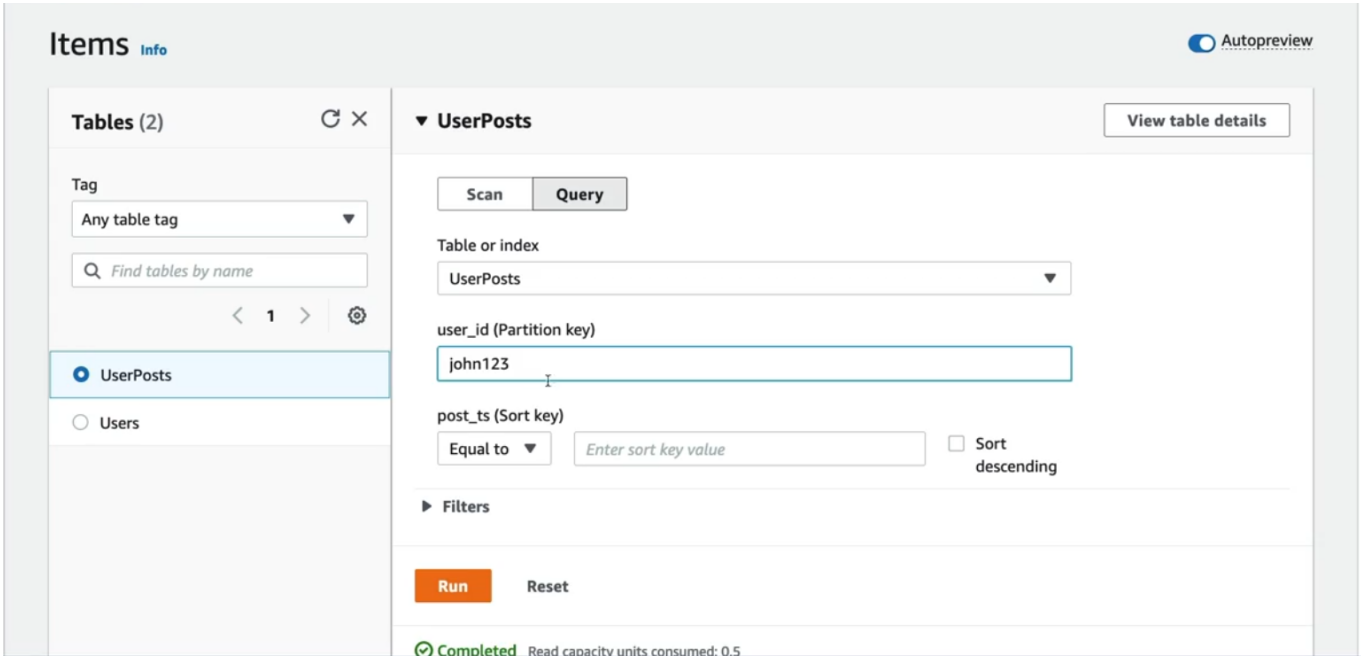
-> Action there is edit option you update the item is called get item



back action



query



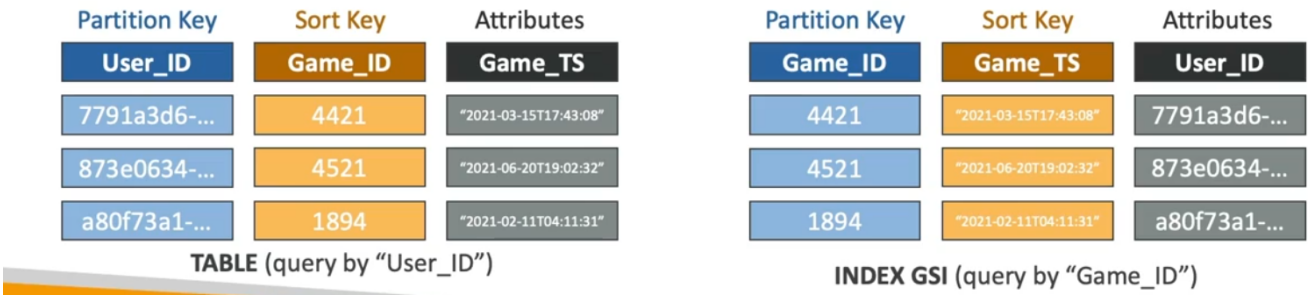
DynamoDB – Local Secondary Index (LSI)

- Alternative Sort Key for your table (same Partition Key as that of base table)
- The Sort Key consists of one scalar attribute (String, Number, or Binary)
- Up to 5 Local Secondary Indexes per table
- Must be defined at table creation time
- Attribute Projections – can contain some or all the attributes of the base table (KEYS_ONLY, INCLUDE, ALL)

Primary Key		Attributes		
Partition Key	Sort Key	LSI		
User_ID	Game_ID	Game_TS	Score	Result
7791a3d6-...	4421	"2021-03-15T17:43:08"	92	Win
873e0634-...	4521	"2021-06-20T19:02:32"		Lose
a80f73a1-...	1894	"2021-02-11T04:11:31"	77	Win

DynamoDB – Global Secondary Index (GSI)

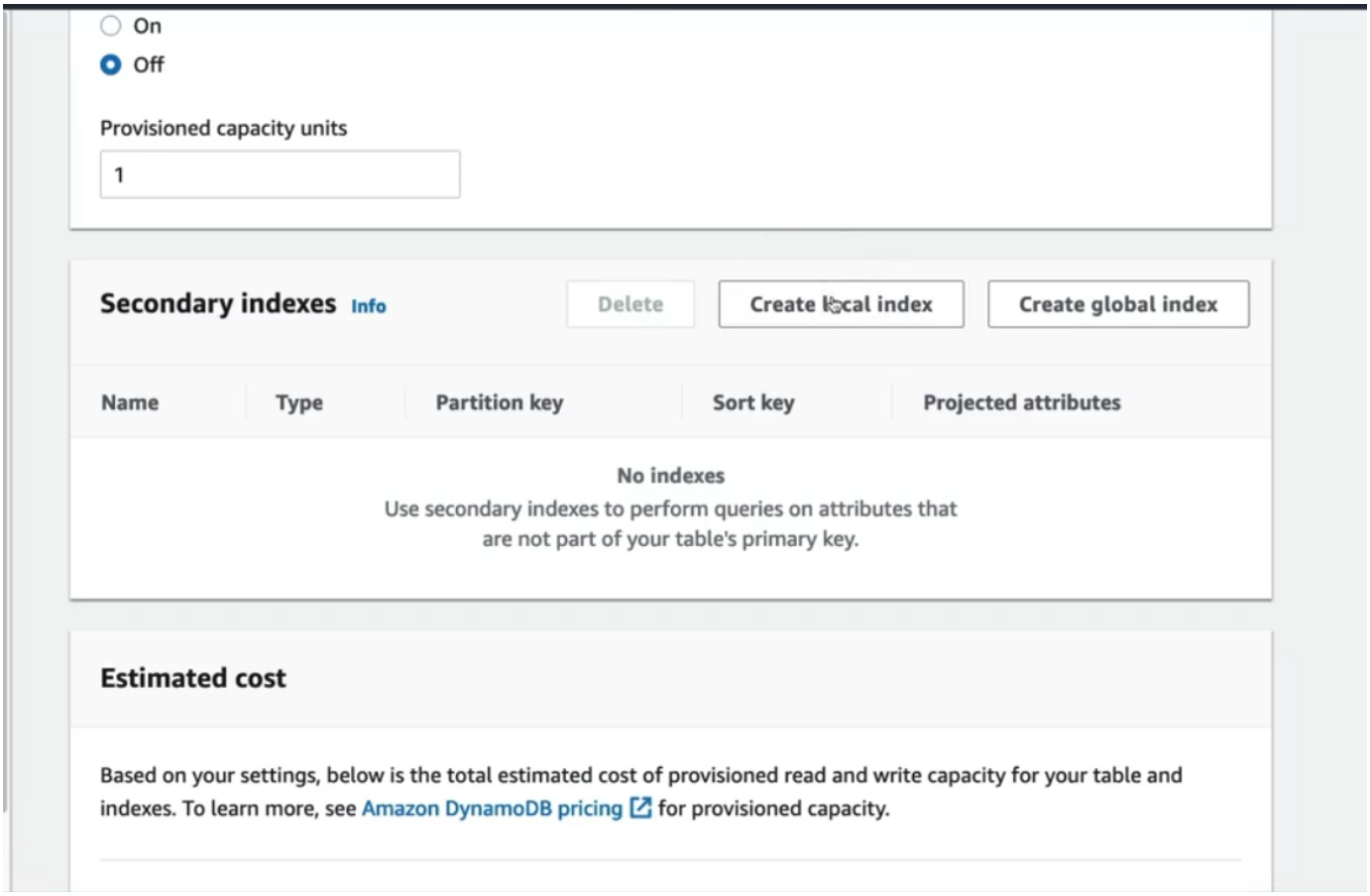
- **Alternative Primary Key (HASH or HASH+RANGE)** from the base table
- Speed up queries on non-key attributes
- The Index Key consists of scalar attributes (String, Number, or Binary)
- **Attribute Projections** – some or all the attributes of the base table (KEYS_ONLY, INCLUDE, ALL)
- Must provision RCUs & WCUs for the index
- **Can be added/modified after table creation**



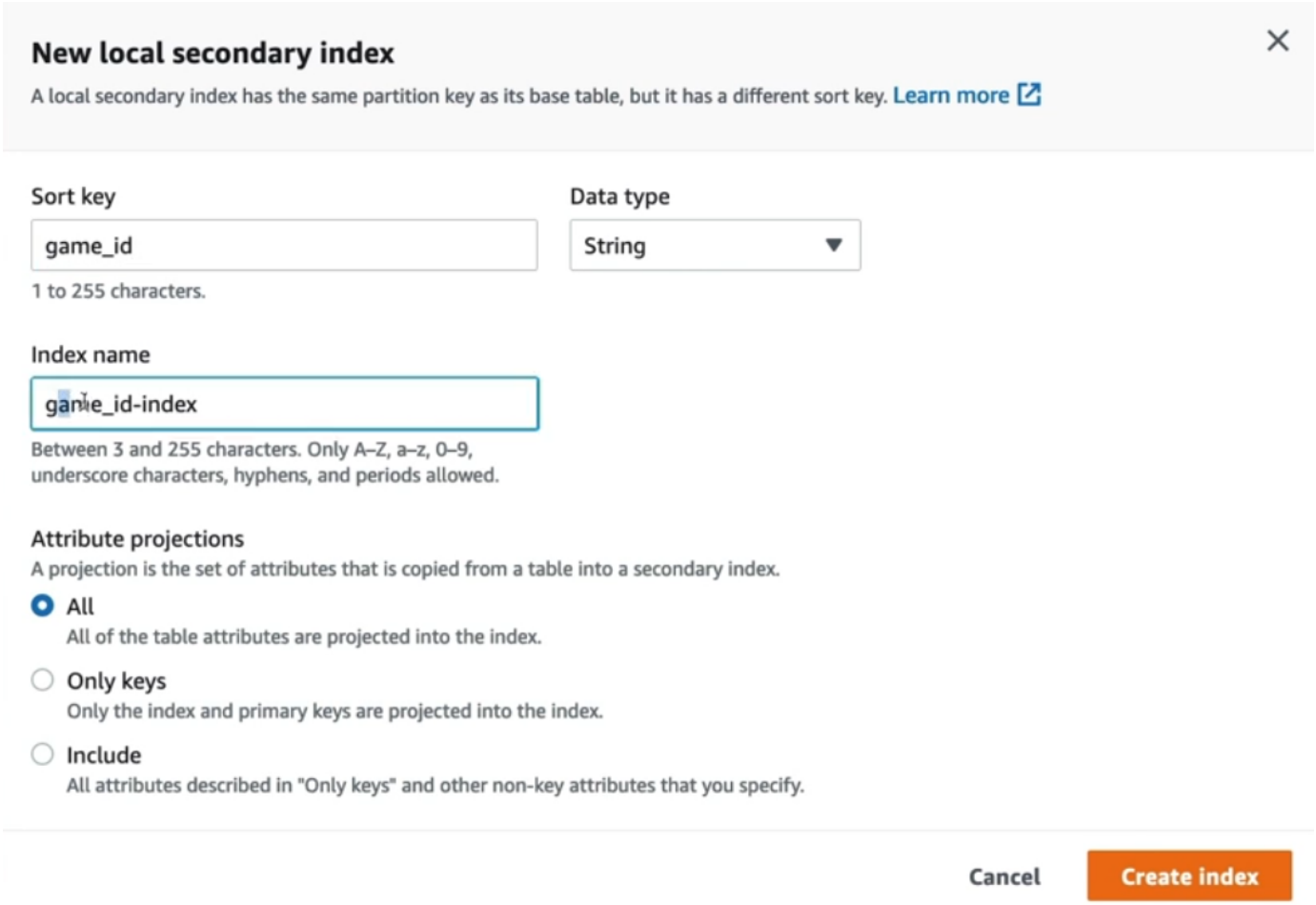
DynamoDB – Indexes and Throttling

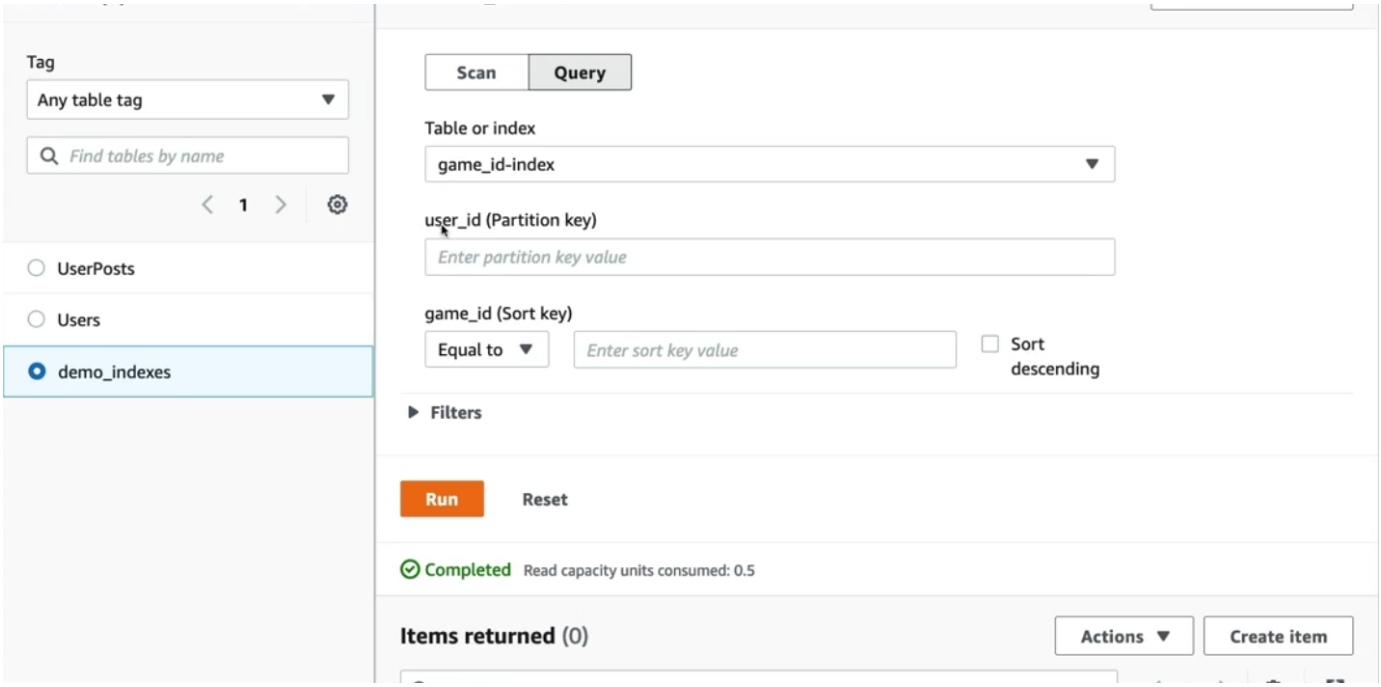
- Global Secondary Index (GSI):
  - If the writes are throttled on the GSI, then the main table will be throttled!
  - Even if the WCU on the main tables are fine
  - Choose your GSI partition key carefully!
  - Assign your WCU capacity carefully!
- Local Secondary Index (LSI):
  - Uses the WCUs and RCUs of the main table
  - No special throttling considerations

local index you can create only when table was created ; gobal index you can create also after table was created



#creating a index

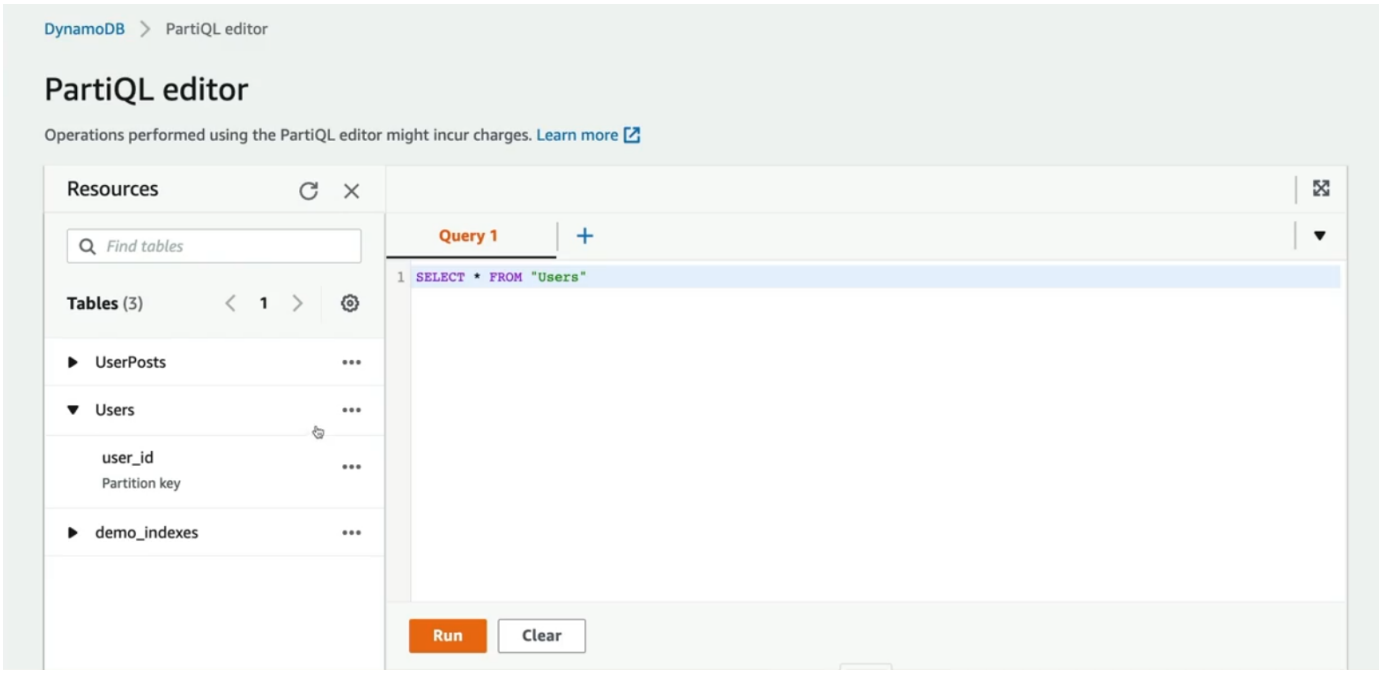




DynamoDB - PartiQL

- Use a SQL-like syntax to manipulate DynamoDB tables

- Supports some (but not all) statements:
  - INSERT
  - UPDATE
  - SELECT
  - DELETE



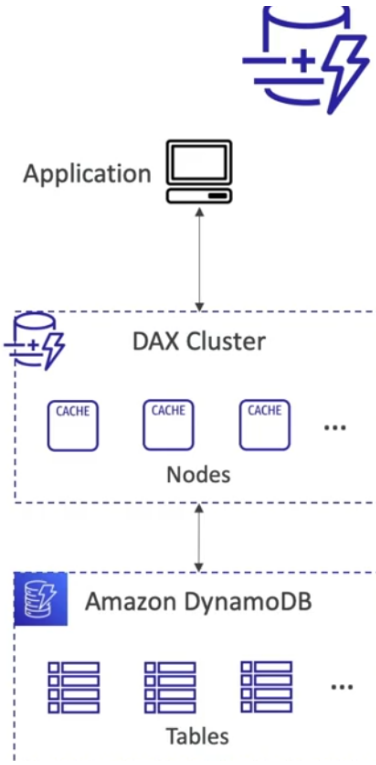
Run Clear

Table view | JSON view

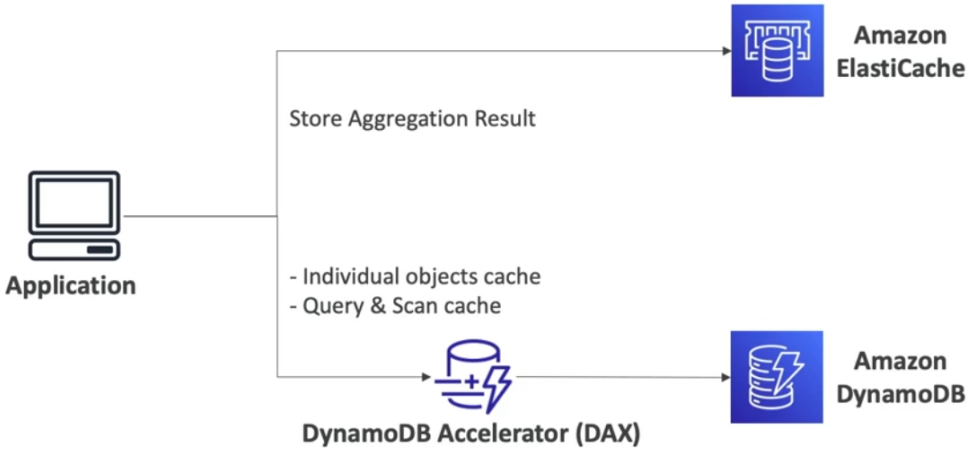

```
"Items" : [ 1 item
  0 : { 2 items
    "name" : { 1 item
      "S" : "stephane"
    "user_id" : { 1 item
      "S" : "123"
    }
  }
]
```


DynamoDB Accelerator (DAX)

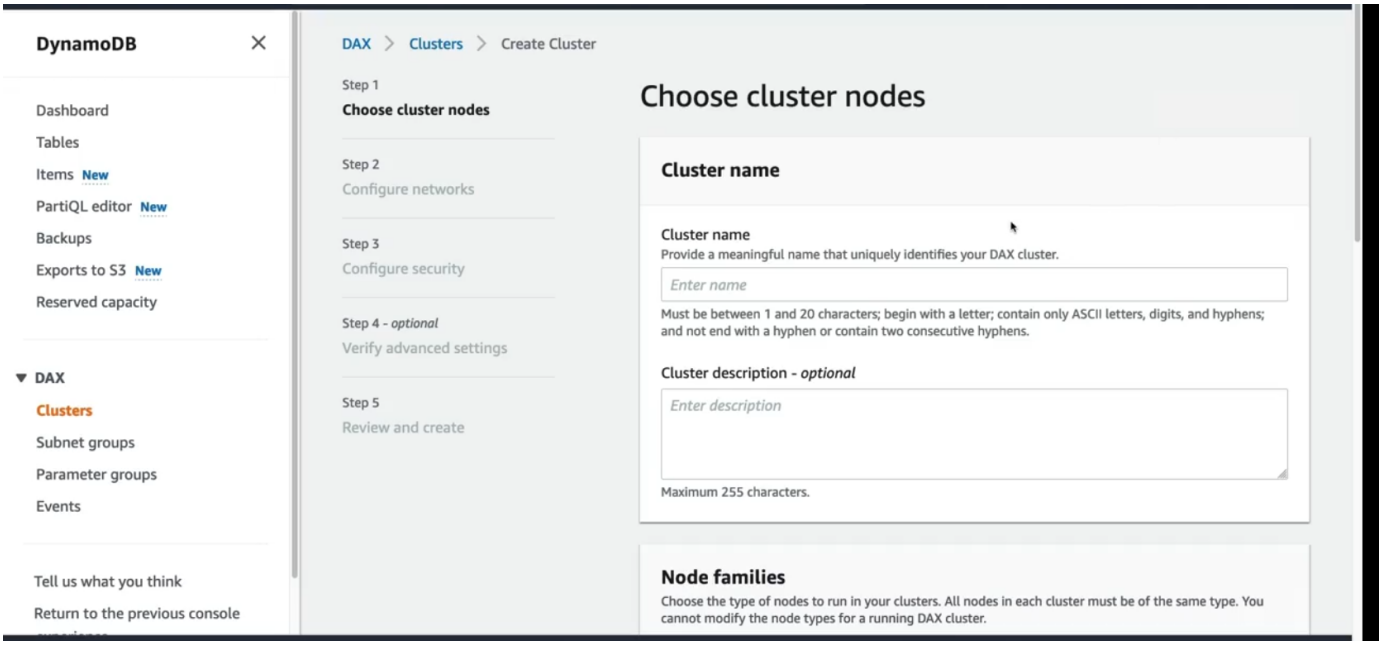
- Fully-managed, highly available, seamless in-memory cache for DynamoDB
- Microseconds latency for cached reads & queries
- Doesn't require application logic modification (compatible with existing DynamoDB APIs)
- Solves the "Hot Key" problem (too many reads)
- 5 minutes TTL for cache (default)
- Up to 10 nodes in the cluster
- Multi-AZ (3 nodes minimum recommended for production)
- Secure (Encryption at rest with KMS, VPC, IAM, CloudTrail, …)



DynamoDB Accelerator (DAX) vs. ElastiCache



dax is not free it expensive

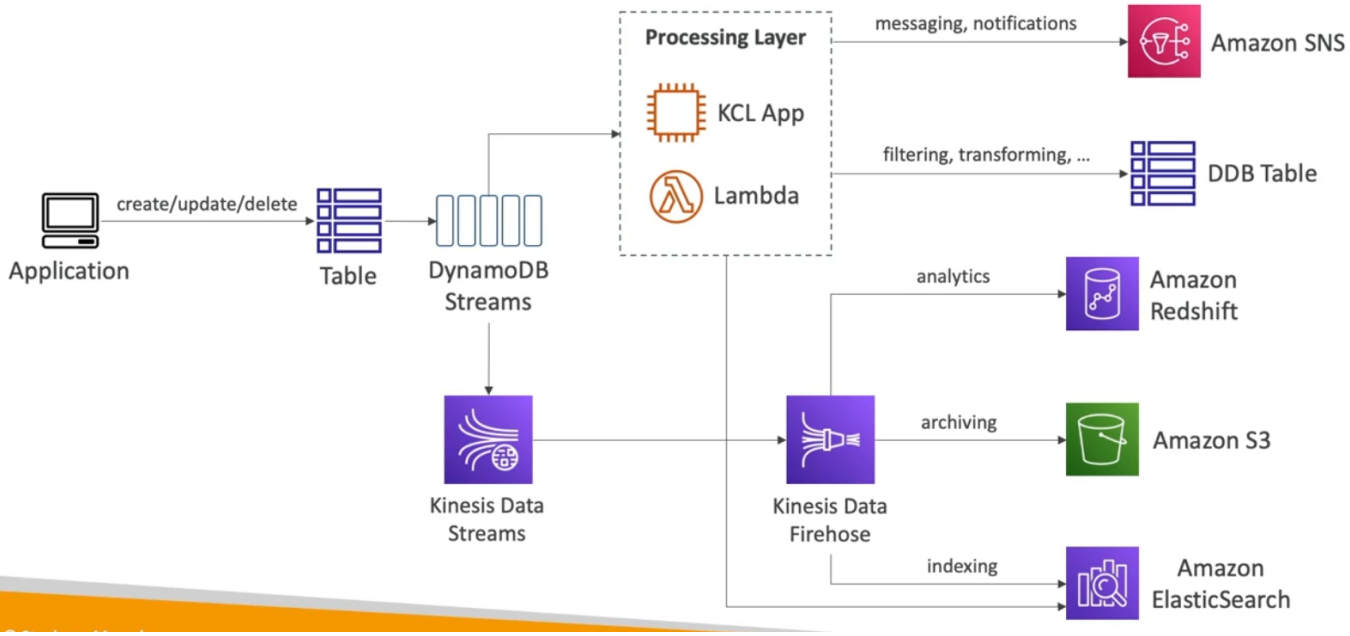


then you select you create node family clusters , subnets , create a iam role after that they shows ports are select default vpc r ec2 machine

DynamoDB Streams

- Ordered stream of item-level modifications (create/update/delete) in a table
- Stream records can be:
  - Sent to **Kinesis Data Streams**
  - Read by **AWS Lambda**
  - Read by **Kinesis Client Library applications**
- Data Retention for up to 24 hours
- Use cases:
  - react to changes in real-time (welcome email to users)
  - Analytics
  - Insert into derivative tables
  - Insert into ElasticSearch
  - Implement cross-region replication

DynamoDB Streams



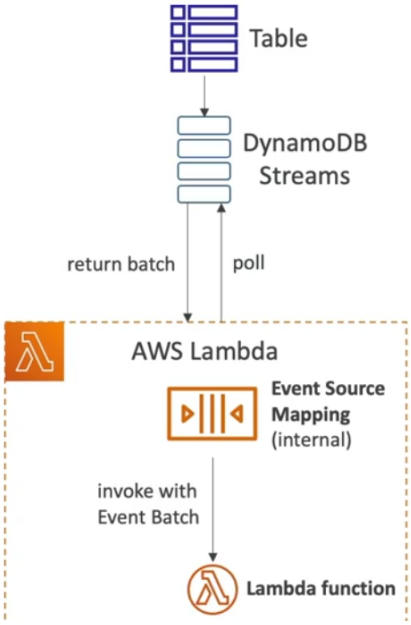
DynamoDB Streams

- Ability to choose the information that will be written to the stream:
  - KEYS_ONLY – only the key attributes of the modified item
  - NEW_IMAGE – the entire item, as it appears after it was modified
  - OLD_IMAGE – the entire item, as it appeared before it was modified
  - NEW_AND_OLD_IMAGES – both the new and the old images of the item
- DynamoDB Streams are made of shards, just like Kinesis Data Streams
- You don't provision shards, this is automated by AWS

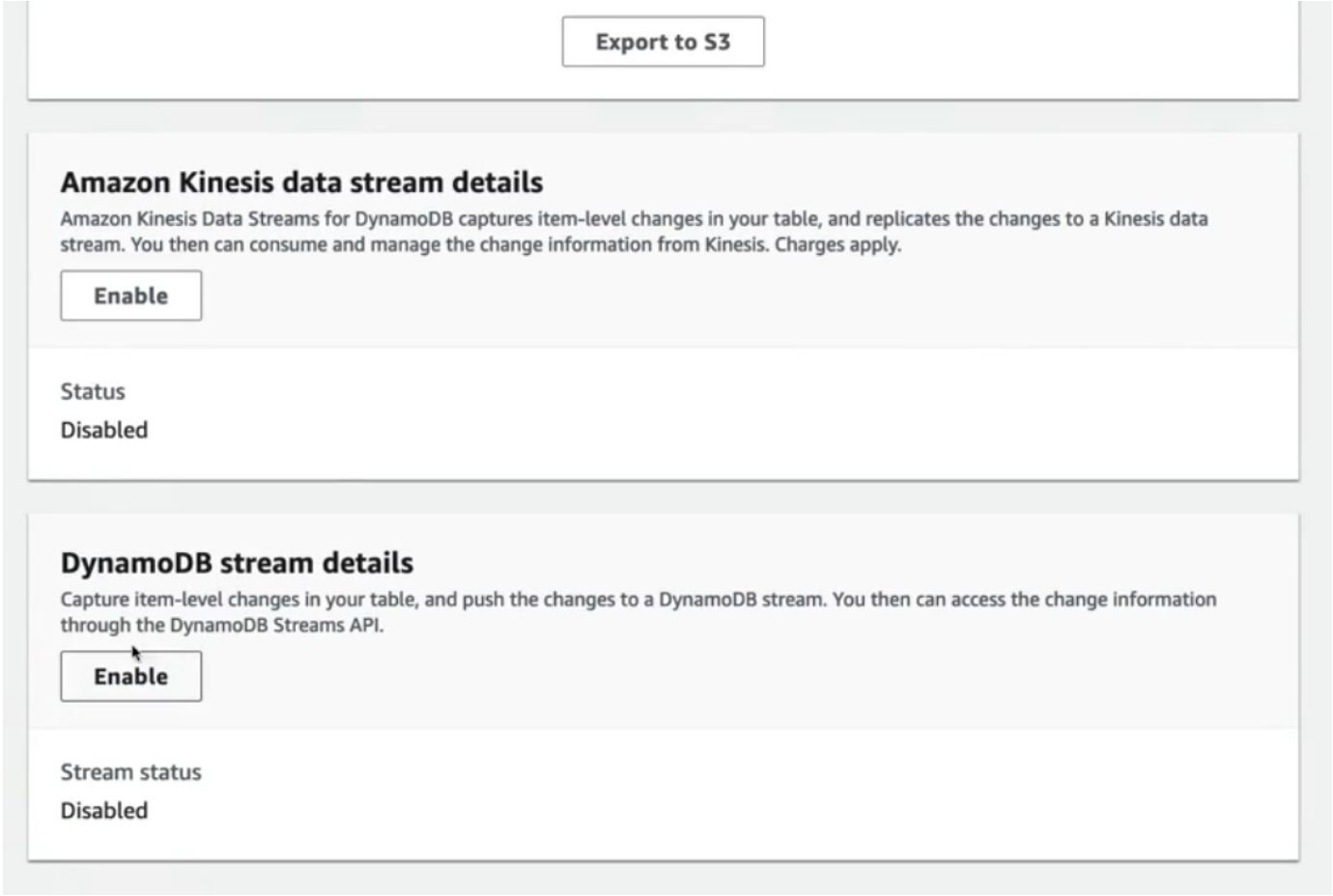
- Records are not retroactively populated in a stream after enabling it

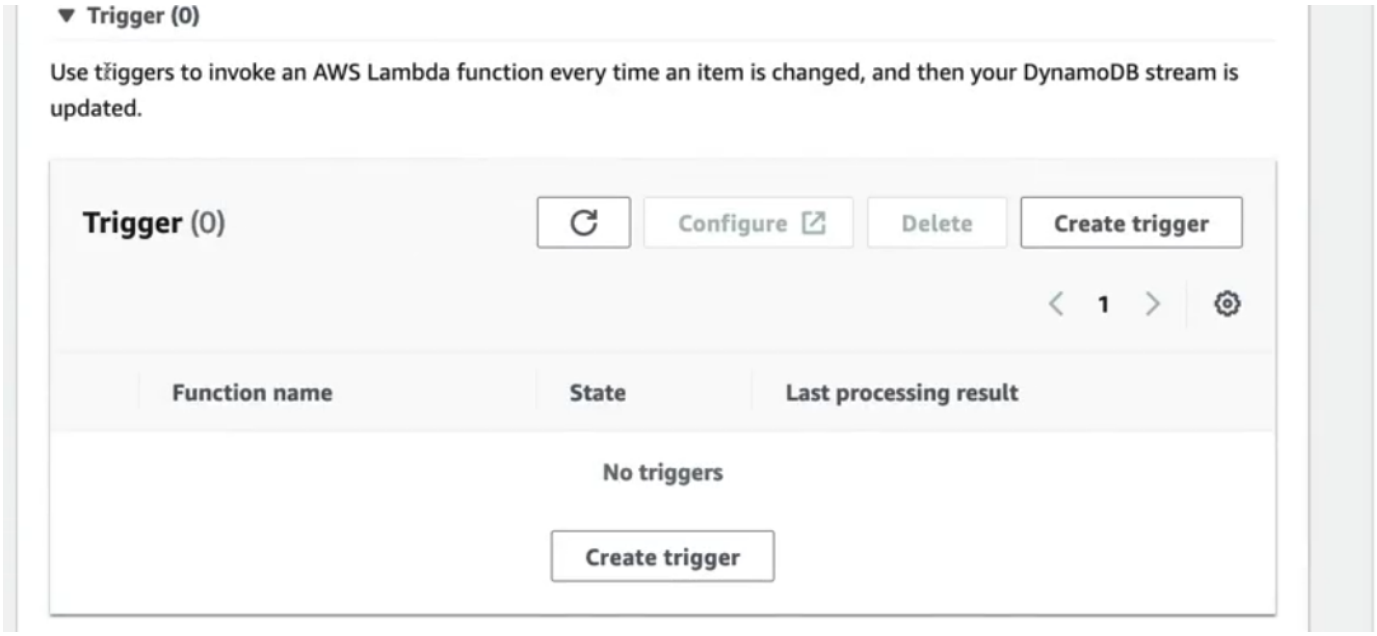
DynamoDB Streams & AWS Lambda

- You need to define an **Event Source Mapping** to read from a DynamoDB Streams
- You need to ensure the Lambda function has the **appropriate permissions**
- Your Lambda function is invoked synchronously



dynamodb stream in hands on you need to go export and streams --> dynamodb stream details --> enable option

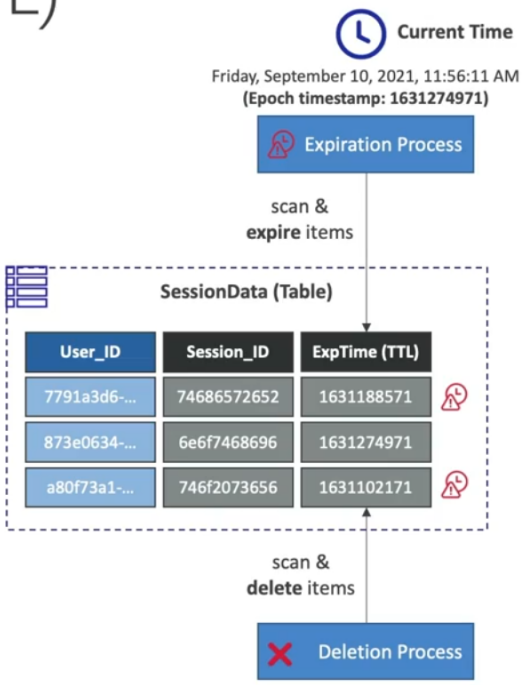


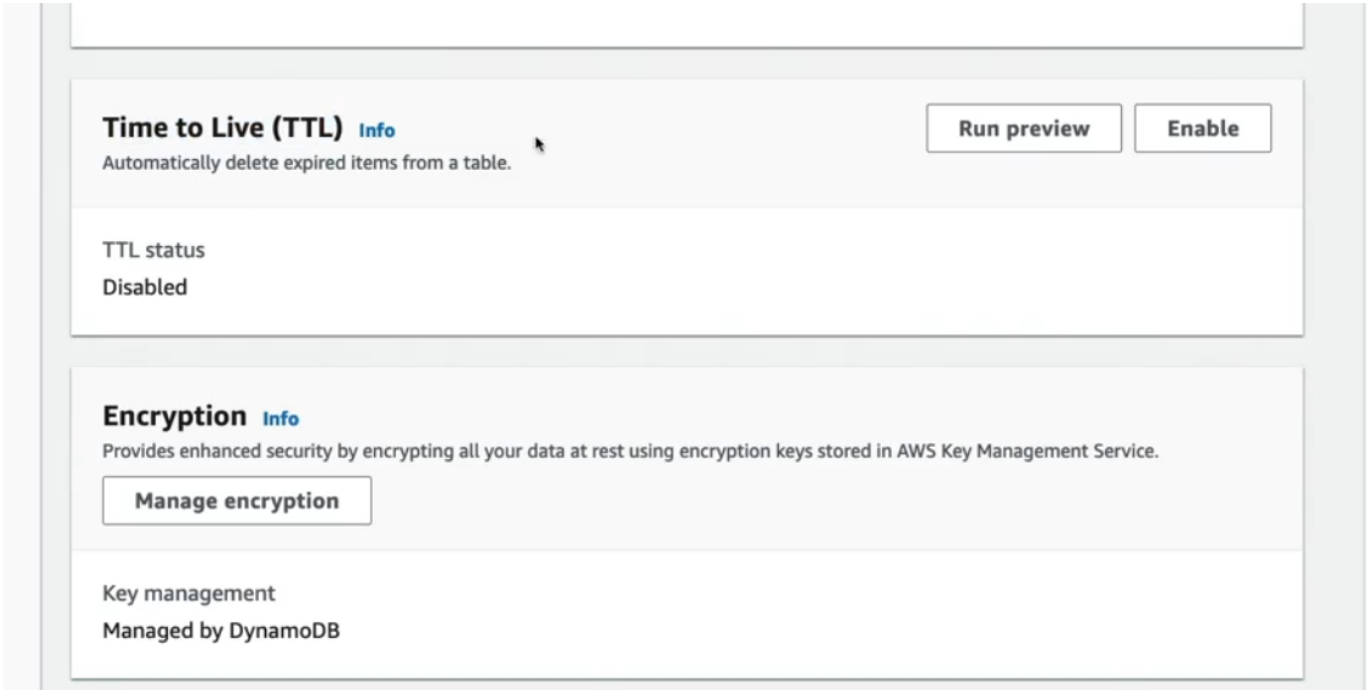


then create a trigger -> create a lambda function using blue print are from strach and attache lambda function to the dynamodb stream to run that

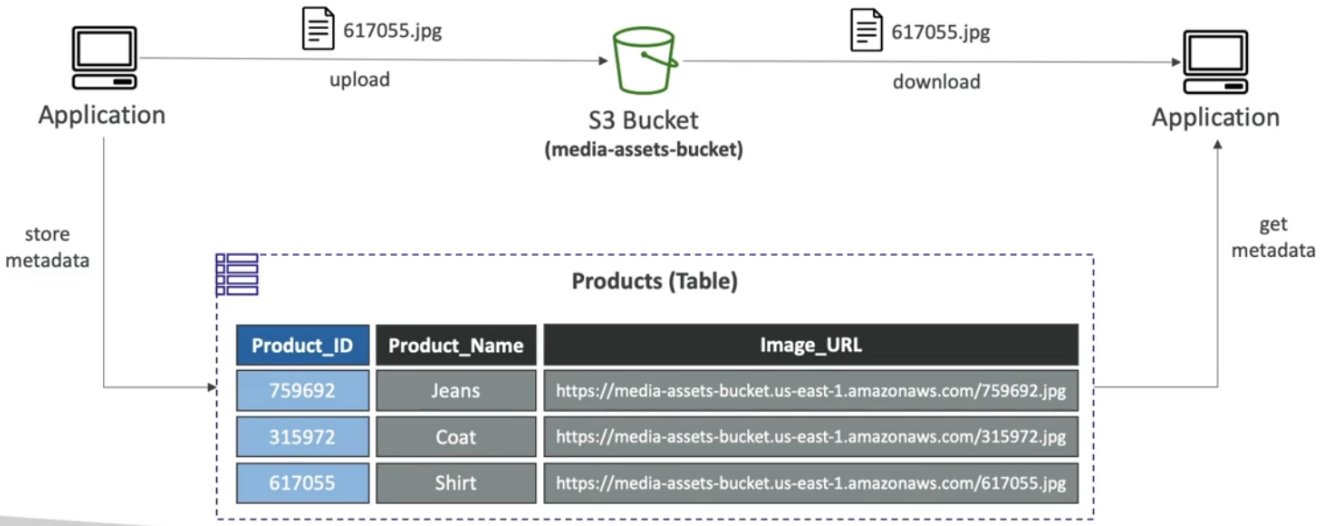
DynamoDB – Time To Live (TTL)

- Automatically delete items after an expiry timestamp
- Doesn't consume any WCUs (i.e., no extra cost)
- The TTL attribute must be a "Number" data type with "Unix Epoch timestamp" value
- Expired items deleted within 48 hours of expiration
- Expired items, that haven't been deleted, appears in reads/queries/scans (if you don't want them, filter them out)
- Expired items are deleted from both LSIs and GSIs
- A delete operation for each expired item enters the DynamoDB Streams (can help recover expired items)
- Use cases: reduce stored data by keeping only current items, adhere to regulatory obligations, …

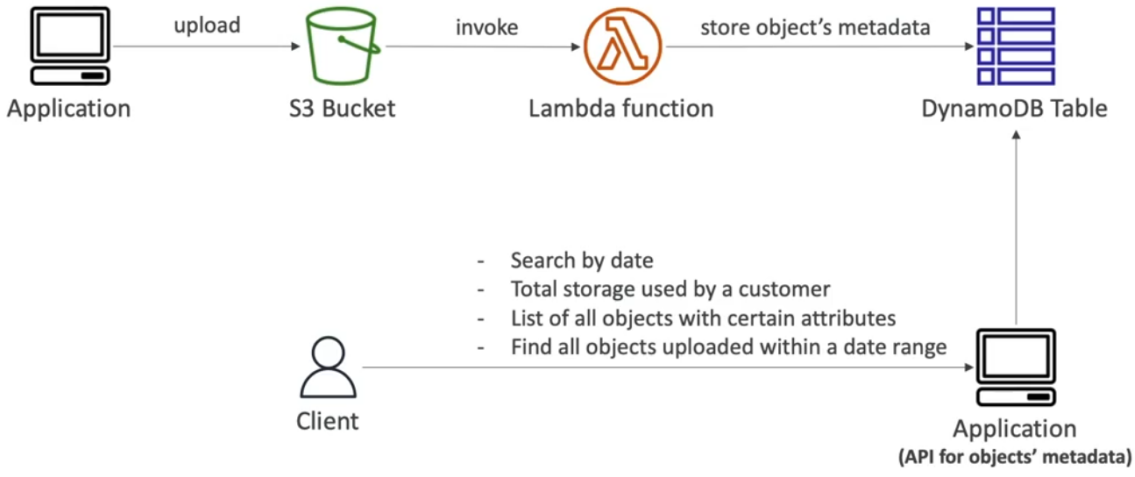




DynamoDB – Large Objects Pattern



DynamoDB – Indexing S3 Objects Metadata



DynamoDB – Security & Other Features

- Security:
  - VPC Endpoints available to access DynamoDB without internet
  - Access fully controlled by IAM
  - Encryption at rest using KMS
  - Encryption in transit using SSL / TLS
- Backup and Restore feature available
  - Point in time restore like RDS
  - No performance impact
- Global Tables
  - Multi region, fully replicated, high performance
- Amazon Database Migration Service (DMS) can be used to migrate to DynamoDB (from Mongo, Oracle, MySQL, S3, etc…)
- You can launch a local DynamoDB on your computer for development purposes