  
 DYNAMO DB - AMAZONS NOSQL DATABASE  
  
1. What is DynamoDB : It is a FAST and FLEXIBLE nosql database. It is for applications that need single digit milli second latency.

2. Supports : It supports both document ( json/html/xml) as well as key-value data models.

3. Reliable performance : dynamodb databases are always backed by **ssd’s**.

4. Serverless , fully managed and configured to autoscale.

5. Availability : The underlying hardware supporting the database instance is always spread across **3** geographically distinct data centers. Thus avoiding any single point failure/unavailability.

6. CONSISTENCY MODELS FOR DYNAMO DB READ OPERATIONS.  
Eventually consistent reads : In this model Performance is given importance over consistency. Consistency across all the copies of the data across the 3 locations is usually reached with in a second (if a read is performed within 100 or 200 milliseconds after you write, there is a chance that you could get old or new data).   
Note : Whenever a data write occurs DynamoDB stores this data 3 times i.e in 3 locations. As soon as it is written in 2 centers as 200 OK status code is returned to the client.  
i.e you will receive a commit OK message after only two of those replicas have been updated. And there is a small chance that a read immediately after a write, will retrieve out of data information from an un-updated replica, unless we want to specify we want a strongly consistent read.

Strongly consistent reads : A strongly consistent read will read from multiple locations at the same time to ensure it receives the most up to date copy of the data. A Strongly consistent read will always return a result that reflects all the writes that received a successful response prior to the read.

7. Tables.  
Items : similar to a row – key value pairs.  
Attributes : columns..  
key value and document data structures are supported.

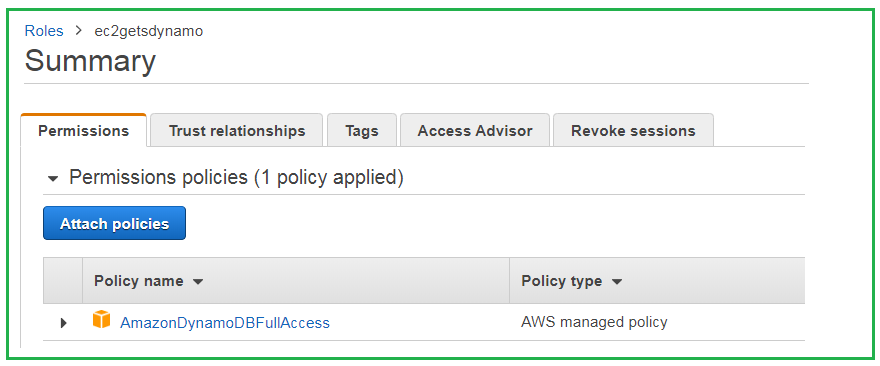
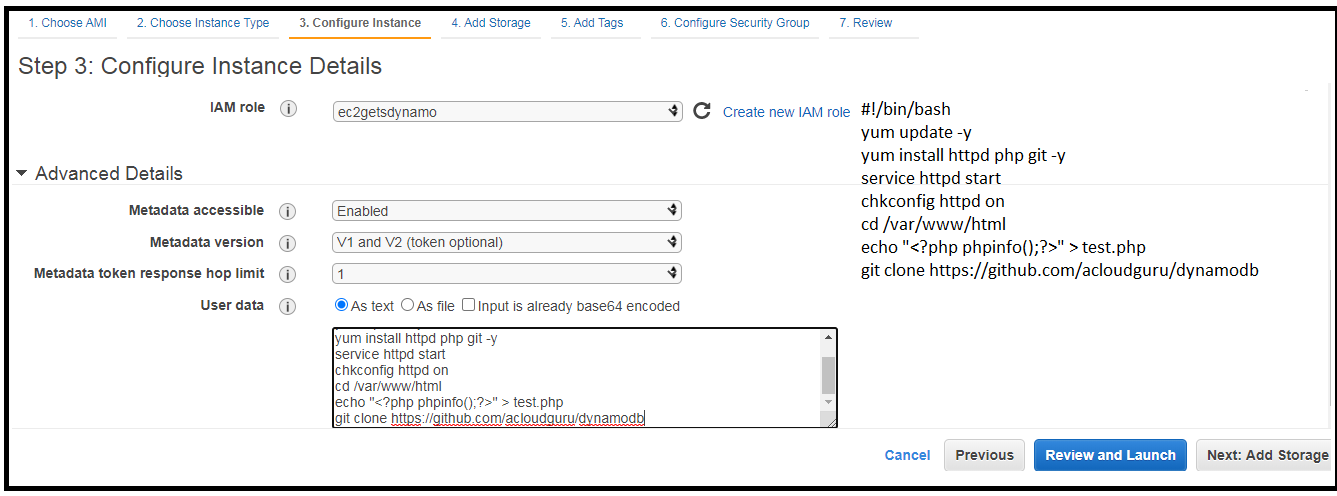
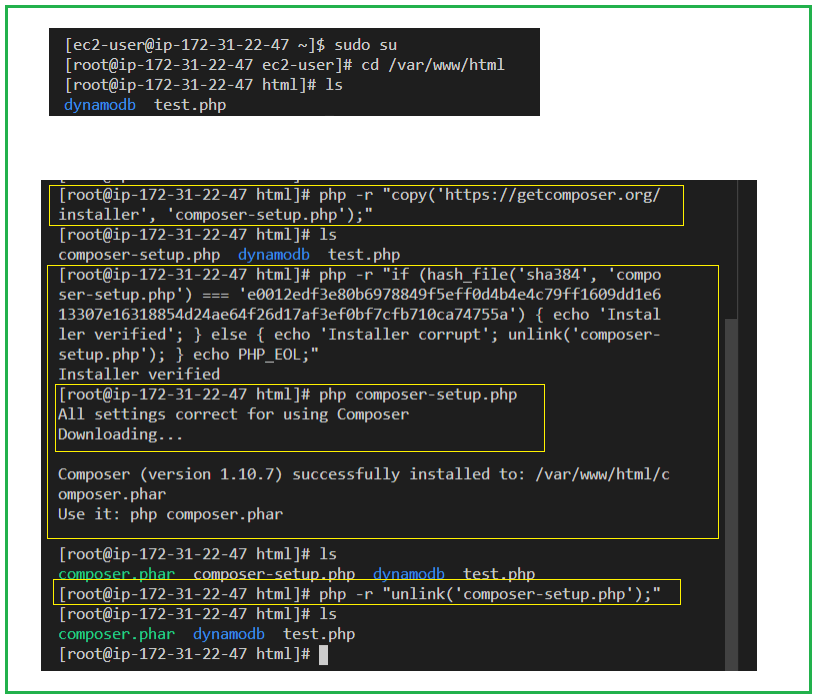
8. Primary keys : How do we query data in the table.  
DynamoDB stores and retrieves data based on the primary key.  
The primary key is used when we query the data.

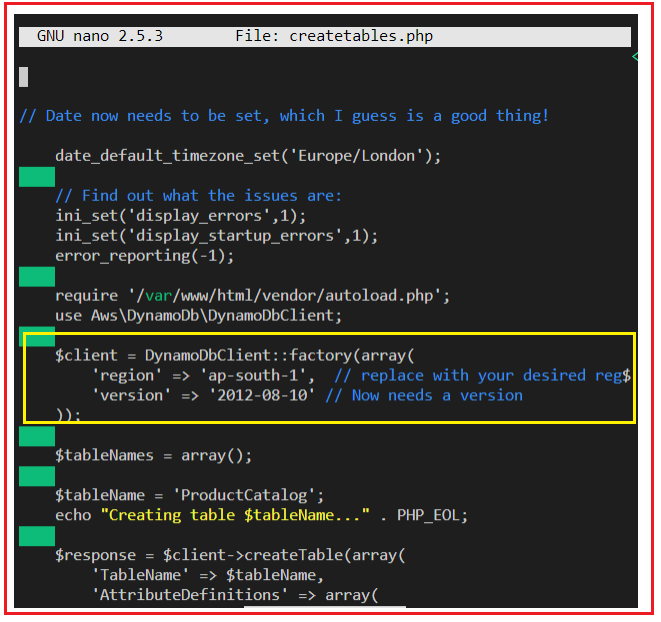
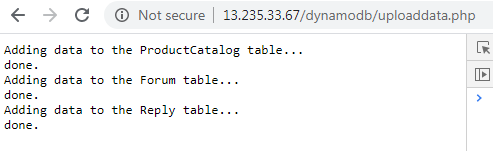
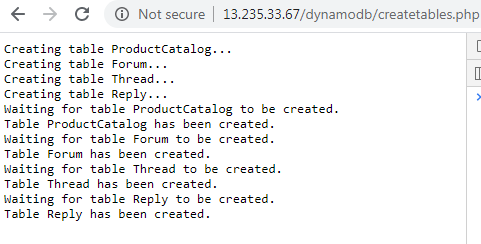
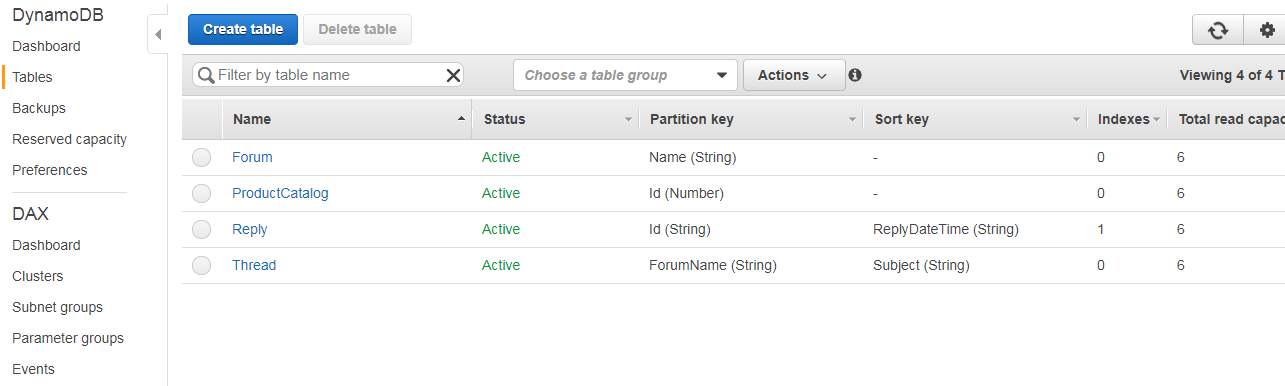
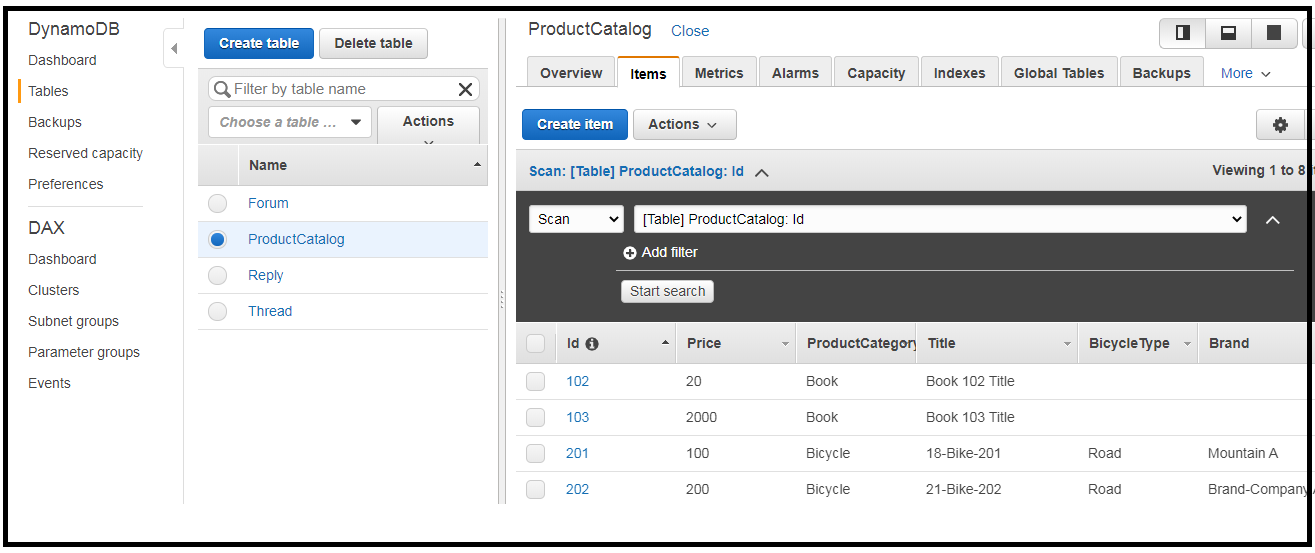
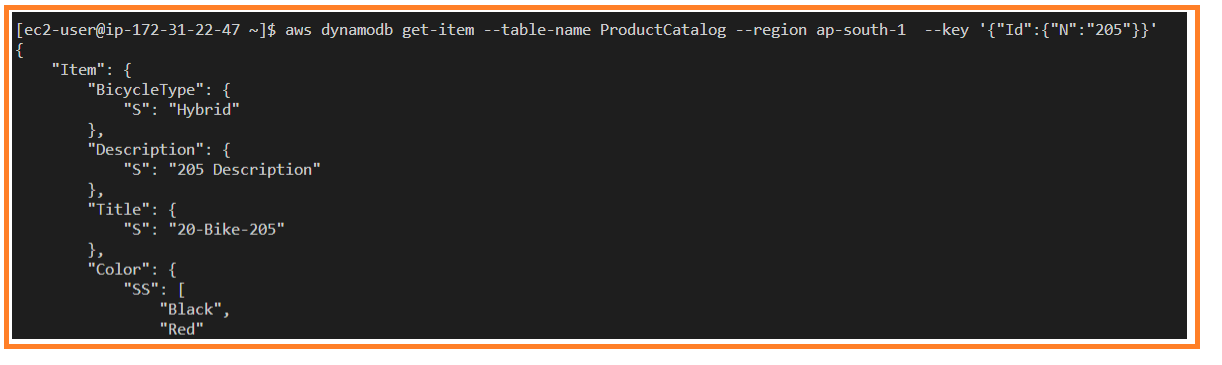
There are 2 types of primary key.  
Partition Key : It is a unique attribute [The value of a key which has to be unique]  
The value of the partition key is input into a internal hash function, The output of the hash function actually determines the partition or the **physical location** where that data is going to be stored.  
If you are using the partition key as the primary key then no two items with in your table can have the same partition key. As we cannot store two items in the same physical location.

Composite key : This is the second type of primary key. This is made up of a combination of the partition key and a sort key.  
composite key = partition key + sort key.  
If the partition key is not necessarily unique then you will use this key.  
Example : Consider a forum posts. We might be storing attributes like the UserID, The message typed in by the user, The timestamp of the post, The name of the Forum. In this example the Primary key could be a composite key which is made up of userID (partition key) and timestamp of the post(sort key).  
In this case two items could have the same partition key, because we could have the same user posting multiple times in the same forum or even different forums. In order to identify the record uniquely we need to use a combination of the ‘userID’ and the ‘time’ that posted the post in a forum.  
Now all items with the same partition key are stored together and they are sorted with a value of the sort key.

9. Access Control to DynamoDB.

- It is all managed by IAM

10. Dynamo DB **lab**. We create an EC2 instance and execute php scripts that will create a table in DynamoDB and insert documents.  
  
10.1 Create a role for EC@ to access DynamoDB service.  
  
20.2  
Create an EC2 instance and install apache and pull a repo from git.  
  
  
20.3 We need to install PHP and AWS SDK for PHP so follow the below steps.  
  
Commands to install Composer and AWS SDK for PHP:  
We use the composer to install the latest version of aws sdk for PHP.  
  
Note : # If you are experiencing memory issues with your t2.micro when trying to install the AWS SDK,

# Please run the following commands and try again:  
/bin/dd if=/dev/zero of=/var/swap.1 bs=1M count=1024  
/sbin/mkswap /var/swap.1  
/sbin/swapon /var/swap.1  
  
20.4 Edit the below files so that the region matches to yours.  
  
  
  
  
20.5  
With the EC2 instance public ip use the below php file to create a dynamodb table and enter data into them.  
  
20.6  
Dynamo DB should look like this.  
  
  
20.7 The items in the table look like below.  
  
  
20.8 Accessing the DynamoDB Service using the CLI.  


21. DYNAMODB INDEXES.  
  
Secondary index : It is a data structure that contains a subset of attributes from a table, along with an alternate key to support query operations.  
A table can have multiple secondary index, indexes which give your application access to many different query patters.  
  
Indexes (pointers to a particular column) enable fast queries on specific data columns[attributes in a document]  
DynamoDb is a not a sql database but still indexes exist.  
DynamoDB provides provides fast access to items in a table, by specifying primary key values.  
Indexing comes into picture if you want to fetch the data of ‘attributes’ other than the primary key.  
[Something like the primary key will fetch the whole document.]

In DynamoDB, 2 types of indexes are supported to help speed-up the dynamodb query.  
  
LOCAL SECONDARY INDEX :   
- The local secondary index can only be **created when you are creating the table**, and cannot be added or removed or modified later.  
- It has the **same** **partition** **key** as your original table, but a different sort key.  
- The data is organized according to the sort key.  
- Any queries based on this sort key, that you run against the local secondary index are much faster that they would be if you run them on the the main table.  
- With the local secondary index we can create 5 per table and its free.

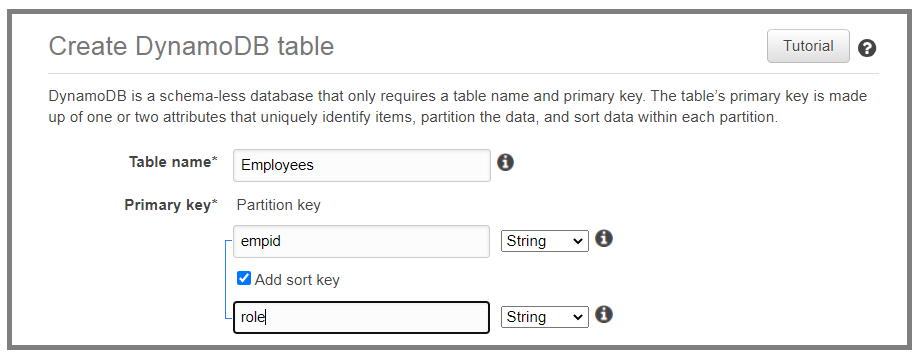
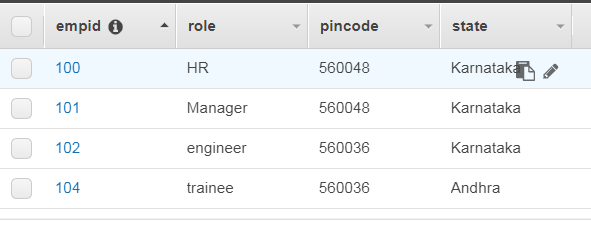
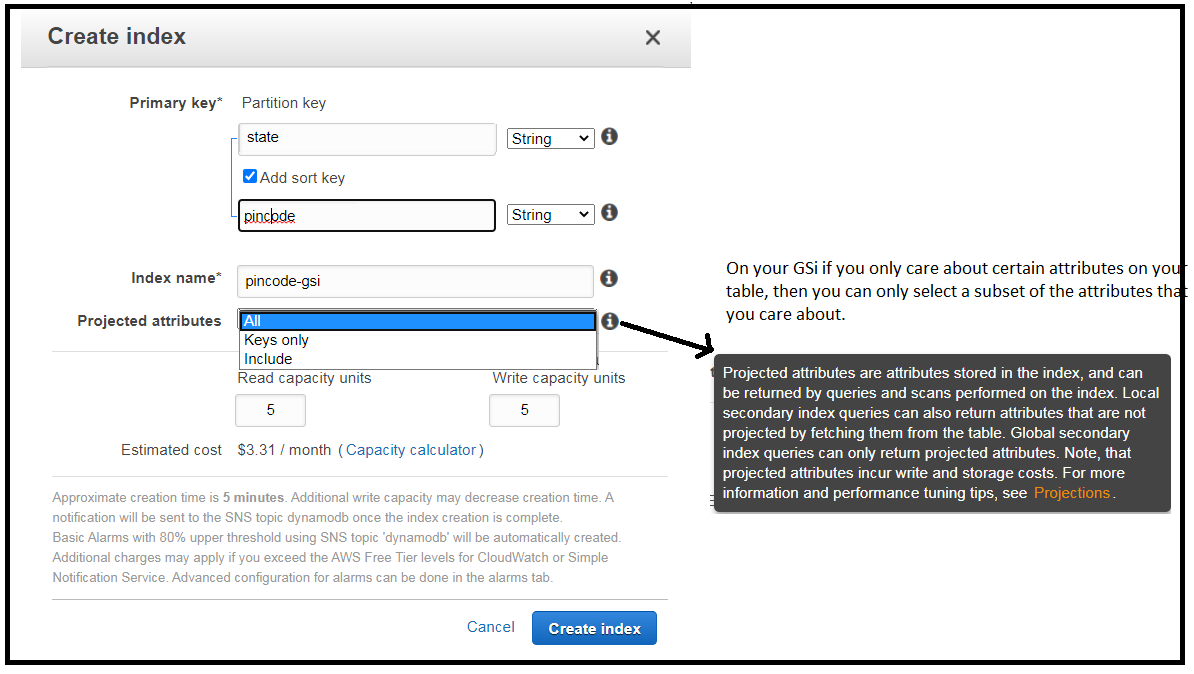
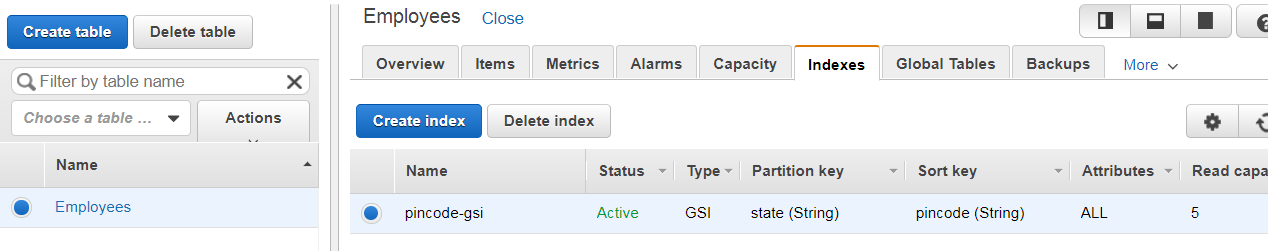
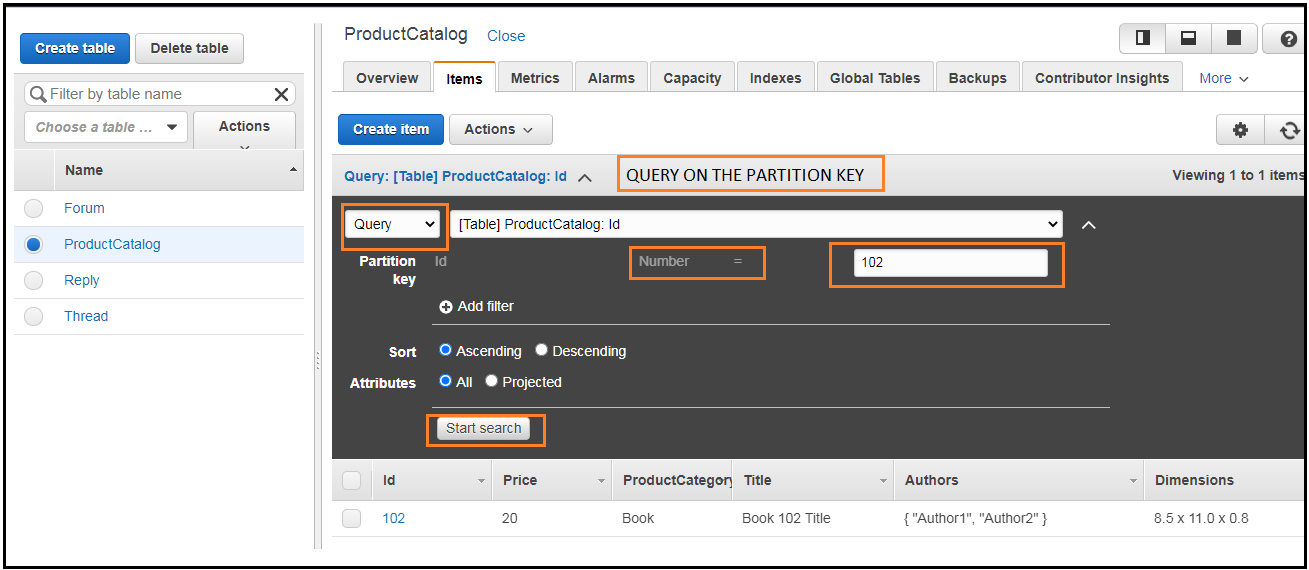
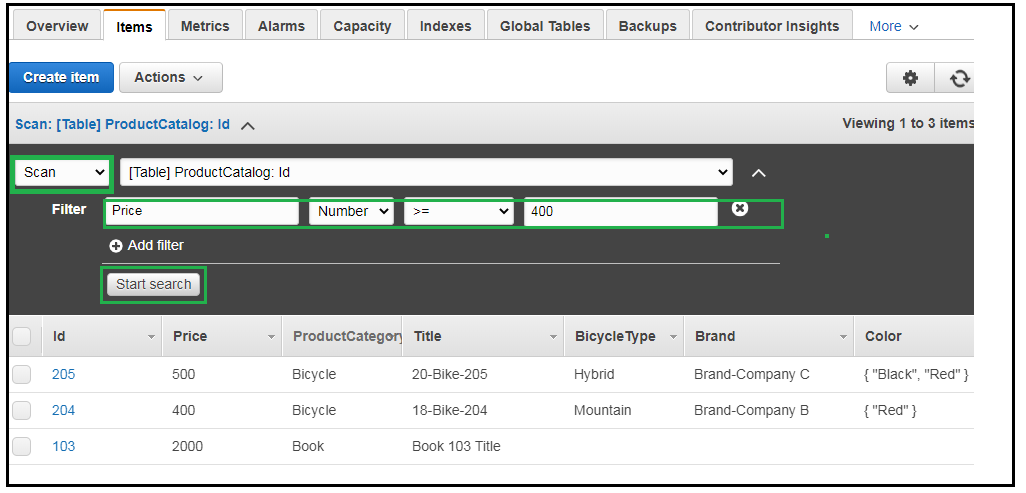
|  |  |  |
| --- | --- | --- |
| FORUM NAME | LASTPOST-TIME | SUBJECT |
| S3 | 2020-09-06 10:10:00 | ‘how to list buckets’ |
| S3 | 2020-09-06 10:12:00 | ‘how to create a bucket’ |
| EC2 | 2020-09-06 10:14:00 | ‘how to create an ec2 instance’ |
| EC2 | 2020-09-06 10:16:00 | ‘how to use ec2 as webserver’ |
| RDS | 2020-09-06 10:18:00 | ‘choose best rds’ |
| RDS | 2020-09-06 10:20:00 | ‘debugging rds’ |

- We track the time about when a post was made to each forum.  
- The schema is as follows : The partition key is ‘forum name’. The ‘lastpost-time’ is the sort-key. ’subject’ is just an attribute.  
- Partition key : Just using the partition key we can perform some direct queries/lookups on it. Example you can ask for all the records with ‘forum name = ‘s3’ ‘  
- Now using the combination of partition key and sort key : we can ask for all S3 records and ‘lastpost-time’ is not more than 15 days old (This is possible because of the sort key as you can use a lot of conditions).  
**- Limitations** : We can apply only one sort key on a table in DynamoDB. If we want to perform same kind of condition operations in a different field, ex on the subject field. Like give me all the rows with ‘forum name’ as ‘s3’ and subject equals to ‘how to list buckets’ – we can’t do this as the sort key is already established on the ‘lasttime-post’.  
**- One way** to overcome this limitation is to scan all the records and filter : this involves consumption of most of your read capacity units.  
**- Another way** is to query for records of ‘s3’ and then filter those with a particular subject name. This is better as it consumes less RCU’s.  
- **The most Optimal Way** : Is to apply a local secondary index on the ‘SUBJECT’ attribute.(subject index)  
Now we can define a query with forumname = S3 AND define a ‘sort key/ range key’ expression/condition on the subject field. This is a direct look up and saves you money, returns a single record.  
**- Key limitations** : LSI can be defined at table creation time And a table can have 5 LSI’s(soft limit) And there is no extra cost associated with usage of LSI’s.  
  
GLOBAL SECONDARY INDEX :   
- These are much more flexible.  
- The Global Secondary Index can be created when you create the table or can be added later on.  
- It allows you to choose a **different** **partition** **key** as well as a different sort key ‘to your main table’.  
- It speeds up any queries relating to this alternative Partition key and sort key.  
- Maximum of 20 global secondary indexes can be created.

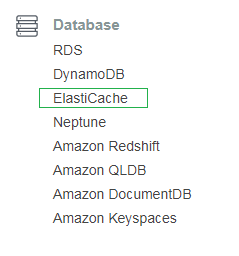
|  |  |  |  |
| --- | --- | --- | --- |
| PRIMARY KEY | |  |  |
| PARTITION KEY | SORT KEY | ATTRIBUTE | ATTRIBUTE |
| empid | role | state | pincode |
| 100 | HR | karnataka | 560048 |
| 101 | manager | karnataka | 560048 |
| 103 | engineer | karnataka | 560036 |
| 104 | trainee | andhra | 560036 |

- If you know the ‘Account Id’ we can perform some very fast lookups.  
- But you can not ask for all the rows with ‘origin country’ as USA.  
- GSI’s allow you to query for attributes of a ‘row that is not the partition key’. Under normal circumstances we can do direct lookups with the partition key.  
- So, If we are performing a query on a GSI, it is very similar to how it would look when performing a normal query, but additionally we provide an ‘index name’ which is the ‘origin country’.  
How does GSI work behind the scenes. ? : Creating a GSI clones the primary table, using the new partition key, but keeps the two tables in sync.   
 - After we specify a (gsi) secondary index – DynamoDb will create a ‘secondary table’ that flips your data.  
 - And depending on the partition key that we have set, we can do lookups based on the new partition key also as opposed to the old partition key.  
- Also if we perform any kind of update/insert/delete operation - DynamoDB will keep these two tables in sync behind the scenes.  
Common things when compared to primary table.  
- Any changes that we do will be eventually propagated to the GSI table.  
- if we are operating a high loads we require that the GSI partition key requires uniform data distribution.  
- Define RCU and WCU separately on the index + we need throttling on the secondary table also.  
GOTCHAS :   
- With every GSI we are effectively creating a clone of the primary table.  
- This means that a write to the main table will result in a write to the GSI, which effectively will double the cost of writing.  
- We can have a maximum of only 20 GSI’s.  
- Writes to the main table are eventually replicated to the GSI, Because of this your GSI can potentially return stale data, especially if there is a substantial delay between ‘updating a record in the primary table ‘ & ‘that record being eventually propagated into the GSI’. So be carful if you are reading from your GSI and attempting to make updates on the primary table, using that record as the source of truth [as we could be getting stale data from the gsi, and putting that back in the primary table]  
- Make sure that you have the WCU of the GSI tables greater than or equal to the WCU of the main/primary table.  
- you will have separate metrics on the GSI and you need to monitor them accordingly like alarms.

|  |  |  |  |
| --- | --- | --- | --- |
| PRIMARY KEY | |  |  |
| PARTITION KEY | SORT KEY | ATTRIBUTE | ATTRIBUTE |
| state | pincode | empid | role |
| karnataka | 560048 | 100 | HR |
| karnataka | 560048 | 101 | manager |
| karnataka | 560036 | 103 | engineer |
| andhra | 560036 | 104 | trainee |

Create a DynamoDB tabe.  
  
Insert data into the table.  
  
  
Create global secondary index.  
  
  
Post creation of index.  
  
  
  
22. Scan vs Query API call.  
- Two different ways of getting the information out of the database.  
  
- Query : A query operation finds items in a table based on the Primary key attribute (and also a distinct value to search for). All the attributes and their values of the item(document) would come up as the query result. We can use an optional sort key ‘name and value’ to refine the results.  
- By default the query returns all the attributes for the items, but you can use the ‘**ProjectionExpression’** parameter if you want the query to only return the specific attributes you want.  
- Results are always sorted by the sort key and in ascending order by default.  
- And the order can be reversed by using the ‘**ScanIndexForwardParameter**’ to false.(It applies only to queries)  
- By default all queries are eventually consistent and you need to explicitly set the query to be strongly consistent.  
- Query is more efficient.  
  
- Scan : A scan operation examines every single item in the table.  
- It returns all the data attributes by default.  
- And you can use the ‘**ProjectionExpression**’ parameter to refine the scan results to only return the attributes you want.  
  
  
lab : Scan vs Query  
  
Scan is not searching the database. It is fetching the whole data and filtering the data.  
  
  
  
How to improve performance : Set a page size of 40, which uses fewer read operations. Set the page size to return 40 items. But large number of smaller operations will have to performed but this will reduce throttling.  
  
23 : Provisioned Throughput.(Provisioned Capacity)  
- It is a mechanism that we use to define the capacity and performance requirements of DynamoDB.  
- Provisioned throughput is measured is Capacity units (CU).  
- When creating a table and when you are not using autoscaling, We need to specify our requirements in Read CU and Write CU.  
- 1 Write Capacity Unit = 1 \* 1KB Write per Second  
- 1 Read Capacity Unit = 1 \* Strongly Consistent Read of 4KB per second.  
 OR   
 2 \* Eventually Consistent Reads of 4KB per second (default).  
  
Ex : Let say we have a table of 5 \* Read capacity and 5 \* Write Capacity Units.  
This configuration will be able to perform : 5 \* 1KB per second writes  
 5 \* 4KB Strongly Consistent Reads per second  
 5 \* 2 \* 4KB Eventually Consistent Reads per second.  
  
Ex :If you have an application which needs to read 80 rows per second + each item is 3KB in size.  
- Size of each item/ 4KB   
- ¾ Kb = 1 Kb rounded to the nearest whole number.  
- So, Each read is going to need 1 Read capacity unit per read operation.  
- The above value multiplied by the no of reads per second = 80\*1 = 80 for Strongly Consistent RCU   
and 40 for Eventually Consistent RCU.  
  
Ex : If we want to write 100 items per second and each item is 512 bytes in size.  
- Calculate how many Capacity Units for each write.  
- size of each item / 1KB for WCU.  
- 512bytes / 1KB = 0.5 = 1 .  
- Rounded up to the nearest whole number, each write will need 1 WCU per write operation.  
- This result will have to be multiplied by the no of writes per second = 100 Write capacity units are needed.

24 : DynamoDB On-Demand Capacity Pricing option.  
  
In addition to the provisioned throughput model seen above, There is also a ‘On-Demand Capacity’ pricing model is also there in DynamoDB. We don’t have to specify our capacity requirements. DynamoDb will automatically Scale up and Scale Down. This is good for Unpredictable work loads, Where you have a new application / or where you have short lived spikes. Pay per request model.  
  
25 : DAX / DynamoDB Accellarator. - Designed to work only with DynamoDB.  
DAX : It is a fully managed clustered in-memory cache for DynamoDB.  
- We only get a massive **READ** performance which is 10 times more.  
- Used in Black Friday promotions.  
  
How does it work : DAX is a **write-through** cache service. This means that data is written to the cache as well as the backend store at the same time.  
We can also point our API calls to the DAX cluster. Instead of having our application query DynamoDB directly it will try and query the DAX cluster, If the item we are looking for is in the cache i.e a ‘cache hit’, DAX returns the result. If there is a ‘cache miss’ DAX performs a **eventually consistent getitem operation** against DynamoDB, And writes it into the cache and return the item to the client.  
We can also save money by reducing the provisioned read capacity units.

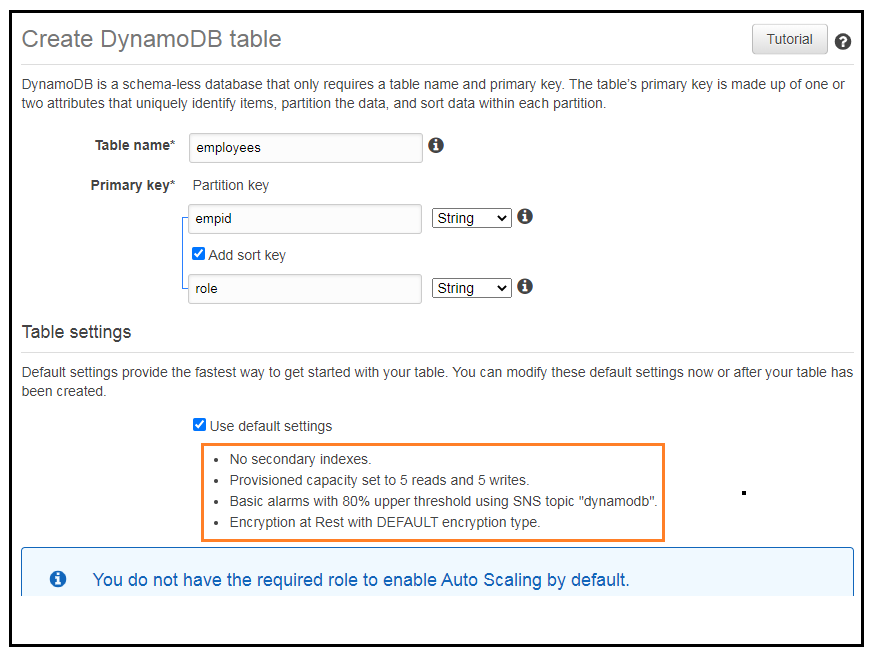
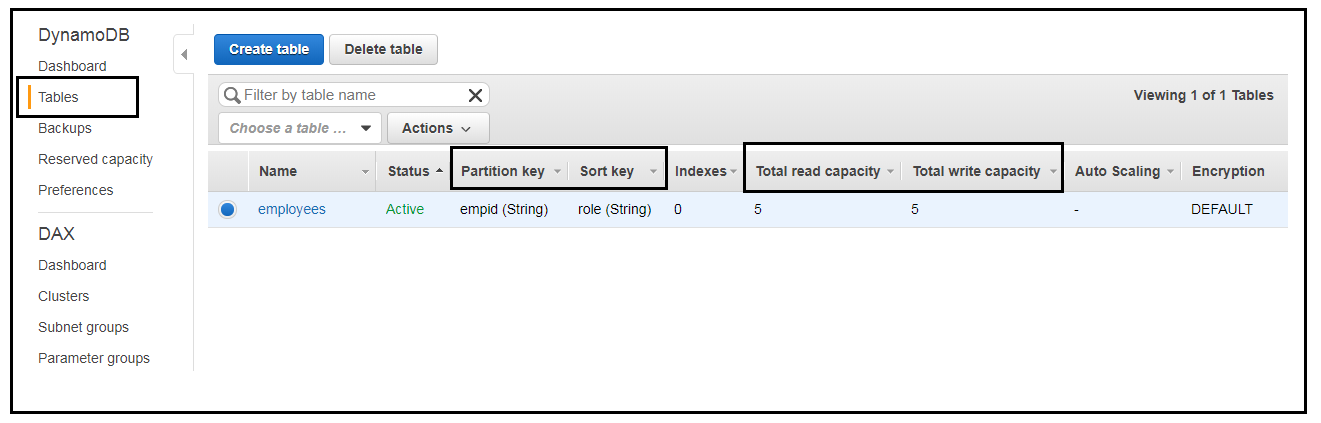
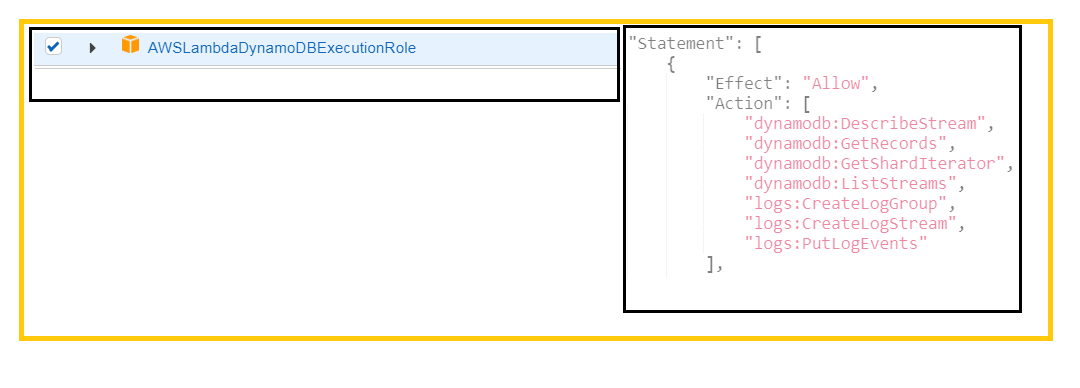
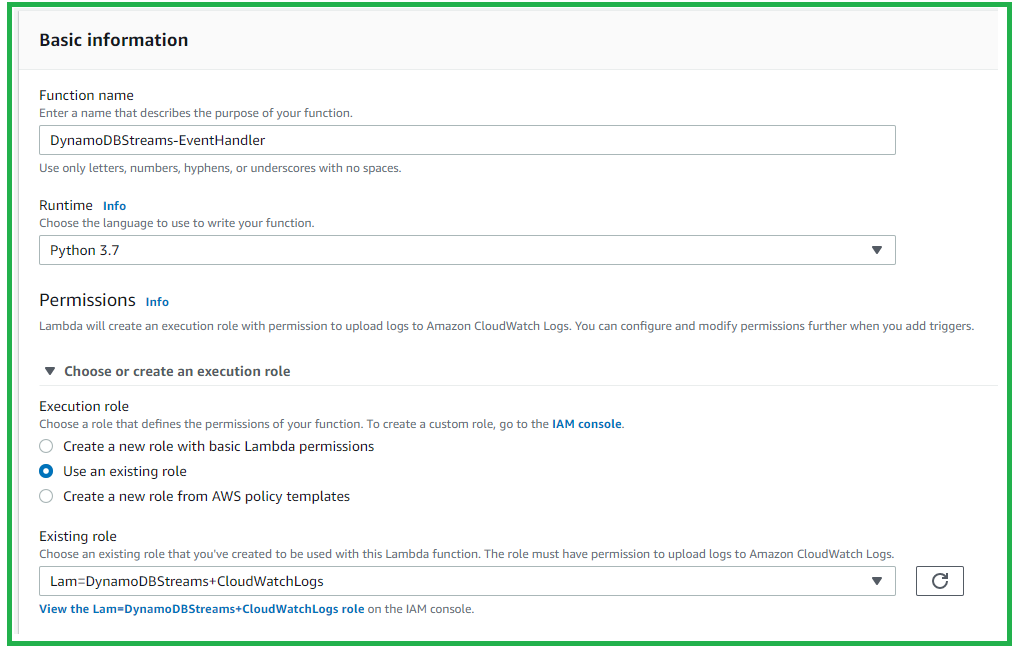
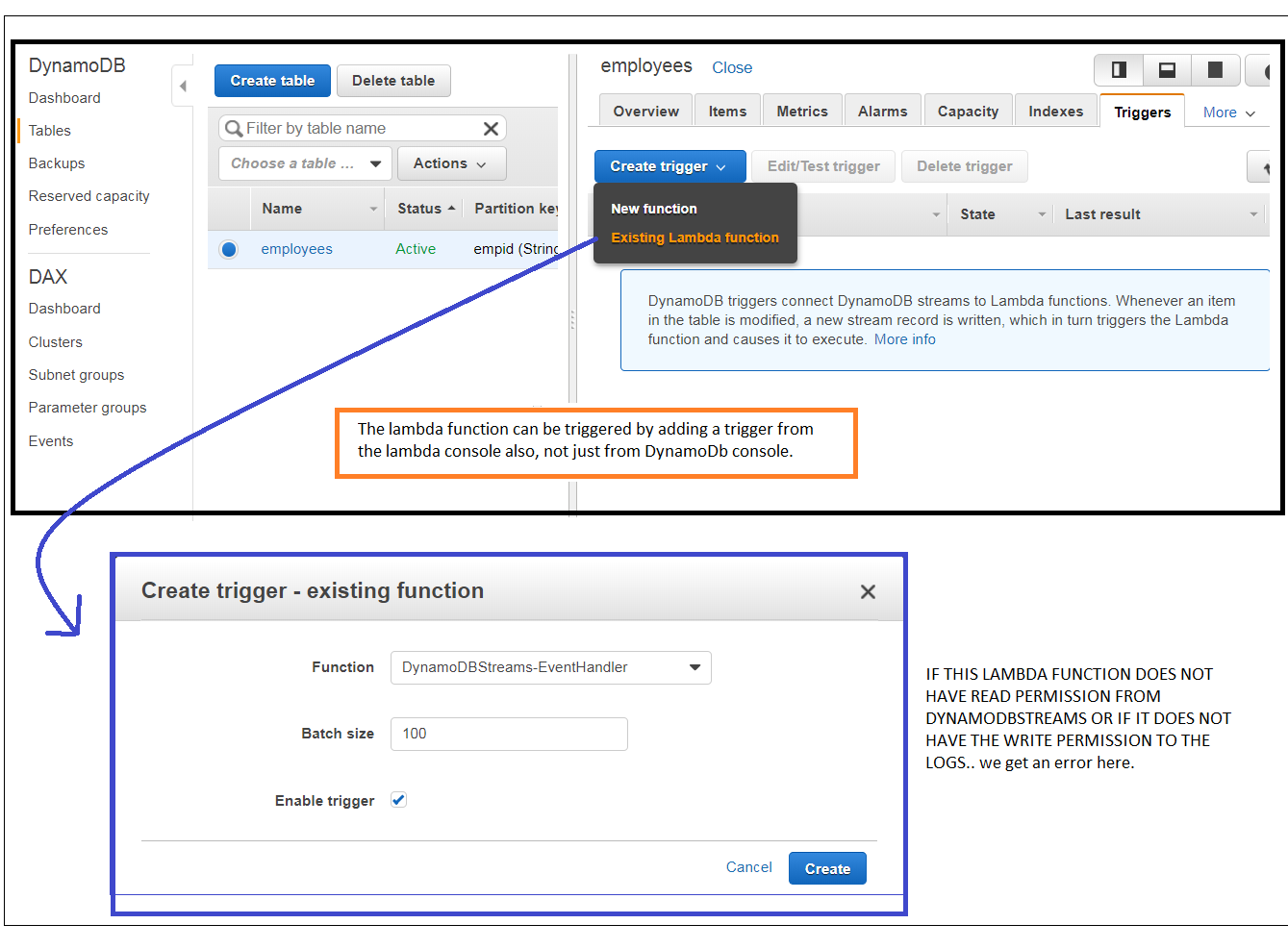
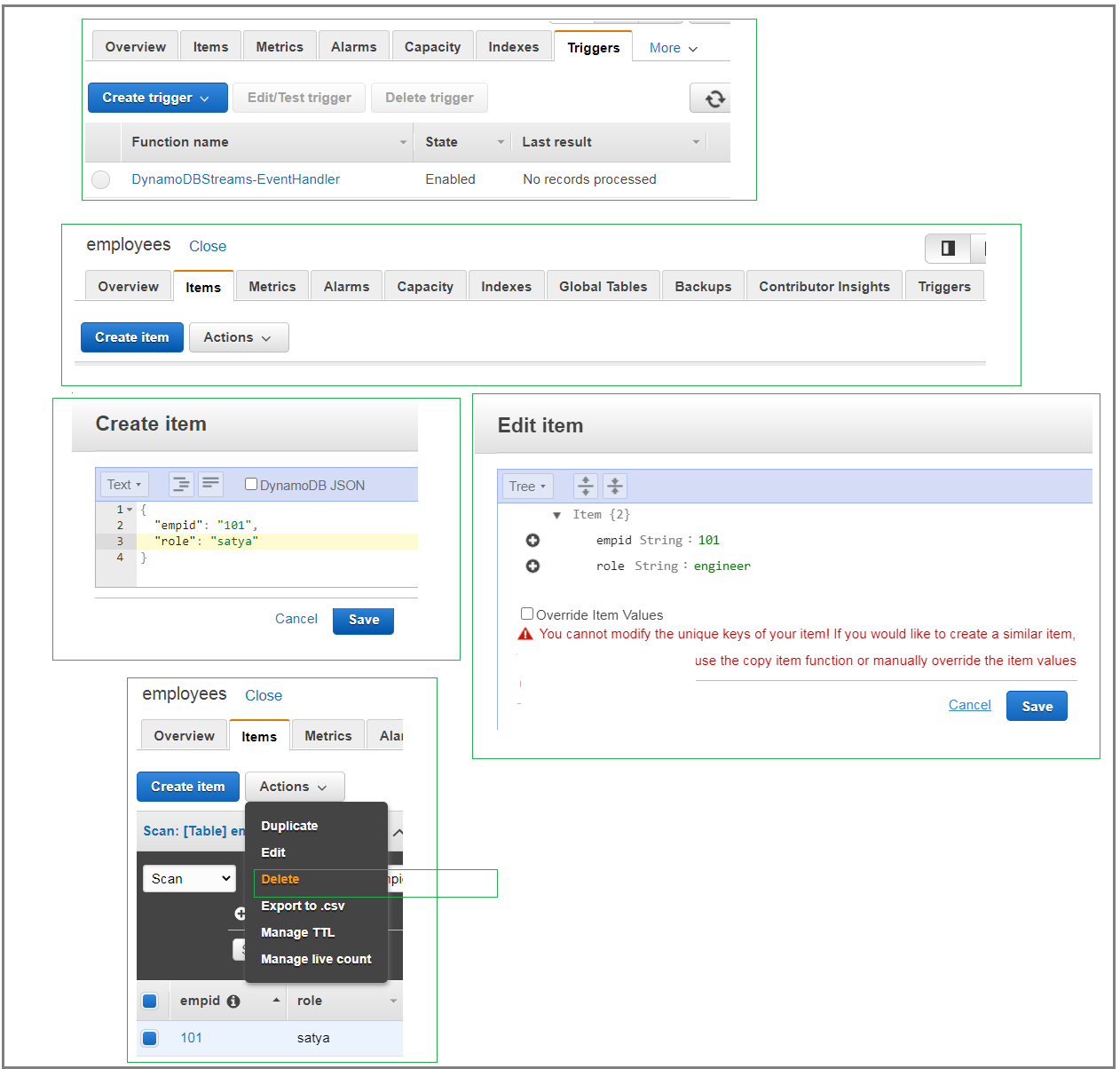
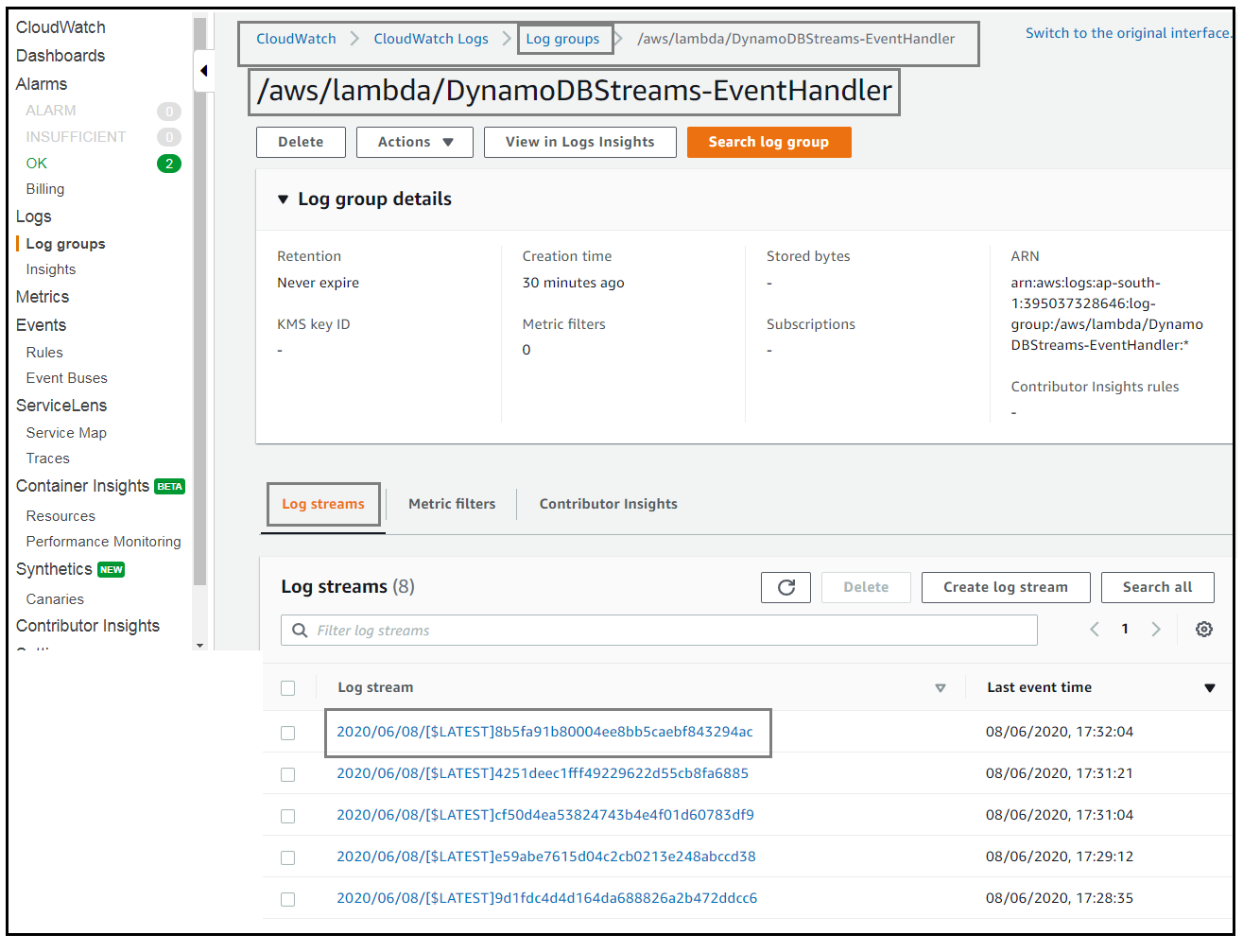
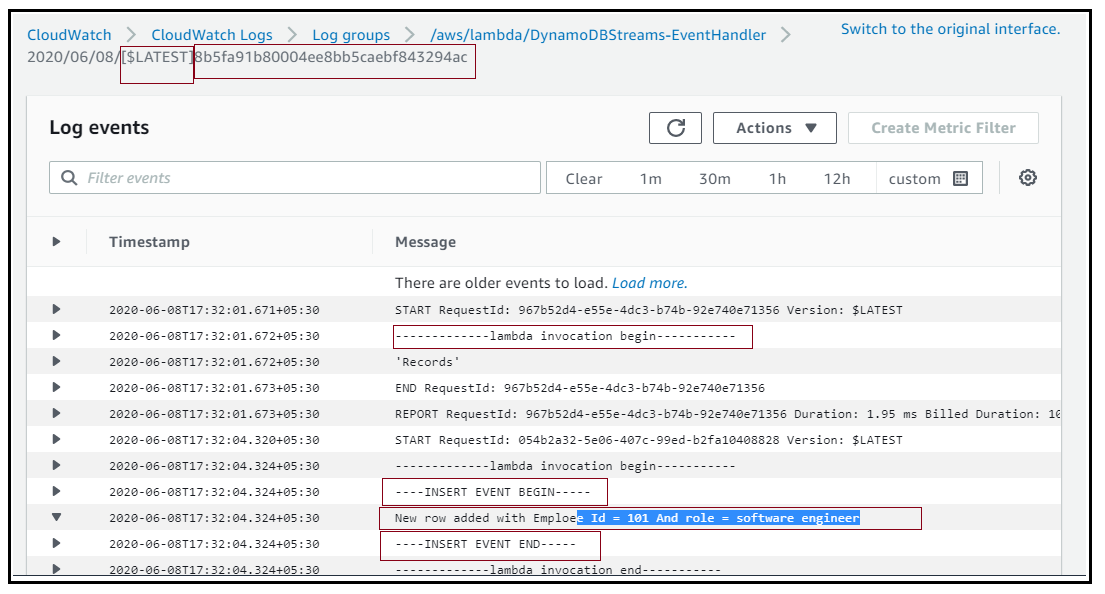
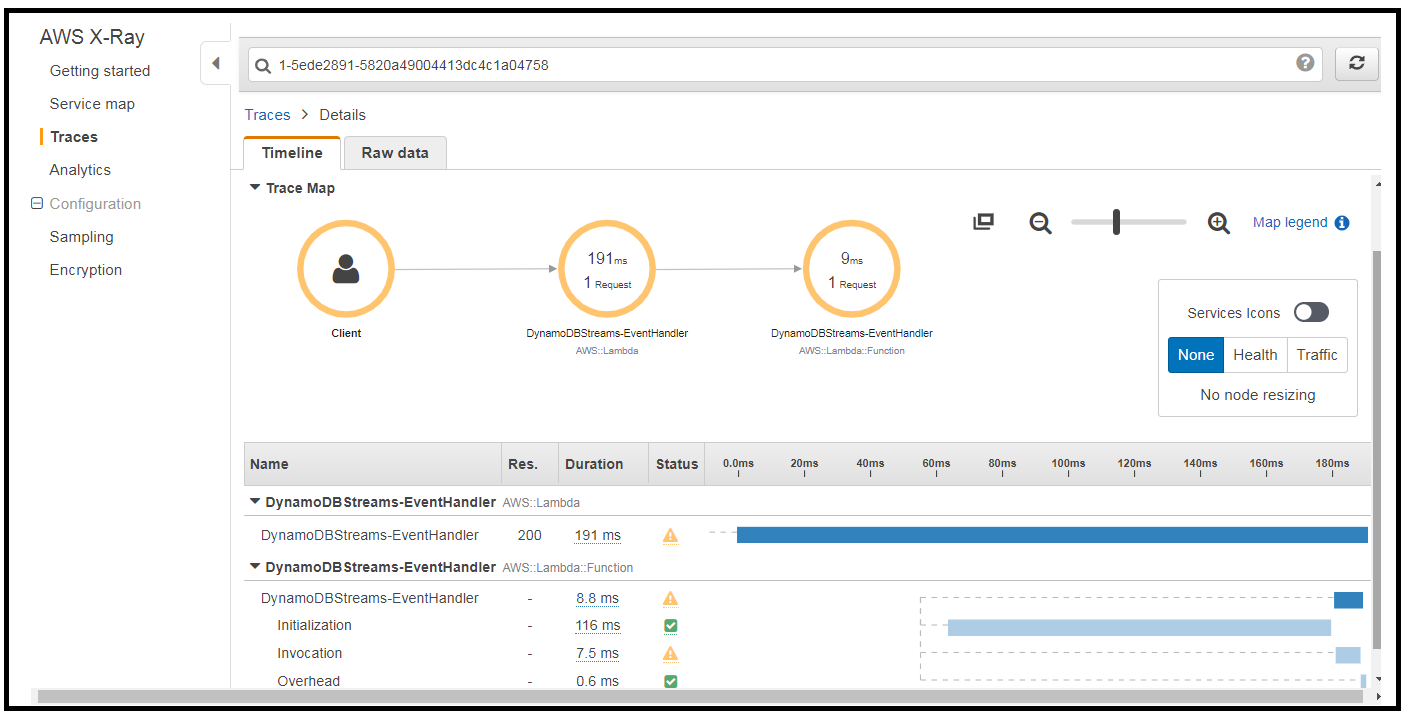
26. **Elasticache**.  
  
  
Elasticache : It is a in memory cache in the cloud. It is used to improve performance of web applications, Allowing us to retrieve information from fast in-memory cache instead of from a slower disk based database.  
- It sits between your application and your database.  
- When requesting best selling products on a platform.  
- Its really good if your database is read heavy and also if the data is not changing frequently.  
- Your frequently accessed data is stored in cache for low latency access, thus improving your application performance.  
- This is also good for compute heavy workloads- Ex, Recommendation engines.   
  
26.1 Types of Elasticache – can use with RDS.  
  
Memcached : This is multi threaded, But it does not have Multi AZ capabilities.  
  
Redis : open source + in memory Key-Value store.  
- It supports more complex data structures like sorted sets and lists.  
- It supports Master/slave replication and Multi-AZ for Cross AZ redundancy.

Caching Strategie.  
  
Lazy-Loading : Loads the data in to the cache only when necessary. If requested data is in the cache, Elasticache returns the data to the application. If the data is not there in the cache or if it has expired Elasticache returns null. Then the application fetches the data from the database and writes the data received into the cache so that it is available next time.  
**Advantages** : Only the requested data is cached and it avoids filling up the cache with useless data that never gets requested. And a node failure means there will initially just be cache misses.  
**Disadvantages** : Cache miss penalty. The data should be written to cache for the first time. Data can become stale. If data is only updated if there is a cache miss data can become out of date. And if data in the database changes, the cache is not automatically updated.  
**TTL :** Within Elasticache we can add a time to live data. This specifies the number of seconds until the key/data expires to avoid keeping stale data. Lazy loading will treat the expired key as a cache miss and will go and fetch the data from the database and subsequently write the data to the cache with a new TTL.  
Write Through : This strategy adds or updates data to the cache whenever data is written to the databse.

**Advantage** : data in the cache is never stale.   
**Disadvantages** : involves a write penalty. Because every write involves a write to the database as well as to the cache.  
Note : Users are generally more tolerant of additional latency when updating data, Than when retrieving it. I.e Write penalty could not be such a big disadvantage.  
If a node fails and a new node is spun up, data in the cache will be missing until it is added or updated in the database. [This can be mitigated by implementing ‘lazy loading’ in conjunction with ‘write-through’]  
- you end up with wasted resources if most of your cache data is never read.  
  
  
27. DynamoDB Transactions.  
- DynamoDB transactions are there to support mission critical applications which need all or nothing approach.  
- ACID : Atomic, consistent, Isolated, Durable. – ideal properties of a database transaction.  
- Financial transactions.

28. DynamoDB TTL.  
- TTL : it is an **attribute** which defines an expiry time for your data.  
- Expired items are marked for deletion and will be deleted with in the next 48 hours(So we need to filter out expired items from the queries and scans).  
- session data.

29. DynamoDB Streams.  
- Dynamo DB Stream is a time ordered sequence/stream and it records any modifications made to the items in your Dynamo DB table.   
- Insert / update / delete operations are recorded in the DynamoDB stream.  
- All of these actions are recorded in the form of logs which are encrypted at rest and stored for 24 hours only.  
- Used in Auditing and Archiving of transactions . Replaying any transactions to a different table.  
- They are mainly used to trigger events based on a particular change with the Dynamo DB table, So we can use DynamoDB streams to trigger a Lambda function based on a change that is done to a table.  
- There are two separate end points to access the DynamoDB table itself and other endpoint for accessing the Dynamo DB stream.  
- We can capture the state of the item before and after change as well.

30.Provisioned Throughput Exceeded Exception and Exponential Backoff.++  
- ProvisionedThroughputExceededException : You would see this if your request rate is too high for the read and write capacity provisioned on the dynamo DB table.  
- If aws SDK is used, it will automatically retry till they are successful.  
- If you are not using the SDK, we will have to configure one or both of the following things.  
: **Reduce request frequency** – reduce the number of ‘concurrent requests / frequency’ that are being sent to the DynamoDB table.  
: **Use Exponential Backoff** – All aws SDK’s will have a simple retry functionality, additionally they also use exponential backoff. Which means the requester will implement progressively longer waits between consecutive retries of a failed request. 50ms after 1st failed request before submitting the request again.. later 100ms.. and later 200ms   
  
  
31. LAB : DynamoDB Streams to Lambda Tutorial.  
We will learn how to detect change in a DynamoDB table- using a lambda function.  
We shall be logging to **cloud-watch** to check everything is working correctly.  
  
31.1 Create a DynamoDB table.  
  
The created table looks like below.  
  
  
31.2   
Before creating Lambda function create a role and attach the below policy so that Lambda can ‘read’ DynamoDB Streams data and ‘write’ logs to cloudwatch.  
  
31.3  
Create a lambda function and use the existing role.  
  
  
31.4 Code for the lambda function to print traces  
  
  
  
31.5   
Add a DynamoDB Trigger to call the lambda function  
  
  
31.6   
After creating the trigger make a few changes to the database table by inserting and deleting records.  
  
  
31.7 Cloud Watch Console.  
  
  
  
31.8   
If you have enabled Xray tracing in the Lambda window. We can see the below TraceMap.  
  
  
31.8 LAMBDA MONITORING  
