Task 01:

In AWS CLI, how would you retrieve only the name and email fields of an item with primary key UserID = 123?

1. aws dynamodb get-item --table-name Users --key '{"UserID":{"S":"123"}}' --attributes 'name,email'

2. aws dynamodb scan --table-name Users --filter-expression 'UserID = :id' --projection-expression 'name,email'

3. aws dynamodb get-item --table-name Users --key '{"UserID":{"S":"123"}}' --projection-expression "name,email"

4. aws dynamodb fetch --table-name Users --key '{"UserID":"123"}' --fields name,email

Task 02:

What is a key advantage of using AWS SDK for interacting with DynamoDB over AWS CLI or Console?

1. The SDK bypasses IAM restrictions by using service-linked roles.

2. SDK supports only high-level batch operations and does not require explicit commands.

3. SDK allows embedding retry logic, pagination handling, and structured data models into application logic.

4. SDK operations are faster because they interact directly with the DynamoDB core engine, not via API endpoints.

Task 03:

When interacting with DynamoDB using AWS Console, how can a user filter data in a table view without writing queries?

1. Use the “Visual Query Builder” to design SQL-like filters dynamically.

2. Switch to JSON editor and apply scan expressions manually.

3. Apply client-side filters using attribute filters under the “Explore Table Items” tab.

4. Enable DynamoDB Streams and subscribe to item change logs.

Task 04:

What does the DynamoDB Architecture primarily rely on to achieve high availability and fault tolerance?

1. Vertical scaling of a single powerful database node.

2. Distributed design with synchronous replication across at least three availability zones.

3. Caching layer backed by Amazon CloudFront to serve all reads globally.

4. Peer-to-peer replication network within a single AZ that dynamically promotes nodes.

Task 05:

What is the result of enabling On-Demand Capacity Mode in DynamoDB?

1. The table disables all autoscaling and locks throughput at the default level.

2. You are billed based only on actual reads and writes, with no need to provision capacity units.

3. Each write incurs a double charge due to lack of predictable scaling.

4. You must migrate to a new table, as switching from provisioned to on-demand is not supported.

Task 06:

How does the AWS CLI handle table creation in DynamoDB, and what is required when setting provisioned throughput?

1. The CLI automatically detects workload and configures read/write capacity based on item size and data type.

2. The CLI requires explicit specification of read and write capacity units during table creation, with optional auto-scaling flags.

3. When using AWS CLI, capacity settings are managed solely by IAM roles and cannot be set during creation.

4. CLI operations only support on-demand capacity creation and cannot be used for provisioned tables.

Task 07:

When using the AWS Console to add an item to a DynamoDB table, which of the following best reflects the behavior of attribute typing?

1. All attributes must be explicitly declared and typed beforehand at the table schema level.

2. Each attribute is typed at runtime during data entry and is stored with metadata to enforce consistent typing for future writes.

3. Attributes can be added dynamically, with their type inferred at the time of input without enforcing a strict schema.

4. AWS Console supports only string types by default and requires SDK or CLI for other types.

Task 08:

How are partition keys critical to DynamoDB's scalability model?

1. They serve as encryption keys for secure storage and access control at the item level.

2. They determine the maximum read and write throughput for individual table rows by distributing load across shards.

3.They influence how data is logically grouped and retrieved but do not impact underlying storage distribution.

4. They are used to map table items to partitions which in turn directly influence the system’s horizontal scalability and access latency.

Task 09:

primary functional difference between a Local Secondary Index (LSI) and a Global Secondary Index (GSI)

1. LSIs replicate the table to a different region while GSIs replicate only selective attributes.

2. LSIs require the same partition key as the base table, whereas GSIs can have a different partition key.

3. LSIs provide faster write performance than GSIs because they are asynchronously updated.

4. GSIs allow duplicate sort keys across different partition keys, while LSIs require sort key uniqueness.

Task 10:

NOT required when creating a table using the AWS Console

1. Defining the primary key, which may consist of a partition key and optionally a sort key.

2. Specifying an IAM role for automatic throughput scaling.

3. Choosing between on-demand and provisioned capacity modes. 4. Setting read and write capacity units if provisioned mode is selected.

Task 11:

Which CLI command can be used to enable Point-in-Time Recovery (PITR) on a table?

1. aws dynamodb enable-pitr --table-name Customers

2. aws dynamodb update-table --table-name Customers --pitr true

3. aws dynamodb update-continuous-backups --table-name Customers --point-in-time-recovery-specification PointInTimeRecoveryEnabled=true

4. aws dynamodb backup-table --table-name Customers --pitr-enabled

Task 12:

Which key metric should you monitor in CloudWatch to detect hot partitions in a DynamoDB table?

1. ConsumedWriteCapacityUnits grouped by PartitionID

2. ThrottledRequests with partition key dimension

3. AccountProvisionedThroughputUtilization across all tables

4. PartitionSplitCount from capacity forecast logs

Task 13:

What is the difference in capacity planning between a base table and a GSI in DynamoDB?

1. GSIs always share capacity units with the base table unless explicitly partitioned.

2. GSIs can be configured with separate provisioned or on-demand capacity settings from the base table.

3. GSIs cannot be provisioned independently and only inherit capacity metrics.

4.GSIs only consume read units, while base tables consume both read and write units.

Task 14:

true regarding Global Secondary Indexes (GSI) write behavior

1. Writes to GSIs are immediately consistent with the base table but slower due to background processing.

2. Write operations to the base table automatically update GSIs asynchronously, which may introduce temporary inconsistencies.

3. GSIs enforce the same throughput as the base table and cannot be scaled independently.

4. GSIs must be written to explicitly using SDK or CLI and do not support automatic updates from base table changes.

Task 15:

What is a key characteristic of DynamoDB’s eventual consistency model for read operations?

1. It ensures all replicas return stale data initially and synchronize only upon write success.

2. It guarantees real-time updates across all regions before responding to read queries.

3. It may return stale data immediately after a write, but will eventually return the latest version.

4. It always reads from a replica instead of the primary node to reduce latency.

Task 16:

What does the DynamoDB Architecture primarily rely on to achieve high availability and fault tolerance?

1. Vertical scaling of a single powerful database node.

2. Distributed design with synchronous replication across at least three availability zones.

3. Caching layer backed by Amazon CloudFront to serve all reads globally.

4. Peer-to-peer replication network within a single AZ that dynamically promotes nodes.

Task 17:

How would you create a table using AWS CLI with on-demand billing mode?

1. aws dynamodb create-table --table-name Inventory --key-schema ... --billing PAY\_PER\_USE

2. aws dynamodb create-table --table-name Inventory --key-schema ... --billing-mode ON\_DEMAND

3. aws dynamodb create-table --table-name Inventory --key-schema ... --billing-mode PAY\_PER\_REQUEST

4. aws dynamodb new-table Inventory --billing on-demand

Task 18:

You want to create a new item using AWS CLI in the Employees table. Which is the correct command syntax?

1. aws dynamodb put-item --table-name Employees --item '{"ID":{"N":"1001"}, "Name":{"S":"John"}, "Role":{"S":"Manager"}}'

2. aws dynamodb add-item --table Employees --values '{"ID":1001, "Name":"John", "Role":"Manager"}'

3. aws dynamodb create --table Employees --item '{"ID":1001, "Name":"John", "Role":"Manager"}'

4. aws dynamodb insert --table-name Employees --data '{"ID":{"N":"1001"}, "Name":{"S":"John"}, "Role":{"S":"Manager"}}'

Task 19:

Why is a good partition key design critical in DynamoDB?

1. It controls the table's storage format and indexing strategy across all regions.

2. Poorly chosen partition keys can lead to uneven data distribution, causing hot partitions and throttled access.

3. Partition keys define data retention policies and are used to automate lifecycle configurations.

4. Partition key design impacts the encryption method used for server-side encryption.

Task 20:

Which feature in the DynamoDB Console allows you to monitor read/write throughput over time?

1. Throughput Inspector Panel under the "Table Schema" tab.

2. Metrics tab that integrates with Amazon CloudWatch for real-time performance charts.

3. "Settings" section with provisioning history logs and capacity usage reports.

4. The "Indexes" tab, which includes capacity monitoring for all associated GSIs.

Task 21:

Which CLI command can be used to enable Point-in-Time Recovery (PITR) on a table?

1. aws dynamodb enable-pitr --table-name Customers

2. aws dynamodb update-table --table-name Customers --pitr true

3. aws dynamodb update-continuous-backups --table-name Customers --point-in-time-recovery-specification PointInTimeRecoveryEnabled=true

4. aws dynamodb backup-table --table-name Customers --pitr-enabled

Task 22:

a valid best practice when designing a high-throughput DynamoDB table for variable workload access patterns

1. Use composite keys that include highly variable timestamp-based components to create uniform partition distribution.

2. Store frequently accessed items under a single partition key to improve read latency for those items.

3. Choose a partition key that aligns with your most frequent filter condition to reduce read units.

4. Use small item sizes and avoid indexes to minimize storage costs for unpredictable read-heavy workloads.

Task 23:

What does DynamoDB Limits – Error Handling suggest when facing a ProvisionedThroughputExceededException?

1. Upgrade to Enterprise Plan to avoid soft limits.

2. Pause writes for 10 minutes to allow throttling to reset.

3. Back off and retry with exponential delays, while monitoring consumption via CloudWatch.

4. Rebuild the table with smaller item sizes to reduce partition pressure.

Task 24:

Which AWS CLI command correctly performs a conditional delete to remove an item only if its status attribute is "inactive"?

1. aws dynamodb delete-item --table-name Users --key '{"UserID":{"S":"123"}}' --condition-expression "status = :s" --expression-attribute-values '{":s":{"S":"inactive"}}'

2. aws dynamodb delete-item --table-name Users --key '{"UserID":{"S":"123"}}' --condition "status = :inactive" --expression-attribute-values '{":inactive":{"S":"inactive"}}'

3. aws dynamodb delete-item --table-name Users --key '{"UserID":{"S":"123"}}' --where status="inactive"

4. aws dynamodb remove-item --table-name Users --key '{"UserID":{"S":"123"}}' --if status equals inactive

Task 25:

Which command in AWS CLI updates email attribute for a user in Users table without overwriting the entire item?

1. aws dynamodb put-item --table-name Users --item '{"UserID":{"S":"u101"}, "email":{"S":"user@example.com"}}'

2. aws dynamodb update-item --table-name Users --key '{"UserID":{"S":"u101"}}' --update-expression "SET email = :e" --expression-attribute-values '{":e":{"S":"user@example.com"}}'

3. aws dynamodb patch-item --table-name Users --key '{"UserID":{"S":"u101"}}' --data '{"email":"user@example.com"}'

4. aws dynamodb set-field --table-name Users --field email --value user@example.com

Task 26:

How can you create a table with LSI using AWS CLI?

1. aws dynamodb create-table --table-name Products --key-schema ... --local-secondary-indexes '[{...}]' --attribute-definitions ...

2. aws dynamodb create-lsi Products ProductTypeIndex

3. aws dynamodb create-table --name Products --lsi '[{"Name":"ProductTypeIndex"}]'

4. aws dynamodb update-table Products --add-local-index ProductTypeIndex

Task 27:

You want to create a new item using AWS CLI in the Employees table. Which is the correct command syntax?

1. aws dynamodb put-item --table-name Employees --item '{"ID":{"N":"1001"}, "Name":{"S":"John"}, "Role":{"S":"Manager"}}'

2. aws dynamodb add-item --table Employees --values '{"ID":1001, "Name":"John", "Role":"Manager"}'

3. aws dynamodb create --table Employees --item '{"ID":1001, "Name":"John", "Role":"Manager"}'

4. aws dynamodb insert --table-name Employees --data '{"ID":{"N":"1001"}, "Name":{"S":"John"}, "Role":{"S":"Manager"}}'

Task 28:

Which command correctly retrieves an item using AWS CLI and returns only consistent reads?

1. aws dynamodb get-item --table-name Users --key '{"UserID":{"S":"1"}}' --return-consistent true

2. aws dynamodb get-item --table-name Users --key '{"UserID":{"S":"1"}}' --consistent-read

3. aws dynamodb fetch --table-name Users --key '{"UserID":{"S":"1"}}' --strong-read true

4. aws dynamodb get-item --table-name Users --key '{"UserID":{"S":"1"}}' --consistent-read true

Task 29:

Using AWS CLI, how can you delete a table named LogsTable?

1. aws dynamodb delete-table --name LogsTable

2. aws dynamodb remove-table LogsTable

3. aws dynamodb delete-table --table-name LogsTable

4. aws dynamodb drop LogsTable

Task 30:

You want to project only title and views attributes when querying a GSI CategoryIndex on table Articles. Which CLI parameter should be used?

1. --attribute-names "title,views"

2. --filter-expression "Category = :cat" --return-fields "title,views"

3. --project-fields title,views

4. --projection-expression "title, views"

Task 31:

Which of the following statements best describes the concept of "Provisioned Capacity" in DynamoDB?

1. It allows you to pay for the resources based on the actual read and write throughput consumed, regardless of your provisioned settings.

2. It enables automatic scaling of capacity units without manual provisioning, optimizing costs under highly variable workloads.

3. It is only applicable when Global Secondary Indexes are used, and not for base table throughput management.

4. It requires predefining read and write throughput capacity, which is reserved and billed regardless of actual usage, and is suitable for predictable workloads.

Task 32:

In DynamoDB, what does the Global Secondary Index (GSI) allow you to do that Local Secondary Indexes (LSI) cannot?

1. GSIs allow using an alternate partition key and sort key on the same table without size constraints linked to the base item.

2. GSIs enforce uniqueness constraints on attributes that are not part of the primary key.

3. GSIs support transactions across multiple AWS regions, while LSIs do not.

4. GSIs are always consistent with the base table, whereas LSIs are eventually consistent.

Task 33:

Which operation is used in AWS SDK to retrieve only a specific set of attributes from a DynamoDB item?

1. Set the ProjectionExpression to list required attributes during a GetItem or Query request.

2. Configure ItemAttributeSelector in SDK settings for attribute filtering during retrieval.

3. Modify ReturnAttributes in the table schema configuration to enforce item projection.

4. Use PartialAttributeMode=true in the Scan operation to return selected attributes.

Task 34:

How does AWS CLI handle item-level updates in DynamoDB?

1. It requires specifying the entire item in update command and does not support partial attribute updates.

2. It allows updating only specific attributes using --update-expression, supporting conditional writes and atomic counters.

3. AWS CLI supports batch update operations only and does not work with individual items.

4. Attribute updates using CLI are available only when global tables are enabled.

Task 35:

In DynamoDB, how does eventual consistency differ from strong consistency when reading data?

1. Eventual consistency requires provisioned capacity mode, while strong consistency is available only in on-demand mode.

2. Eventual consistency reads offer reduced latency but may not reflect the most recent successful write operations to an item.

3.Eventual consistency guarantees that any read returns the latest committed value across all replicas.

4.Strong consistency introduces unpredictable delays due to constant synchronization with global tables.

Task 36:

when creating a table in AWS Console whats not required?

Defining the Primary Key may consist of a partition key and optionally a sort key

Specifying IAM role for automatic throughput scaling

Choosing between on demand and provisioned capacity modes

Setting read and write capacity units if provisioned mode is selected

Task 37:

When executing a Scan operation from the AWS CLI. which  flag controls the number of items returned per API call?

–scan-limit

–max-items

–limit

–page-size

Task 38:

What does DynamoDB Limits- error Handling suggest when facing a provisionedthroughputExceededException?

Upgrade to Enterprise plan to avoid soft limits

Pause writes for 10 minutes to all throttling to reset

Back off and retry with exponential delays, while monitoring consumption via cloudWatch.

Rebuild the table with smaller item sizes to reduce partition pressure

Task 39:

List all tables in your DynamoDB AWS CLI.

Aws dynamodb list-all-tables

Aws dynamodb list-tables

Aws dynamodb get-tables

Aws dynamodb show-tables

Task 40:

What is the correct way to use an AWS CLI scan operation with a filter to get all items where status = "active"?

1. aws dynamodb scan --table-name Accounts --scan-filter '{"status":{"AttributeValueList":[{"S":"active"}],"ComparisonOperator":"EQ"}}'

2. aws dynamodb scan --table-name Accounts --filter-expression 'status = "active"'

3. aws dynamodb filter --table-name Accounts --expression 'status = :s' --values '{":s":"active"}'

4. aws dynamodb read --table Accounts --where status="active"

TASK 41:

Task 42:

Task 43:

In the AWS CLI, what does the --key-condition-expression parameter do in a Query operation?

1. It defines the capacity unit limit to avoid throttling during heavy queries.

2. It specifies filter criteria on non-key attributes to reduce query results post-execution.

3. It sets up primary key constraints to filter results on the partition and optional sort key.

4. It configures response structure to include indexes and key metadata.

Task 44:

what is the role of an ExpressionAttributeValues parameter when using the AWS SDK to perform conditional writes?

1. It defines the format in which data will be returned post-write for validation purposes.

2. It is used to reference literal values safely in expressions, avoiding reserved word conflicts.

3. It sets the default return type for write operations in strongly typed programming languages.

4. It acts as an authentication token for validating write access for conditional expressions.

Task 45:

Which AWS SDK technique helps ensure that a write only occurs if the item doesn't already exist?

1. Use UpdateItem with SET clause and SkipOnDuplicate=true flag.

2. Apply ConditionExpression to check attribute absence using attribute\_not\_exists().

3. Set overwrite=false in the item metadata section.

4. Wrap the write in a loop to detect duplication via GetItem beforehand.

Task 46;

What is Atomic counter?

Plz note:

Atomic counters in Amazon DynamoDB provide a mechanism to consistently increment or decrement a numerical attribute within an item, even under high concurrency. This is achieved through the UpdateItem API operation and its use of update expressions.

Task 47:

What is the purpose of Atomic Counters in DynamoDB when using the AWS SDK?

1.They allow concurrent updates on a numeric attribute using conditional expressions that ignore previous values.

2.They enable real-time monitoring of write throughput consumption per partition.

3.They allow increment or decrement operations on numeric attributes without needing to perform a read-modify-write cycle manually.

4.They create an additional index that tracks numeric field changes across table items.

Task 48:

Do you know about ?

Pagination Reads in general … I hope you do…

Task 49:

Now after doing Task 48.. Can you answer

"What does this AWS SDK paginated read pattern accomplish?

while 'LastEvaluatedKey' in response: response = table.scan(ExclusiveStartKey=response['LastEvaluatedKey'])"

1. reads only a single item on each iteration.

2. ensures the scan() function only runs once and then halts.

3. paginates through all the items in the table by using the last key from the previous scan.

4. repeats the last item read until a manual interrupt is triggered.

Task 50:

The command that retrieves all items from a DynamoDB table while handling pagination using AWS CLI.

1. aws dynamodb get-all-items --table-name Products

2. aws dynamodb scan --table-name Products and check for LastEvaluatedKey in response for further scans

3. aws dynamodb retrieve --table-name Products --all-pages true

4. aws dynamodb fetch-items --table-name Products --limit max

Task 51:

When reading a paginated result set using the AWS SDK from DynamoDB, what mechanism allows you to retrieve the next set of items?

1. Enable the DynamoCursor class in the SDK, which automatically detects batch size and fetches data until EOF.

2. Use the NextToken from the current response and pass it as the StartKey in the subsequent request.

3. Enable stream reading and use shard iterators to continue reading in paginated mode.

4. Set the pageSize parameter to 0, which automatically triggers iterative fetching.

Task 52:

What AWS SDK feature helps in retrieving paginated data from a table or index with multiple result pages?

1. Enable AutoPagination=true in SDK configuration settings to automatically scroll results.

2. Use the PageResultToken from the response and pass it to StartFromPage in the next call.

3. Use the LastEvaluatedKey from the response and pass it as ExclusiveStartKey in the next query.

4. Set NextCursor in the response as the new PaginationCursor for loop continuation.

Add ons:

Task 53:

Juz FYI

What is Boto3:

Boto3 is the official Amazon Web Services (AWS) Software Development Kit (SDK) for Python. It enables Python developers to create, configure, and manage AWS services through Python code. Boto3 is developed and maintained by AWS.

Task 54:

How would you use Boto3 to query the Orders table by CustomerID, using a secondary index named CustomerIndex and retrieve only OrderID and Amount attributes?

1. table.get\_item(Key={'CustomerID': customer\_id}, ProjectionExpression="OrderID,Amount", IndexName="CustomerIndex")

2.  table.query(KeyConditionExpression=Key('CustomerID').eq(customer\_id), ProjectionExpression="OrderID,Amount", IndexName="CustomerIndex")

3. dynamodb.query(TableName="Orders", KeyCondition={'CustomerID': {'S': customer\_id}}, Index="CustomerIndex", ReturnFields="OrderID,Amount")

4. table.scan(FilterExpression="CustomerID = :id", ExpressionAttributeValues={":id": customer\_id}, Attributes="OrderID,Amount", IndexName="CustomerIndex")

Task 55:

What is the correct Boto3 code to perform a scan operation with a filter where score > 80 in Students table?

1. table.scan(FilterExpression=Key('score').gt(80))

2. table.get\_item(Filter='score > 80')

3. table.scan(FilterExpression=Attr('score').gt(80))

4. table.query(FilterExpression=Attr('score').gt(80))

Task 56:

In Boto3, how would you read an item from Books table with a key {"ISBN": "123-456-789"}?

1. table.read({"ISBN": "123-456-789"})

2. table.get\_item(Key={"ISBN": "123-456-789"})

3. table.fetch\_item(Key={"ISBN": "123-456-789"})

4. dynamodb.get(Key={"ISBN": "123-456-789"})