DynamoDB Cheatsheet

A quick reference guide to Amazon DynamoDB, covering key concepts, data types, operations, and best practices for efficient database management.

Core Concepts

Basic Definitions

DynamoDB: A fully managed, serverless, key-value and document database offered by Amazon Web Services (AWS).

Table: A collection of items, similar to a table in a relational database.

Item: A collection of attributes, which is analogous to a row in a relational database.

Attribute: A key-value pair that describes a property of an item.

Primary Key: A unique identifier for each item in a table, composed of either a partition key or a partition key and sort key.

Partition Key (Hash Key): Used to distribute data across partitions for scalability.

Sort Key (Range Key): Used to sort items within a partition.

Secondary Index: A data structure that allows you to query the table using attributes other than the primary key.

Data Types

Scalar Types	String, Number, Binary, Boolean, Null
Document Types	List, Map
Set Types	String Set, Number Set, Binary Set
Note	DynamoDB is schemaless, meaning each item in a table can have different attributes.

Provisioned vs. On-Demand Capacity

Provisioned Capacity	You specify the number of read and write capacity units (RCUs/WCUs) your application requires. Good for predictable workloads.
On-Demand Capacity	DynamoDB automatically scales capacity based on your application's traffic patterns. Good for unpredictable workloads.

Basic Operations

CRUD Operations

`PutItem`: Creates a new item or replaces an existing item.

Example (AWS CLI):

```
aws dynamodb put-item --table-name MyTable --item '{"id": {"N": "1"}, "name": {"S": "Example"}}'
```

`GetItem`: Retrieves an item by its primary key.

Example (AWS CLI):

```
aws dynamodb get-item --table-name MyTable --key '{"id": {"N": "1"}}'
```

`UpdateItem`: Modifies an existing item.

Example (AWS CLI):

```
aws dynamodb update-item --table-name MyTable --key '{"id": {"N": "1"}}' --update-expression 'SET #n = :val' --expression-attribute-names '{"#n": "name"}' --expression-attribute-values '{ ":val": {"S": "Updated Example"} }'
```

`DeleteItem`: Deletes an item by its primary key.

Example (AWS CLI):

```
aws dynamodb delete-item --table-name MyTable --key '{"id": {"N": "1"}}'
```

Query and Scan

`Query`	Retrieves items based on primary key attributes. Requires the partition key and optionally a condition on the sort key. More efficient than `Scan`.
`Scan`	Retrieves all items in a table (or a subset based on filter expressions). Less efficient than `Query`, especially for large tables, as it reads every item.
Example Query (AWS CLI)	`aws dynamodb query --table-name MyTable --key-condition-expression 'id = :id' --expression-attribute-values '{ ":id": {"N": "1"} }'`
Example Scan (AWS CLI)	`aws dynamodb scan --table-name MyTable`

Batch Operations

`BatchWriteItem`	Performs multiple `PutItem` and `DeleteItem` operations in a single request, improving efficiency for bulk data operations.
`BatchGetItem`	Retrieves multiple items from one or more tables in a single request, reducing the number of API calls.

Indexes

Global Secondary Index (GSI)

An index that allows queries on attributes other than the primary key. Can have a different partition and sort key than the base table.

Key characteristics:

- Can be created or deleted at any time.
- Queries can span all items in the table.
- Has its own provisioned throughput capacity.

Local Secondary Index (LSI)

An index that has the same partition key as the base table but a different sort key. Must be created when the table is created.

Key characteristics:

- Shares the provisioned throughput capacity of the base table.
- Limited to 5 LSIs per table.
- Offers strong consistency reads.

Choosing an Index

Use GSI when:	• You need to query on attributes other than the primary key. • Your query patterns are diverse and don't align with the base table's primary key. • You need to project only a subset of attributes to improve query performance and reduce costs.
Use LSI when:	• You need to query using an alternate sort key but the same partition key as the base table. • You require strongly consistent reads.

Best Practices

Data Modeling

Understand Access Patterns: Before designing your table, carefully analyze your application's read and write access patterns to optimize your schema for performance and cost.

Avoid Hot Partitions: Ensure even distribution of data across partitions by choosing appropriate partition keys. Avoid keys with low cardinality or that lead to uneven distribution of writes.

Denormalization: Consider denormalizing your data by embedding related data within a single item to reduce the need for multiple queries.

Performance Optimization

Use Projections: When querying indexes, project only the attributes you need to reduce the amount of data read and improve performance.

Batch Operations: Use `BatchGetItem` and `BatchWriteItem` to perform multiple read and write operations in a single request, reducing latency and improving throughput.

Parallel Scans: For large tables, use parallel scans to divide the scan operation into multiple segments, improving the overall scan time. (Use with caution as it can consume significant RCUs).

Security

IAM Roles: Use IAM roles to grant fine-grained permissions to your application to access DynamoDB tables, following the principle of least privilege.

Encryption: Enable encryption at rest and in transit to protect sensitive data. DynamoDB supports encryption using AWS KMS.