1. What is DynamoDB?

- **Amazon DynamoDB** is a fully managed **NoSQL** database service by AWS.
- It stores data as **key-value pairs** and **documents**, optimized for fast, predictable performance.
- Designed to **scale horizontally** and handle large volumes of data with low latency.
- Designed for single-digit millisecond latency at any scale.
- Automatically scales throughput capacity to meet traffic demands.
- Offers built-in security, backup, and restore capabilities.
- Provides event-driven programming via DynamoDB Streams and AWS Lambda.

2. NoSQL vs SQL Databases

Aspect	SQL (Relational DB)	NoSQL (DynamoDB)
Schema	Fixed schema, tables with rows and columns	Schema-less, flexible attributes
Data Model	Relational, normalized	Key-value and document-oriented
Query Language	SQL	DynamoDB API and PartiQL (SQL-like)
Scaling	Vertical scaling	Horizontal scaling (partitioned)
Transactions	ACID transactions	Supports transactions but different (optimized for speed)
Use Cases	Complex relational data	High-scale, flexible, real-time apps

3. Core Components of DynamoDB

Component	Description
Table	Collection of data (like a SQL table but schema-less)
Item	A single record in a table (like a row)
Attribute	A key-value pair within an item (like a column)
Primary Key	Uniquely identifies an item in the table

4. Primary Keys: The Heart of DynamoDB

Primary keys uniquely identify each item in a table. They determine data distribution across partitions.

Two types:

- **Partition Key (Hash Key):**
    - Single attribute used to partition data.
    - DynamoDB uses a hash function on this key to determine the storage partition.
    - Must be unique for each item if only partition key is used.
- **Composite Key (Partition + Sort Key):**
    - Partition Key + Sort Key (range key).
    - Partition Key distributes data.

5. Data Types Supported

Data Type	Description
Scalar Types	String, Number, Binary, Boolean, Null
Document Types	List, Map
Set Types	String Set, Number Set, Binary Set

6. Capacity Modes

- **Provisioned Mode:**
  - You specify Read Capacity Units (RCU) and Write Capacity Units (WCU).
  - Good for predictable workloads.
- **On-Demand Mode:**
  - Pay-per-request pricing, scales automatically.
  - Great for unpredictable or spiky workloads.

7. Read & Write Capacity Units (RCU & WCU)

Capacity Unit	Operation	Amount of data
1 RCU	Strongly consistent read	Up to 4 KB per second
1 RCU	Eventually consistent read	Up to 8 KB per second
1 WCU	Write operation	Up to 1 KB per second

If you read/write items larger than these sizes, DynamoDB consumes more capacity units.

8. Query vs Scan

Operation	Description	Use Case	Performance
Query	Retrieve items by Primary Key or index	Efficient for known key lookups	Fast and cost-effective
Scan	Reads entire table or index	Retrieve large datasets, no key	Slow and expensive, avoid if possible

Summary for Beginner Level Concepts:

- DynamoDB is a **NoSQL, key-value and document database**.
- Uses **Tables** with **Items** and **Attributes**.
- Requires careful design of **Primary Keys** for performance.
- Supports flexible **data types** and capacity modes.
- Query efficiently using **Query** operation, avoid expensive **Scan**.
- Capacity units govern cost and throughput.

Next Steps:
Do you want me to continue explaining the next beginner topics like **Basic Operations (CRUD)** and **Security Basics**? Or would you prefer some questions/quizzes to check understanding so far?

- Document (tables)
- Items (rows)
- Attributes(column field)

List tables

```
aws dynamodb list-tables
```

`Point-in-time-backups for Dynamodb`

It's considered best practice to enable Point-in-time backups for DynamoDB on the table by running the following command

🧭 AWS CLI DynamoDB — Complete Command Reference

> **Base Command:**
>
> ```
> aws dynamodb <subcommand> [options]
> ```
>
> Requires AWS CLI configured (`aws configure`) with valid credentials and region.

📘 Table Management

1. `create-table`

Purpose: Create a new DynamoDB table.

Example:

```
aws dynamodb create-table \
  --table-name Users \
  --attribute-definitions AttributeName=UserID,AttributeType=S \
  --key-schema AttributeName=UserID,KeyType=HASH \
  --billing-mode PAY_PER_REQUEST
```

Tips:

- Use `--billing-mode PAY_PER_REQUEST` for unpredictable workloads.
- `--provisioned-throughput` needed if using `PROVISIONED` mode.

2. `describe-table`

Purpose: Show metadata and status of a table.

```
aws dynamodb describe-table --table-name Users
```

Tips: Use this after creation to check status (`CREATING`, `ACTIVE`).

3. `list-tables`

Purpose: List all tables in the current region.

```
aws dynamodb list-tables
```

Tips: Combine with `--max-items` and `--starting-token` for pagination.

4. `update-table`

Purpose: Modify table capacity, indexes, or stream settings.

```
aws dynamodb update-table \
  --table-name Users \
  --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5
```

5. `delete-table`

Purpose: Delete a table and all its data.

```
aws dynamodb delete-table --table-name Users
```

⚠️ **Tip:** Back up data first — deletes are irreversible.

🔄 CRUD Operations

6. `put-item`

Purpose: Insert or replace an item.

```
aws dynamodb put-item \
  --table-name Users \
  --item '{"UserID":{"S":"123"}, "Name":{"S":"Alice"}}'
```

💡 **Tip:** To prevent overwrite, use `--condition-expression`:

```
--condition-expression "attribute_not_exists(UserID)"
```

7. `get-item`

Purpose: Retrieve a single item by key.

```
aws dynamodb get-item \
  --table-name Users \
  --key '{"UserID":{"S":"123"}}'
```

8. `update-item`

Purpose: Update specific attributes.

```
aws dynamodb update-item \
  --table-name Users \
  --key '{"UserID":{"S":"123"}}' \
  --update-expression "SET Age = :a" \
  --expression-attribute-values '{":a":{"N":"30"}}'
```

9. `delete-item`

Purpose: Remove an item.

```
aws dynamodb delete-item \
  --table-name Users \
  --key '{"UserID":{"S":"123"}}'
```

🔎 Querying & Scanning

10. `query`

Purpose: Retrieve items by primary key or index.

```
aws dynamodb query \
  --table-name Users \
  --key-condition-expression "UserID = :u" \
  --expression-attribute-values '{":u":{"S":"123"}}'
```

💡 **Tips:**

- Queries use indexed attributes — faster than `scan`.
- Use `--index-name` for secondary indexes.

11. `scan`

Purpose: Read all items in a table.

```
aws dynamodb scan --table-name Users
```

⚠️ **Tip:** Expensive! Use pagination (`--max-items`, `--starting-token`).

⚙ Batch Operations

12. `batch-get-item`

Purpose: Get multiple items across tables.

```
aws dynamodb batch-get-item \
  --request-items file://batch-get.json
```

batch-get.json Example:

```
{
  "Users": {
    "Keys": [{ "UserID": { "S": "123" } }, { "UserID": { "S": "456" } }]
  }
}
```

13. `batch-write-item`

Purpose: Insert or delete multiple items.

```
aws dynamodb batch-write-item \
  --request-items file://batch-write.json
```

batch-write.json Example:

```
{
  "Users": [
    {
      "PutRequest": {
        "Item": { "UserID": { "S": "789" }, "Name": { "S": "Bob" } }
      }
    },
    { "DeleteRequest": { "Key": { "UserID": { "S": "123" } } } }
  ]
}
```

💡 **Tip:** Each batch max 25 items; handle unprocessed items in response

💾 Backup & Restore

14. `create-backup`

Purpose: Create on-demand backup.

```
aws dynamodb create-backup --table-name Users --backup-name UsersBackup1
```

15. `list-backups`

Purpose: List table backups.

```
aws dynamodb list-backups --table-name Users
```

16. `restore-table-from-backup`

Purpose: Restore from a backup.

```
aws dynamodb restore-table-from-backup \
  --target-table-name UsersRestored \
  --backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Users/backup/0155…
```

17. `export-table-to-point-in-time`

Purpose: Export data to S3.

```
aws dynamodb export-table-to-point-in-time \
  --table-arn arn:aws:dynamodb:us-east-1:123456789012:table/Users \
  --s3-bucket my-dynamodb-exports
```

🔄 Transactions

18. `transact-get-items`

Retrieve multiple items atomically.

```
aws dynamodb transact-get-items --transact-items file://transact-get.json
```

19. `transact-write-items`

Write multiple items atomically.

```
aws dynamodb transact-write-items --transact-items file://transact-write.json
```

💡 **Tip:** Use for multi-table atomic operations; 25-item limit.

🧠 SQL-like Access

20. `execute-statement`

Run PartiQL (SQL-like) commands.

```
aws dynamodb execute-statement --statement "SELECT * FROM Users WHERE UserID='123'"
```

21. `execute-transaction`

Run multiple PartiQL statements atomically.

```
aws dynamodb execute-transaction --transact-statements file://transaction.json
```

🌍 Global Tables & Replication

22. `create-global-table`

Create a global table with replicas in multiple regions.

23. `update-global-table`

Add new regions.

24. `describe-global-table`

View details.

💡 **Tip:** All replica tables must have identical schemas and indexes.

🔔 Streams, TTL, Insights

25. `update-time-to-live`

Enable automatic expiry.

```
aws dynamodb update-time-to-live \
  --table-name Users \
  --time-to-live-specification "Enabled=true, AttributeName=expireAt"
```

26. `describe-time-to-live`

Check TTL status.

27. `update-contributor-insights` / `describe-contributor-insights`

Monitor read/write hot spots.

🏷️ Tagging

28. `tag-resource`

Add tags to a resource.

```
aws dynamodb tag-resource \
  --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Users \
  --tags Key=Environment,Value=Production
```

29. `untag-resource`

Remove tags.

30. `list-tags-of-resource`

List all tags.

⏳ Utilities & Misc

31. `wait`

Wait until resource reaches a state (e.g., table exists).

```
aws dynamodb wait table-exists --table-name Users
```

32. `help`

Show help for subcommands.

33. `wizard`

Interactive CLI table creation helper (rarely used).

⚡ Pro Tips

✅ Use `--return-consumed-capacity` and `--return-values` for debugging writes.
✅ Use `--projection-expression` to limit retrieved attributes.
✅ Combine with `jq` to parse output:

```
aws dynamodb list-tables | jq -r '.TableNames[]'
```

✅ Automate batch jobs with **AWS CLI pagination tokens**.
✅ For repeatable operations, store parameters in JSON files (`file://params.json`).

Would you like me to generate this as a **formatted PDF “DynamoDB CLI Master Guide”** (with sections, examples, and best practices)?
It’ll include **every command** and **developer tips** in an easy-to-read format.

SDK (@aws-sdk/client-dynamodb)

Absolutely! Let’s do a **complete expert-level guide** for DynamoDB’s main commands in **AWS SDK v3 (`@aws-sdk/lib-dynamodb`)**, including required and optional parameters, examples, and best practices. I’ll break it down command by command.

AWS DynamoDB Commands – Expert Guide

We’ll assume `docClient` is initialized as:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
export const docClient = DynamoDBDocumentClient.from(client);
```

1️⃣ GetCommand – Retrieve a single item by primary key

Purpose: Get one item by **PK** or **PK+SK**.

Required Parameters:

- `TableName` – string
- `Key` – object with exact **primary key** (`{ PK, SK? }`)

Optional Parameters:

- `ProjectionExpression` – string, specify which attributes to return
- `ConsistentRead` – boolean, default `false` (strong consistency if `true`)
- `ExpressionAttributeNames` – mapping for reserved keywords

Example:

```
import { GetCommand } from "@aws-sdk/lib-dynamodb";
```

2️⃣ PutCommand – Create or replace an item

Purpose: Insert or overwrite an item.

Required Parameters:

- `TableName` – string
- `Item` – object containing all attributes for the item

Optional Parameters:

- `ConditionExpression` – only insert if condition matches (avoid overwrites)
- `ExpressionAttributeValues` – values for condition expression
- `ReturnValues` – what to return after operation (`NONE`, `ALL_OLD`)

Example:

```
import { PutCommand } from "@aws-sdk/lib-dynamodb";
```

3️⃣ UpdateCommand – Update attributes of an item

Purpose: Modify attributes of an existing item without overwriting the whole item.

Required Parameters:

- `TableName`
- `Key` – primary key object
- `UpdateExpression` – string, defines how to modify attributes
- `ExpressionAttributeValues` – values for update expression

Optional Parameters:

- `ConditionExpression` – only update if condition matches
- `ExpressionAttributeNames` – for reserved keywords
- `ReturnValues` – `"NONE" | "UPDATED_OLD" | "ALL_OLD" | "UPDATED_NEW" | "ALL_NEW"`

4️⃣ DeleteCommand – Remove an item

Purpose: Delete a single item by primary key.

Required Parameters:

- `TableName`
- `Key` – primary key object

Optional Parameters:

- `ConditionExpression` – delete only if condition matches
- `ReturnValues` – `"NONE" | "ALL_OLD"`

Example:

```
import { DeleteCommand } from "@aws-sdk/lib-dynamodb";

async function deleteUser(userId) {
```

5️⃣ QueryCommand – Retrieve multiple items by partition key (efficient)

Purpose: Fetch items by **PK** (optionally SK range).

Required Parameters:

- `TableName`
- `KeyConditionExpression` – string like `"PK = :pk"`
- `ExpressionAttributeValues` – mapping for placeholders in KeyConditionExpression

Optional Parameters:

- `FilterExpression` – filter items after fetching
- `ProjectionExpression` – select specific attributes
- `ScanIndexForward` – true=ascending, false=descending
- `Limit` – max number of items per page

6️⃣ ScanCommand – Read all items in table (less efficient)

Purpose: Scan the whole table and optionally filter.

Required Parameters:

- `TableName`

Optional Parameters:

- `FilterExpression` – filter items after scan
- `ProjectionExpression` – select specific attributes
- `Limit` – max items per request
- `ExclusiveStartKey` – for pagination

Example: Fetch all users (with pagination)

```
import { ScanCommand } from "@aws-sdk/lib-dynamodb";
```

✅ Key Notes / Best Practices

1. **Get vs Query vs Scan**
   - `Get` – single item by full key
   - `Query` – multiple items by PK (fast)
   - `Scan` – all items (slow, costly)
2. **UpdateCommand** can be used to **increment counters** or modify parts of item without overwriting it.
3. **PutCommand** supports `ConditionExpression` to avoid overwrites (like upserts).
4. **Pagination**
   - Query and Scan support `Limit` + `ExclusiveStartKey` for paginated results.
5. **Reserved keywords**

If you want, I can make a **fully working Express.js CRUD example** using all 6 commands **with PK/SK pattern** like `USER#id` / `PROFILE#id`, so you can copy-paste into production.

Do you want me to do that?