What is dynamodb?

Dynamodb is a fully managed NoSQL database provided by AWS.

- 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 dynamodb update-continuous-backups --table-name Music --point-in-time-
recovery-specification  PointInTimeRecoveryEnabled=true
```

There are cost implications to enabling continuous backups with point-in-time recovery.

`Dynamodb command:`

Excellent — you're looking for a **complete DynamoDB CLI command guide** (the commands you listed are from the AWS CLI's `aws dynamodb` command group).

This is a *big* list (50+ commands), so instead of dumping a giant unreadable blob, let's organize this into a **clear, structured reference** — a *comprehensive DynamoDB CLI guide* with:

1. **Categorized commands**
2. **Purpose & syntax summary**
3. **Example usage**
4. **Tips, tricks, and gotchas**

🧭 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";

async function getUser(userId) {
  const command = new GetCommand({
    TableName: "Users",
    Key: { PK: `USER#${userId}`, SK: `PROFILE#${userId}` },
    ProjectionExpression: "PK, SK, name, email", // optional
    ConsistentRead: true,
  });

  const result = await docClient.send(command);
  return result.Item; // single object or undefined
}
```

---

2 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";

async function createUser(user) {
  const command = new PutCommand({
    TableName: "Users",
    Item: {
      PK: `USER#${user.id}`,
      SK: `PROFILE#${user.id}`,
      name: user.name,
      email: user.email,
      mobile: user.mobile,
    },
    ConditionExpression: "attribute_not_exists(PK)", // prevent overwrite
    ReturnValues: "ALL_OLD",
  });

  return await docClient.send(command);
}
```

3 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"`

Example:

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

async function updateUserEmail(userId, newEmail) {
  const command = new UpdateCommand({
    TableName: "Users",
    Key: { PK: `USER#${userId}`, SK: `PROFILE#${userId}` },
    UpdateExpression: "SET email = :email",
    ExpressionAttributeValues: { ":email": newEmail },
    ReturnValues: "ALL_NEW",
  });

  return await docClient.send(command);
}
```

4 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) {
  const command = new DeleteCommand({
    TableName: "Users",
    Key: { PK: `USER#${userId}`, SK: `PROFILE#${userId}` },
    ReturnValues: "ALL_OLD",
  });

  return await docClient.send(command);
}
```

---

5 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
- `ExclusiveStartKey` – for pagination

Example: Fetch all items for a user PK

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

async function getUserItems(userId) {
  const command = new QueryCommand({
    TableName: "Users",
    KeyConditionExpression: "PK = :pk",
    ExpressionAttributeValues: { ":pk": `USER#${userId}` },
    ScanIndexForward: true,
  });

  const result = await docClient.send(command);
  return result.Items; // array of items
}
```

Example: Fetch only profile item by SK prefix

```
const command = new QueryCommand({
  TableName: "Users",
  KeyConditionExpression: "PK = :pk AND begins_with(SK, :skPrefix)",
  ExpressionAttributeValues: {
    ":pk": `USER#${userId}`,
    ":skPrefix": "PROFILE#",
  },
});
```

---

6 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";

async function getAllUsers() {
  let items = [];
  let ExclusiveStartKey = undefined;

  do {
    const command = new ScanCommand({
      TableName: "Users",
      Limit: 50,
      ExclusiveStartKey,
    });

    const result = await docClient.send(command);
    items = items.concat(result.Items);
    ExclusiveStartKey = result.LastEvaluatedKey;
  } while (ExclusiveStartKey);

  return items;
}
```

✅ 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**
   - Use `ExpressionAttributeNames` if attribute names conflict with DynamoDB reserved words.
6. **ReturnValues**
   - Always choose carefully:
     - `"ALL_OLD"` – returns item before update/delete
     - `"ALL_NEW"` – returns item after update
     - `"NONE"` – saves bandwidth

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?