Amazon DynamoDB

API Reference

API Version 2012-08-10

Amazon DynamoDB: API Reference

Table of Contents

Welcome 1
Actions 3
BatchGetItem 4
Request Syntax 4
Request Parameters 5
Response Syntax 7
Response Elements 9
Errors 10
Examples 10
BatchWriteItem 13
Request Syntax 14
Request Parameters 15
Response Syntax 16
Response Elements 18
Errors 19
Examples 20
CreateTable 23
Request Syntax 23
Request Parameters 24
Response Syntax 26
Response Elements 28
Errors 28
Examples 28
DeleteItem 32
Request Syntax 32
Request Parameters 34
Response Syntax 40
Response Elements 41
Errors 42
Examples 43
DeleteTable 45
Request Syntax 45
Request Parameters 45
Response Syntax 45
Response Elements 47
Errors 47
Examples 48
DescribeTable 49
Request Syntax 49
Request Parameters 49
Response Syntax 49
Response Elements 51
Errors 51
Examples 51
GetItem 54
Request Syntax 54
Request Parameters 55
Response Syntax 57
Response Elements 58
Errors 58
Examples 58
ListTables 60
Request Syntax 60
Request Parameters 60
Response Syntax 60

Response Elements 60
Errors 61
Examples 61
PutItem 63
Request Syntax 63
Request Parameters 65
Response Syntax 71
Response Elements 73
Errors 73
Examples 74
Query 76
Request Syntax 76
Request Parameters 79
Response Syntax 87
Response Elements 88
Errors 89
Examples 89
Scan 92
Request Syntax 92
Request Parameters 94
Response Syntax 99
Response Elements 101
Errors 102
Examples 102
UpdateItem 106
Request Syntax 106
Request Parameters 109
Response Syntax 117
Response Elements 118
Errors 119
Examples 119
UpdateTable 122
Request Syntax 122
Request Parameters 123
Response Syntax 124
Response Elements 125
Errors 126
Examples 126
Data Types 129
AttributeDefinition 130
Description 130
Contents 130
AttributeValue 130
Description 130
Contents 130
AttributeValueUpdate 132
Description 132
Contents 132
Capacity 133
Description 133
Contents 133
Condition 134
Description 134
Contents 134
ConsumedCapacity 136
Description 136
Contents 136
CreateGlobalSecondaryIndexAction 137

Description ... 137
Contents ... 137
DeleteGlobalSecondaryIndexAction ... 138
Description ... 138
Contents ... 138
DeleteRequest ... 138
Description ... 138
Contents ... 139
ExpectedAttributeValue ... 139
Description ... 139
Contents ... 139
GlobalSecondaryIndex ... 142
Description ... 142
Contents ... 142
GlobalSecondaryIndexDescription ... 143
Description ... 143
Contents ... 143
GlobalSecondaryIndexUpdate ... 145
Description ... 145
Contents ... 145
ItemCollectionMetrics ... 145
Description ... 145
Contents ... 146
KeysAndAttributes ... 146
Description ... 146
Contents ... 146
KeySchemaElement ... 148
Description ... 148
Contents ... 148
LocalSecondaryIndex ... 148
Description ... 148
Contents ... 148
LocalSecondaryIndexDescription ... 149
Description ... 149
Contents ... 149
Projection ... 150
Description ... 150
Contents ... 150
ProvisionedThroughput ... 151
Description ... 151
Contents ... 151
ProvisionedThroughputDescription ... 151
Description ... 151
Contents ... 151
PutRequest ... 152
Description ... 152
Contents ... 152
StreamSpecification ... 153
Description ... 153
Contents ... 153
TableDescription ... 153
Description ... 153
Contents ... 153
UpdateGlobalSecondaryIndexAction ... 157
Description ... 157
Contents ... 157
WriteRequest ... 157
Description ... 157

Contents 158
Common Errors 159
..... 159

Welcome

This is the Amazon DynamoDB API Reference. This guide provides descriptions of the low-level DynamoDB API.

This guide is intended for use with the following DynamoDB documentation:

- Amazon DynamoDB Getting Started Guide - provides hands-on exercises that help you learn the basics of working with DynamoDB. *If you are new to DynamoDB, we recommend that you begin with the Getting Started Guide.*
- Amazon DynamoDB Developer Guide - contains detailed information about DynamoDB concepts, usage, and best practices.
- Amazon DynamoDB Streams API Reference - provides descriptions and samples of the DynamoDB Streams API. (For more information, see Capturing Table Activity with DynamoDB Streams in the Amazon DynamoDB Developer Guide.)

Instead of making the requests to the low-level DynamoDB API directly from your application, we recommend that you use the AWS Software Development Kits (SDKs). The easy-to-use libraries in the AWS SDKs make it unnecessary to call the low-level DynamoDB API directly from your application. The libraries take care of request authentication, serialization, and connection management. For more information, see Using the AWS SDKs with DynamoDB in the Amazon DynamoDB Developer Guide.

If you decide to code against the low-level DynamoDB API directly, you will need to write the necessary code to authenticate your requests. For more information on signing your requests, see Using the DynamoDB API in the *Amazon DynamoDB Developer Guide*.

The following are short descriptions of each low-level API action, organized by function.

Managing Tables

- *CreateTable* - Creates a table with user-specified provisioned throughput settings. You must designate one attribute as the hash primary key for the table; you can optionally designate a second attribute as the range primary key. DynamoDB creates indexes on these key attributes for fast data access. Optionally, you can create one or more secondary indexes, which provide fast data access using non-key attributes.
- *DescribeTable* - Returns metadata for a table, such as table size, status, and index information.
- *UpdateTable* - Modifies the provisioned throughput settings for a table. Optionally, you can modify the provisioned throughput settings for global secondary indexes on the table.
- *ListTables* - Returns a list of all tables associated with the current AWS account and endpoint.

- *DeleteTable* - Deletes a table and all of its indexes.

For conceptual information about managing tables, see Working with Tables in the *Amazon DynamoDB Developer Guide*.

Reading Data

- *GetItem* - Returns a set of attributes for the item that has a given primary key. By default, *GetItem* performs an eventually consistent read; however, applications can request a strongly consistent read instead.
- *BatchGetItem* - Performs multiple *GetItem* requests for data items using their primary keys, from one table or multiple tables. The response from *BatchGetItem* has a size limit of 16 MB and returns a maximum of 100 items. Both eventually consistent and strongly consistent reads can be used.
- *Query* - Returns one or more items from a table or a secondary index. You must provide a specific hash key value. You can narrow the scope of the query using comparison operators against a range key value, or on the index key. *Query* supports either eventual or strong consistency. A single response has a size limit of 1 MB.
- *Scan* - Reads every item in a table; the result set is eventually consistent. You can limit the number of items returned by filtering the data attributes, using conditional expressions. *Scan* can be used to enable ad-hoc querying of a table against non-key attributes; however, since this is a full table scan without using an index, *Scan* should not be used for any application query use case that requires predictable performance.

For conceptual information about reading data, see Working with Items and Query and Scan Operations in the *Amazon DynamoDB Developer Guide*.

Modifying Data

- *PutItem* - Creates a new item, or replaces an existing item with a new item (including all the attributes). By default, if an item in the table already exists with the same primary key, the new item completely replaces the existing item. You can use conditional operators to replace an item only if its attribute values match certain conditions, or to insert a new item only if that item doesn't already exist.
- *UpdateItem* - Modifies the attributes of an existing item. You can also use conditional operators to perform an update only if the item's attribute values match certain conditions.
- *DeleteItem* - Deletes an item in a table by primary key. You can use conditional operators to perform a delete an item only if the item's attribute values match certain conditions.
- *BatchWriteItem* - Performs multiple *PutItem* and *DeleteItem* requests across multiple tables in a single request. A failure of any request(s) in the batch will not cause the entire *BatchWriteItem* operation to fail. Supports batches of up to 25 items to put or delete, with a maximum total request size of 16 MB.

For conceptual information about modifying data, see Working with Items and Query and Scan Operations in the *Amazon DynamoDB Developer Guide*.

This document was last updated on September 3, 2015.

Actions

The following actions are supported:

- BatchGetItem (p. 4)
- BatchWriteItem (p. 13)
- CreateTable (p. 23)
- DeleteItem (p. 32)
- DeleteTable (p. 45)
- DescribeTable (p. 49)
- GetItem (p. 54)
- ListTables (p. 60)
- PutItem (p. 63)
- Query (p. 76)
- Scan (p. 92)
- UpdateItem (p. 106)
- UpdateTable (p. 122)

BatchGetItem

The *BatchGetItem* operation returns the attributes of one or more items from one or more tables. You identify requested items by primary key.

A single operation can retrieve up to 16 MB of data, which can contain as many as 100 items. *BatchGetItem* will return a partial result if the response size limit is exceeded, the table's provisioned throughput is exceeded, or an internal processing failure occurs. If a partial result is returned, the operation returns a value for *UnprocessedKeys*. You can use this value to retry the operation starting with the next item to get.

> **Important**
> If you request more than 100 items BatchGetItem will return a ValidationException with the message "Too many items requested for the BatchGetItem call".

For example, if you ask to retrieve 100 items, but each individual item is 300 KB in size, the system returns 52 items (so as not to exceed the 16 MB limit). It also returns an appropriate *UnprocessedKeys* value so you can get the next page of results. If desired, your application can include its own logic to assemble the pages of results into one data set.

If *none* of the items can be processed due to insufficient provisioned throughput on all of the tables in the request, then *BatchGetItem* will return a *ProvisionedThroughputExceededException*. If *at least one* of the items is successfully processed, then *BatchGetItem* completes successfully, while returning the keys of the unread items in *UnprocessedKeys*.

> **Important**
> If DynamoDB returns any unprocessed items, you should retry the batch operation on those items. However, we strongly recommend that you use an exponential backoff algorithm. If you retry the batch operation immediately, the underlying read or write requests can still fail due to throttling on the individual tables. If you delay the batch operation using exponential backoff, the individual requests in the batch are much more likely to succeed. For more information, see Batch Operations and Error Handling in the Amazon DynamoDB Developer Guide.

By default, *BatchGetItem* performs eventually consistent reads on every table in the request. If you want strongly consistent reads instead, you can set *ConsistentRead* to `true` for any or all tables.

In order to minimize response latency, *BatchGetItem* retrieves items in parallel.

When designing your application, keep in mind that DynamoDB does not return attributes in any particular order. To help parse the response by item, include the primary key values for the items in your request in the *AttributesToGet* parameter.

If a requested item does not exist, it is not returned in the result. Requests for nonexistent items consume the minimum read capacity units according to the type of read. For more information, see Capacity Units Calculations in the *Amazon DynamoDB Developer Guide*.

Request Syntax

```
{
    "RequestItems":
        {
            "string" :
                {
                    "AttributesToGet": [
                        "string"
                    ],
```

```
                    "ConsistentRead": boolean,
                    "ExpressionAttributeNames":
                        {
                            "string" :
                                "string"
                        },
                    "Keys": [

                            {
                                "string" :
                                    {
                                        "B": blob,
                                        "BOOL": boolean,
                                        "BS": [
                                            blob
                                        ],
                                        "L": [
                                            AttributeValue
                                        ],
                                        "M":
                                            {
                                                "string" :
                                                    AttributeValue
                                            },
                                        "N": "string",
                                        "NS": [
                                            "string"
                                        ],
                                        "NULL": boolean,
                                        "S": "string",
                                        "SS": [
                                            "string"
                                        ]
                                    }
                            }
                    ],
                    "ProjectionExpression": "string"
                }
        },
    "ReturnConsumedCapacity": "string"
}
```

Request Parameters

The request requires the following data in JSON format.

Note
In the following list, the required parameters are described first.

RequestItems

A map of one or more table names and, for each table, a map that describes one or more items to retrieve from that table. Each table name can be used only once per *BatchGetItem* request.

Each element in the map of items to retrieve consists of the following:

- *ConsistentRead* - If `true`, a strongly consistent read is used; if `false` (the default), an eventually consistent read is used.

- *ExpressionAttributeNames* - One or more substitution tokens for attribute names in the *ProjectionExpression* parameter. The following are some use cases for using *ExpressionAttributeNames*:
  - To access an attribute whose name conflicts with a DynamoDB reserved word.
  - To create a placeholder for repeating occurrences of an attribute name in an expression.
  - To prevent special characters in an attribute name from being misinterpreted in an expression.

  Use the # character in an expression to dereference an attribute name. For example, consider the following attribute name:
  - `Percentile`

  The name of this attribute conflicts with a reserved word, so it cannot be used directly in an expression. (For the complete list of reserved words, see Reserved Words in the *Amazon DynamoDB Developer Guide*). To work around this, you could specify the following for *ExpressionAttributeNames*:
  - `{"#P":"Percentile"}`

  You could then use this substitution in an expression, as in this example:
  - `#P = :val`

    Note
    Tokens that begin with the : character are expression attribute values, which are placeholders for the actual value at runtime.

  For more information on expression attribute names, see Accessing Item Attributes in the *Amazon DynamoDB Developer Guide*.
- *Keys* - An array of primary key attribute values that define specific items in the table. For each primary key, you must provide *all* of the key attributes. For example, with a hash type primary key, you only need to provide the hash attribute. For a hash-and-range type primary key, you must provide *both* the hash attribute and the range attribute.
- *ProjectionExpression* - A string that identifies one or more attributes to retrieve from the table. These attributes can include scalars, sets, or elements of a JSON document. The attributes in the expression must be separated by commas.

  If no attribute names are specified, then all attributes will be returned. If any of the requested attributes are not found, they will not appear in the result.

  For more information, see Accessing Item Attributes in the *Amazon DynamoDB Developer Guide*.
- *AttributesToGet* -

  Important
  This is a legacy parameter, for backward compatibility. New applications should use ProjectionExpression instead. Do not combine legacy parameters and expression parameters in a single API call; otherwise, DynamoDB will return a ValidationException exception. This parameter allows you to retrieve attributes of type List or Map; however, it cannot retrieve individual elements within a List or a Map.

  The names of one or more attributes to retrieve. If no attribute names are provided, then all attributes will be returned. If any of the requested attributes are not found, they will not appear in the result.

  Note that *AttributesToGet* has no effect on provisioned throughput consumption. DynamoDB determines capacity units consumed based on item size, not on the amount of data that is returned to an application.

Type: String to KeysAndAttributes (p. 146) object map

Length constraints: Minimum length of 1. Maximum length of 100.

Required: Yes

ReturnConsumedCapacity

Determines the level of detail about provisioned throughput consumption that is returned in the response:

- *INDEXES* - The response includes the aggregate *ConsumedCapacity* for the operation, together with *ConsumedCapacity* for each table and secondary index that was accessed.

  Note that some operations, such as *GetItem* and *BatchGetItem*, do not access any indexes at all. In these cases, specifying *INDEXES* will only return *ConsumedCapacity* information for table(s).
- *TOTAL* - The response includes only the aggregate *ConsumedCapacity* for the operation.
- *NONE* - No *ConsumedCapacity* details are included in the response.

Type: String

Valid Values: `INDEXES | TOTAL | NONE`

Required: No

Response Syntax

```
{
    "ConsumedCapacity": [
        {
            "CapacityUnits": number,
            "GlobalSecondaryIndexes":
                {
                    "string" :
                        {
                            "CapacityUnits": number
                        }
                },
            "LocalSecondaryIndexes":
                {
                    "string" :
                        {
                            "CapacityUnits": number
                        }
                },
            "Table": {
                "CapacityUnits": number
            },
            "TableName": "string"
        }
    ],
    "Responses":
        {
            "string" :
                [

                        {
                            "string" :
                                {
                                    "B": blob,
                                    "BOOL": boolean,
```

```
                                    "BS": [
                                        blob
                                    ],
                                    "L": [
                                        AttributeValue
                                    ],
                                    "M":
                                        {
                                            "string" :
                                                AttributeValue
                                        },
                                    "N": "string",
                                    "NS": [
                                        "string"
                                    ],
                                    "NULL": boolean,
                                    "S": "string",
                                    "SS": [
                                        "string"
                                    ]
                                }
                        }
                ]
        },
    "UnprocessedKeys":
        {
            "string" :
                {
                    "AttributesToGet": [
                        "string"
                    ],
                    "ConsistentRead": boolean,
                    "ExpressionAttributeNames":
                        {
                            "string" :
                                "string"
                        },
                    "Keys": [

                            {
                                "string" :
                                    {
                                        "B": blob,
                                        "BOOL": boolean,
                                        "BS": [
                                            blob
                                        ],
                                        "L": [
                                            AttributeValue
                                        ],
                                        "M":
                                            {
                                                "string" :
                                                    AttributeValue
                                            },
                                        "N": "string",
                                        "NS": [
                                            "string"
```

```
                                    ],
                                    "NULL": boolean,
                                    "S": "string",
                                    "SS": [
                                        "string"
                                    ]
                                }
                            }
                        ],
                        "ProjectionExpression": "string"
                    }
            }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ConsumedCapacity
The read capacity units consumed by the operation.

Each element consists of:

- *TableName* - The table that consumed the provisioned throughput.
- *CapacityUnits* - The total number of capacity units consumed.

Type: array of ConsumedCapacity (p. 136) objects

Responses
A map of table name to a list of items. Each object in *Responses* consists of a table name, along with a map of attribute data consisting of the data type and attribute value.

Type: String to map

UnprocessedKeys
A map of tables and their respective keys that were not processed with the current response. The *UnprocessedKeys* value is in the same form as *RequestItems*, so the value can be provided directly to a subsequent *BatchGetItem* operation. For more information, see *RequestItems* in the Request Parameters section.

Each element consists of:

- *Keys* - An array of primary key attribute values that define specific items in the table.
- *AttributesToGet* - One or more attributes to be retrieved from the table or index. By default, all attributes are returned. If a requested attribute is not found, it does not appear in the result.
- *ConsistentRead* - The consistency of a read operation. If set to `true`, then a strongly consistent read is used; otherwise, an eventually consistent read is used.

If there are no unprocessed keys remaining, the response contains an empty *UnprocessedKeys* map.

Type: String to KeysAndAttributes (p. 146) object map

Length constraints: Minimum length of 1. Maximum length of 100.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 159).

InternalServerError
An error occurred on the server side.

HTTP Status Code: 500

ProvisionedThroughputExceededException
Your request rate is too high. The AWS SDKs for DynamoDB automatically retry requests that receive this exception. Your request is eventually successful, unless your retry queue is too large to finish. Reduce the frequency of requests and use exponential backoff. For more information, go to Error Retries and Exponential Backoff in the *Amazon DynamoDB Developer Guide*.

HTTP Status Code: 400

ResourceNotFoundException
The operation tried to access a nonexistent table or index. The resource might not be specified correctly, or its status might not be `ACTIVE`.

HTTP Status Code: 400

Examples

Retrieve Items From Multiple Tables

The following sample requests attributes from two different tables.

Sample Request

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date> X-Amz-Target: DynamoDB_20120810.BatchGetItem

{
    "RequestItems": {
        "Forum": {
            "Keys": [
                {
                    "Name":{"S":"Amazon DynamoDB"}
                },
                {
                    "Name":{"S":"Amazon RDS"}
                },
                {
                    "Name":{"S":"Amazon Redshift"}
                }
            ],
```

```
            "ProjectionExpression":"Name, Threads, Messages, Views"
        },
        "Thread": {
            "Keys": [
                {
                    "ForumName":{"S":"Amazon DynamoDB"},
                    "Subject":{"S":"Concurrent reads"}
                }
            ],
            "ProjectionExpression":"Tags, Message"
        }
    },
    "ReturnConsumedCapacity": "TOTAL"
}
```

Sample Response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
 {
    "Responses": {
        "Forum": [
            {
                "Name":{
                    "S":"Amazon DynamoDB"
                },
                "Threads":{
                    "N":"5"
                },
                "Messages":{
                    "N":"19"
                },
                "Views":{
                    "N":"35"
                }
            },
            {
                "Name":{
                    "S":"Amazon RDS"
                },
                "Threads":{
                    "N":"8"
                },
                "Messages":{
                    "N":"32"
                },
                "Views":{
                    "N":"38"
                }
            },
```

```
            {
                "Name":{
                    "S":"Amazon Redshift"
                },
                "Threads":{
                    "N":"12"
                },
                "Messages":{
                    "N":"55"
                },
                "Views":{
                    "N":"47"
                }
            }
        ]
        "Thread": [
            {
                "Tags":{
                    "SS":["Reads","MultipleUsers"]
                },
                "Message":{
                    "S":"How many users can read a single data item at a time?
 Are there any limits?"
                }
            }
        ]
    },
    "UnprocessedKeys": {
    },
    "ConsumedCapacity": [
        {
            "TableName": "Forum",
            "CapacityUnits": 3
        },
        {
            "TableName": "Thread",
            "CapacityUnits": 1
        }
    ]
}
```

BatchWriteItem

The *BatchWriteItem* operation puts or deletes multiple items in one or more tables. A single call to *BatchWriteItem* can write up to 16 MB of data, which can comprise as many as 25 put or delete requests. Individual items to be written can be as large as 400 KB.

> **Note**
> BatchWriteItem cannot update items. To update items, use the UpdateItem API.

The individual *PutItem* and *DeleteItem* operations specified in *BatchWriteItem* are atomic; however *BatchWriteItem* as a whole is not. If any requested operations fail because the table's provisioned throughput is exceeded or an internal processing failure occurs, the failed operations are returned in the *UnprocessedItems* response parameter. You can investigate and optionally resend the requests. Typically, you would call *BatchWriteItem* in a loop. Each iteration would check for unprocessed items and submit a new *BatchWriteItem* request with those unprocessed items until all items have been processed.

Note that if *none* of the items can be processed due to insufficient provisioned throughput on all of the tables in the request, then *BatchWriteItem* will return a *ProvisionedThroughputExceededException*.

> **Important**
> If DynamoDB returns any unprocessed items, you should retry the batch operation on those items. However, we strongly recommend that you use an exponential backoff algorithm. If you retry the batch operation immediately, the underlying read or write requests can still fail due to throttling on the individual tables. If you delay the batch operation using exponential backoff, the individual requests in the batch are much more likely to succeed. For more information, see Batch Operations and Error Handling in the Amazon DynamoDB Developer Guide.

With *BatchWriteItem*, you can efficiently write or delete large amounts of data, such as from Amazon Elastic MapReduce (EMR), or copy data from another database into DynamoDB. In order to improve performance with these large-scale operations, *BatchWriteItem* does not behave in the same way as individual *PutItem* and *DeleteItem* calls would. For example, you cannot specify conditions on individual put and delete requests, and *BatchWriteItem* does not return deleted items in the response.

If you use a programming language that supports concurrency, you can use threads to write items in parallel. Your application must include the necessary logic to manage the threads. With languages that don't support threading, you must update or delete the specified items one at a time. In both situations, *BatchWriteItem* provides an alternative where the API performs the specified put and delete operations in parallel, giving you the power of the thread pool approach without having to introduce complexity into your application.

Parallel processing reduces latency, but each specified put and delete request consumes the same number of write capacity units whether it is processed in parallel or not. Delete operations on nonexistent items consume one write capacity unit.

If one or more of the following is true, DynamoDB rejects the entire batch write operation:

- One or more tables specified in the *BatchWriteItem* request does not exist.
- Primary key attributes specified on an item in the request do not match those in the corresponding table's primary key schema.
- You try to perform multiple operations on the same item in the same *BatchWriteItem* request. For example, you cannot put and delete the same item in the same *BatchWriteItem* request.
- There are more than 25 requests in the batch.
- Any individual item in a batch exceeds 400 KB.
- The total request size exceeds 16 MB.

Request Syntax

```
{
    "RequestItems":
        {
            "string" :
                [
                    {
                        "DeleteRequest": {
                            "Key":
                                {
                                    "string" :
                                        {
                                            "B": blob,
                                            "BOOL": boolean,
                                            "BS": [
                                                blob
                                            ],
                                            "L": [
                                                AttributeValue
                                            ],
                                            "M":
                                                {
                                                    "string" :
                                                        AttributeValue
                                                },
                                            "N": "string",
                                            "NS": [
                                                "string"
                                            ],
                                            "NULL": boolean,
                                            "S": "string",
                                            "SS": [
                                                "string"
                                            ]
                                        }
                                }
                        },
                        "PutRequest": {
                            "Item":
                                {
                                    "string" :
                                        {
                                            "B": blob,
                                            "BOOL": boolean,
                                            "BS": [
                                                blob
                                            ],
                                            "L": [
                                                AttributeValue
                                            ],
                                            "M":
                                                {
                                                    "string" :
                                                        AttributeValue
                                                },
```

```
                                        "N": "string",
                                        "NS": [
                                            "string"
                                        ],
                                        "NULL": boolean,
                                        "S": "string",
                                        "SS": [
                                            "string"
                                        ]
                                    }
                                }
                            }
                        }
                    ]
        },
    "ReturnConsumedCapacity": "string",
    "ReturnItemCollectionMetrics": "string"
}
```

Request Parameters

The request requires the following data in JSON format.

Note
In the following list, the required parameters are described first.

RequestItems
A map of one or more table names and, for each table, a list of operations to be performed (*DeleteRequest* or *PutRequest*). Each element in the map consists of the following:

- *DeleteRequest* - Perform a *DeleteItem* operation on the specified item. The item to be deleted is identified by a *Key* subelement:
  - *Key* - A map of primary key attribute values that uniquely identify the ! item. Each entry in this map consists of an attribute name and an attribute value. For each primary key, you must provide *all* of the key attributes. For example, with a hash type primary key, you only need to provide the hash attribute. For a hash-and-range type primary key, you must provide *both* the hash attribute and the range attribute.
- *PutRequest* - Perform a *PutItem* operation on the specified item. The item to be put is identified by an *Item* subelement:
  - *Item* - A map of attributes and their values. Each entry in this map consists of an attribute name and an attribute value. Attribute values must not be null; string and binary type attributes must have lengths greater than zero; and set type attributes must not be empty. Requests that contain empty values will be rejected with a *ValidationException* exception.

    If you specify any attributes that are part of an index key, then the data types for those attributes must match those of the schema in the table's attribute definition.

Type: String to WriteRequest (p. 157) object map

Length constraints: Minimum length of 1. Maximum length of 25.

Required: Yes

ReturnConsumedCapacity
Determines the level of detail about provisioned throughput consumption that is returned in the response:

- *INDEXES* - The response includes the aggregate *ConsumedCapacity* for the operation, together with *ConsumedCapacity* for each table and secondary index that was accessed.

Note that some operations, such as *GetItem* and *BatchGetItem*, do not access any indexes at all. In these cases, specifying *INDEXES* will only return *ConsumedCapacity* information for table(s).

- *TOTAL* - The response includes only the aggregate *ConsumedCapacity* for the operation.
- *NONE* - No *ConsumedCapacity* details are included in the response.

Type: String

Valid Values: `INDEXES | TOTAL | NONE`

Required: No

ReturnItemCollectionMetrics

Determines whether item collection metrics are returned. If set to `SIZE`, the response includes statistics about item collections, if any, that were modified during the operation are returned in the response. If set to `NONE` (the default), no statistics are returned.

Type: String

Valid Values: `SIZE | NONE`

Required: No

Response Syntax

```
{
    "ConsumedCapacity": [
        {
            "CapacityUnits": number,
            "GlobalSecondaryIndexes":
                {
                    "string" :
                        {
                            "CapacityUnits": number
                        }
                },
            "LocalSecondaryIndexes":
                {
                    "string" :
                        {
                            "CapacityUnits": number
                        }
                },
            "Table": {
                "CapacityUnits": number
            },
            "TableName": "string"
        }
    ],
    "ItemCollectionMetrics":
        {
            "string" :
                [
                    {
                        "ItemCollectionKey":
                            {
                                "string" :
```

```
                                {
                                    "B": blob,
                                    "BOOL": boolean,
                                    "BS": [
                                        blob
                                    ],
                                    "L": [
                                        AttributeValue
                                    ],
                                    "M":
                                        {
                                            "string" :
                                                AttributeValue
                                        },
                                    "N": "string",
                                    "NS": [
                                        "string"
                                    ],
                                    "NULL": boolean,
                                    "S": "string",
                                    "SS": [
                                        "string"
                                    ]
                                }
                            },
                        "SizeEstimateRangeGB": [
                            number
                        ]
                    }
                ]
        },
    "UnprocessedItems":
        {
            "string" :
                [
                    {
                        "DeleteRequest": {
                            "Key":
                                {
                                    "string" :
                                        {
                                            "B": blob,
                                            "BOOL": boolean,
                                            "BS": [
                                                blob
                                            ],
                                            "L": [
                                                AttributeValue
                                            ],
                                            "M":
                                                {
                                                    "string" :
                                                        AttributeValue
                                                },
                                            "N": "string",
                                            "NS": [
                                                "string"
                                            ],
```

```
                                        "NULL": boolean,
                                        "S": "string",
                                        "SS": [
                                            "string"
                                        ]
                                    }
                                }
                            },
                            "PutRequest": {
                                "Item":
                                    {
                                        "string" :
                                            {
                                                "B": blob,
                                                "BOOL": boolean,
                                                "BS": [
                                                    blob
                                                ],
                                                "L": [
                                                    AttributeValue
                                                ],
                                                "M":
                                                    {
                                                        "string" :
                                                            AttributeValue
                                                    },
                                                "N": "string",
                                                "NS": [
                                                    "string"
                                                ],
                                                "NULL": boolean,
                                                "S": "string",
                                                "SS": [
                                                    "string"
                                                ]
                                            }
                                    }
                            }
                        }
                    ]
            }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ConsumedCapacity
The capacity units consumed by the operation.

Each element consists of:

- *TableName* - The table that consumed the provisioned throughput.
- *CapacityUnits* - The total number of capacity units consumed.

Type: array of ConsumedCapacity (p. 136) objects

ItemCollectionMetrics
A list of tables that were processed by *BatchWriteItem* and, for each table, information about any item collections that were affected by individual *DeleteItem* or *PutItem* operations.

Each entry consists of the following subelements:

- *ItemCollectionKey* - The hash key value of the item collection. This is the same as the hash key of the item.
- *SizeEstimateRange* - An estimate of item collection size, expressed in GB. This is a two-element array containing a lower bound and an upper bound for the estimate. The estimate includes the size of all the items in the table, plus the size of all attributes projected into all of the local secondary indexes on the table. Use this estimate to measure whether a local secondary index is approaching its size limit.

  The estimate is subject to change over time; therefore, do not rely on the precision or accuracy of the estimate.

Type: String to ItemCollectionMetrics (p. 145) object map

UnprocessedItems
A map of tables and requests against those tables that were not processed. The *UnprocessedItems* value is in the same form as *RequestItems*, so you can provide this value directly to a subsequent *BatchGetItem* operation. For more information, see *RequestItems* in the Request Parameters section.

Each *UnprocessedItems* entry consists of a table name and, for that table, a list of operations to perform (*DeleteRequest* or *PutRequest*).

- *DeleteRequest* - Perform a *DeleteItem* operation on the specified item. The item to be deleted is identified by a *Key* subelement:
  - *Key* - A map of primary key attribute values that uniquely identify the item. Each entry in this map consists of an attribute name and an attribute value.
- *PutRequest* - Perform a *PutItem* operation on the specified item. The item to be put is identified by an *Item* subelement:
  - *Item* - A map of attributes and their values. Each entry in this map consists of an attribute name and an attribute value. Attribute values must not be null; string and binary type attributes must have lengths greater than zero; and set type attributes must not be empty. Requests that contain empty values will be rejected with a *ValidationException* exception.

    If you specify any attributes that are part of an index key, then the data types for those attributes must match those of the schema in the table's attribute definition.

If there are no unprocessed items remaining, the response contains an empty *UnprocessedItems* map.

Type: String to WriteRequest (p. 157) object map

Length constraints: Minimum length of 1. Maximum length of 25.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 159).

InternalServerError
An error occurred on the server side.

HTTP Status Code: 500

ItemCollectionSizeLimitExceededException
An item collection is too large. This exception is only returned for tables that have one or more local secondary indexes.

HTTP Status Code: 400

ProvisionedThroughputExceededException

Your request rate is too high. The AWS SDKs for DynamoDB automatically retry requests that receive this exception. Your request is eventually successful, unless your retry queue is too large to finish. Reduce the frequency of requests and use exponential backoff. For more information, go to Error Retries and Exponential Backoff in the *Amazon DynamoDB Developer Guide*.

HTTP Status Code: 400

ResourceNotFoundException

The operation tried to access a nonexistent table or index. The resource might not be specified correctly, or its status might not be `ACTIVE`.

HTTP Status Code: 400

Examples

Multiple Operations on One Table

This example writes several items to the Forum table. The response shows that the final put operation failed, possibly because the application exceeded the provisioned throughput on the table. The UnprocessedItems object shows the unsuccessful put request. The application can call BatchWriteItem again to address such unprocessed requests.

Sample Request

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date> X-Amz-Target: DynamoDB_20120810.BatchWriteItem

{
    "RequestItems": {
        "Forum": [
            {
                "PutRequest": {
                    "Item": {
                        "Name": {
                            "S": "Amazon DynamoDB"
                        },
                        "Category": {
                            "S": "Amazon Web Services"
                        }
                    }
                }
            },
            {
                "PutRequest": {
                    "Item": {
                        "Name": {
```

```
                        "S": "Amazon RDS"
                    },
                    "Category": {
                        "S": "Amazon Web Services"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Name": {
                        "S": "Amazon Redshift"
                    },
                    "Category": {
                        "S": "Amazon Web Services"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Name": {
                        "S": "Amazon ElastiCache"
                    },
                    "Category": {
                        "S": "Amazon Web Services"
                    }
                }
            }
        }
    ]
},
"ReturnConsumedCapacity": "TOTAL"
}
```

Sample Response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
 {
    "UnprocessedItems": {
        "Forum": [
            {
                "PutRequest": {
                    "Item": {
                        "Name": {
                            "S": "Amazon ElastiCache"
                        },
```

```
                        "Category": {
                            "S": "Amazon Web Services"
                        }
                    }
                }
            }
        ]
    },
    "ConsumedCapacity": [
        {
            "TableName": "Forum",
            "CapacityUnits": 3
        }
    ]
}
```

CreateTable

The *CreateTable* operation adds a new table to your account. In an AWS account, table names must be unique within each region. That is, you can have two tables with same name if you create the tables in different regions.

CreateTable is an asynchronous operation. Upon receiving a *CreateTable* request, DynamoDB immediately returns a response with a *TableStatus* of `CREATING`. After the table is created, DynamoDB sets the *TableStatus* to `ACTIVE`. You can perform read and write operations only on an `ACTIVE` table.

You can optionally define secondary indexes on the new table, as part of the *CreateTable* operation. If you want to create multiple tables with secondary indexes on them, you must create the tables sequentially. Only one table with secondary indexes can be in the `CREATING` state at any given time.

You can use the *DescribeTable* API to check the table status.

Request Syntax

```
{
    "AttributeDefinitions": [
        {
            "AttributeName": "string",
            "AttributeType": "string"
        }
    ],
    "GlobalSecondaryIndexes": [
        {
            "IndexName": "string",
            "KeySchema": [
                {
                    "AttributeName": "string",
                    "KeyType": "string"
                }
            ],
            "Projection": {
                "NonKeyAttributes": [
                    "string"
                ],
                "ProjectionType": "string"
            },
            "ProvisionedThroughput": {
                "ReadCapacityUnits": number,
                "WriteCapacityUnits": number
            }
        }
    ],
    "KeySchema": [
        {
            "AttributeName": "string",
            "KeyType": "string"
        }
    ],
    "LocalSecondaryIndexes": [
        {
            "IndexName": "string",
```

```
            "KeySchema": [
                {
                    "AttributeName": "string",
                    "KeyType": "string"
                }
            ],
            "Projection": {
                "NonKeyAttributes": [
                    "string"
                ],
                "ProjectionType": "string"
            }
        }
    ],
    "ProvisionedThroughput": {
        "ReadCapacityUnits": number,
        "WriteCapacityUnits": number
    },
    "StreamSpecification": {
        "StreamEnabled": boolean,
        "StreamViewType": "string"
    },
    "TableName": "string"
}
```

Request Parameters

The request requires the following data in JSON format.

Note
In the following list, the required parameters are described first.

AttributeDefinitions
An array of attributes that describe the key schema for the table and indexes.

Type: array of AttributeDefinition (p. 130) objects

Required: Yes

KeySchema
Specifies the attributes that make up the primary key for a table or an index. The attributes in *KeySchema* must also be defined in the *AttributeDefinitions* array. For more information, see Data Model in the *Amazon DynamoDB Developer Guide*.

Each *KeySchemaElement* in the array is composed of:

- *AttributeName* - The name of this key attribute.
- *KeyType* - Determines whether the key attribute is `HASH` or `RANGE`.

For a primary key that consists of a hash attribute, you must provide exactly one element with a *KeyType* of `HASH`.

For a primary key that consists of hash and range attributes, you must provide exactly two elements, in this order: The first element must have a *KeyType* of `HASH`, and the second element must have a *KeyType* of `RANGE`.

For more information, see Specifying the Primary Key in the *Amazon DynamoDB Developer Guide*.

Type: array of KeySchemaElement (p. 148) objects

Length constraints: Minimum of 1 item(s) in the list. Maximum of 2 item(s) in the list.

Required: Yes

ProvisionedThroughput

Represents the provisioned throughput settings for a specified table or index. The settings can be modified using the *UpdateTable* operation.

For current minimum and maximum provisioned throughput values, see Limits in the *Amazon DynamoDB Developer Guide*.

Type: ProvisionedThroughput (p. 151) object

Required: Yes

TableName

The name of the table to create.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

GlobalSecondaryIndexes

One or more global secondary indexes (the maximum is five) to be created on the table. Each global secondary index in the array includes the following:

- *IndexName* - The name of the global secondary index. Must be unique only for this table.
- *KeySchema* - Specifies the key schema for the global secondary index.
- *Projection* - Specifies attributes that are copied (projected) from the table into the index. These are in addition to the primary key attributes and index key attributes, which are automatically projected. Each attribute specification is composed of:
  - *ProjectionType* - One of the following:
    - `KEYS_ONLY` - Only the index and primary keys are projected into the index.
    - `INCLUDE` - Only the specified table attributes are projected into the index. The list of projected attributes are in *NonKeyAttributes*.
    - `ALL` - All of the table attributes are projected into the index.
  - *NonKeyAttributes* - A list of one or more non-key attribute names that are projected into the secondary index. The total count of attributes provided in *NonKeyAttributes*, summed across all of the secondary indexes, must not exceed 20. If you project the same attribute into two different indexes, this counts as two distinct attributes when determining the total.
- *ProvisionedThroughput* - The provisioned throughput settings for the global secondary index, consisting of read and write capacity units.

Type: array of GlobalSecondaryIndex (p. 142) objects

Required: No

LocalSecondaryIndexes

One or more local secondary indexes (the maximum is five) to be created on the table. Each index is scoped to a given hash key value. There is a 10 GB size limit per hash key; otherwise, the size of a local secondary index is unconstrained.

Each local secondary index in the array includes the following:

- *IndexName* - The name of the local secondary index. Must be unique only for this table.

- *KeySchema* - Specifies the key schema for the local secondary index. The key schema must begin with the same hash key attribute as the table.
- *Projection* - Specifies attributes that are copied (projected) from the table into the index. These are in addition to the primary key attributes and index key attributes, which are automatically projected. Each attribute specification is composed of:
  - *ProjectionType* - One of the following:
    - `KEYS_ONLY` - Only the index and primary keys are projected into the index.
    - `INCLUDE` - Only the specified table attributes are projected into the index. The list of projected attributes are in *NonKeyAttributes*.
    - `ALL` - All of the table attributes are projected into the index.
  - *NonKeyAttributes* - A list of one or more non-key attribute names that are projected into the secondary index. The total count of attributes provided in *NonKeyAttributes*, summed across all of the secondary indexes, must not exceed 20. If you project the same attribute into two different indexes, this counts as two distinct attributes when determining the total.

Type: array of LocalSecondaryIndex (p. 148) objects

Required: No

StreamSpecification

The settings for DynamoDB Streams on the table. These settings consist of:

- *StreamEnabled* - Indicates whether Streams is to be enabled (true) or disabled (false).
- *StreamViewType* - When an item in the table is modified, *StreamViewType* determines what information is written to the table's stream. Valid values for *StreamViewType* are:
  - *KEYS_ONLY* - Only the key attributes of the modified item are written to the stream.
  - *NEW_IMAGE* - The entire item, as it appears after it was modified, is written to the stream.
  - *OLD_IMAGE* - The entire item, as it appeared before it was modified, is written to the stream.
  - *NEW_AND_OLD_IMAGES* - Both the new and the old item images of the item are written to the stream.

Type: StreamSpecification (p. 153) object

Required: No

Response Syntax

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "string",
                "AttributeType": "string"
            }
        ],
        "CreationDateTime": number,
        "GlobalSecondaryIndexes": [
            {
                "Backfilling": boolean,
                "IndexArn": "string",
                "IndexName": "string",
                "IndexSizeBytes": number,
                "IndexStatus": "string",
                "ItemCount": number,
```

```
            "KeySchema": [
                {
                    "AttributeName": "string",
                    "KeyType": "string"
                }
            ],
            "Projection": {
                "NonKeyAttributes": [
                    "string"
                ],
                "ProjectionType": "string"
            },
            "ProvisionedThroughput": {
                "LastDecreaseDateTime": number,
                "LastIncreaseDateTime": number,
                "NumberOfDecreasesToday": number,
                "ReadCapacityUnits": number,
                "WriteCapacityUnits": number
            }
        }
    ],
    "ItemCount": number,
    "KeySchema": [
        {
            "AttributeName": "string",
            "KeyType": "string"
        }
    ],
    "LatestStreamArn": "string",
    "LatestStreamLabel": "string",
    "LocalSecondaryIndexes": [
        {
            "IndexArn": "string",
            "IndexName": "string",
            "IndexSizeBytes": number,
            "ItemCount": number,
            "KeySchema": [
                {
                    "AttributeName": "string",
                    "KeyType": "string"
                }
            ],
            "Projection": {
                "NonKeyAttributes": [
                    "string"
                ],
                "ProjectionType": "string"
            }
        }
    ],
    "ProvisionedThroughput": {
        "LastDecreaseDateTime": number,
        "LastIncreaseDateTime": number,
        "NumberOfDecreasesToday": number,
        "ReadCapacityUnits": number,
        "WriteCapacityUnits": number
    },
    "StreamSpecification": {
```

```
            "StreamEnabled": boolean,
            "StreamViewType": "string"
        },
        "TableArn": "string",
        "TableName": "string",
        "TableSizeBytes": number,
        "TableStatus": "string"
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TableDescription
Represents the properties of a table.

Type: TableDescription (p. 153) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 159).

InternalServerError
An error occurred on the server side.

HTTP Status Code: 500

LimitExceededException
The number of concurrent table requests (cumulative number of tables in the `CREATING`, `DELETING` or `UPDATING` state) exceeds the maximum allowed of 10.

Also, for tables with secondary indexes, only one of those tables can be in the `CREATING` state at any point in time. Do not attempt to create more than one such table simultaneously.

The total limit of tables in the `ACTIVE` state is 250.

HTTP Status Code: 400

ResourceInUseException
The operation conflicts with the resource's availability. For example, you attempted to recreate an existing table, or tried to delete a table currently in the `CREATING` state.

HTTP Status Code: 400

Examples

Create a Table

This example creates a table named Thread. The table primary key consists of ForumName (hash) and Subject (range). A local secondary index is also created; its key consists of ForumName (hash) and LastPostDateTime (range).

Sample Request

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date> X-Amz-Target: DynamoDB_20120810.CreateTable

{
    "AttributeDefinitions": [
        {
            "AttributeName": "ForumName",
            "AttributeType": "S"
        },
        {
            "AttributeName": "Subject",
            "AttributeType": "S"
        },
        {
            "AttributeName": "LastPostDateTime",
            "AttributeType": "S"
        }
    ],
    "TableName": "Thread",
    "KeySchema": [
        {
            "AttributeName": "ForumName",
            "KeyType": "HASH"
        },
        {
            "AttributeName": "Subject",
            "KeyType": "RANGE"
        }
    ],
    "LocalSecondaryIndexes": [
        {
            "IndexName": "LastPostIndex",
            "KeySchema": [
                {
                    "AttributeName": "ForumName",
                    "KeyType": "HASH"
                },
                {
                    "AttributeName": "LastPostDateTime",
                    "KeyType": "RANGE"
                }
            ],
            "Projection": {
                "ProjectionType": "KEYS_ONLY"
            }
        }
    ],
```

```
    "ProvisionedThroughput": {
        "ReadCapacityUnits": 5,
        "WriteCapacityUnits": 5
    }
}
```

Sample Response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
 {
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/Thread",
        "AttributeDefinitions": [
            {
                "AttributeName": "ForumName",
                "AttributeType": "S"
            },
            {
                "AttributeName": "LastPostDateTime",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Subject",
                "AttributeType": "S"
            }
        ],
        "CreationDateTime": 1.36372808007E9,
        "ItemCount": 0,
        "KeySchema": [
            {
                "AttributeName": "ForumName",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "Subject",
                "KeyType": "RANGE"
            }
        ],
        "LocalSecondaryIndexes": [
            {
                "IndexArn": "arn:aws:dynamodb:us-west-
2:123456789012:table/Thread/index/LastPostIndex",
                "IndexName": "LastPostIndex",
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "KeySchema": [
                    {
                        "AttributeName": "ForumName",
                        "KeyType": "HASH"
```

```
                    },
                    {
                        "AttributeName": "LastPostDateTime",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "KEYS_ONLY"
                }
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableName": "Thread",
        "TableSizeBytes": 0,
        "TableStatus": "CREATING"
    }
}
```

DeleteItem

Deletes a single item in a table by primary key. You can perform a conditional delete operation that deletes the item if it exists, or if it has an expected attribute value.

In addition to deleting an item, you can also return the item's attribute values in the same operation, using the *ReturnValues* parameter.

Unless you specify conditions, the *DeleteItem* is an idempotent operation; running it multiple times on the same item or attribute does *not* result in an error response.

Conditional deletes are useful for deleting items only if specific conditions are met. If those conditions are met, DynamoDB performs the delete. Otherwise, the item is not deleted.

Request Syntax

```
{
    "ConditionalOperator": "string",
    "ConditionExpression": "string",
    "Expected":
        {
            "string" :
                {
                    "AttributeValueList": [
                        {
                            "B": blob,
                            "BOOL": boolean,
                            "BS": [
                                blob
                            ],
                            "L": [
                                AttributeValue
                            ],
                            "M":
                                {
                                    "string" :
                                        AttributeValue
                                },
                            "N": "string",
                            "NS": [
                                "string"
                            ],
                            "NULL": boolean,
                            "S": "string",
                            "SS": [
                                "string"
                            ]
                        }
                    ],
                    "ComparisonOperator": "string",
                    "Exists": boolean,
                    "Value": {
                        "B": blob,
                        "BOOL": boolean,
                        "BS": [
```

```
                        blob
                    ],
                    "L": [
                        AttributeValue
                    ],
                    "M":
                        {
                            "string" :
                                AttributeValue
                        },
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
                        "string"
                    ]
                }
            }
    },
"ExpressionAttributeNames":
    {
        "string" :
            "string"
    },
"ExpressionAttributeValues":
    {
        "string" :
            {
                "B": blob,
                "BOOL": boolean,
                "BS": [
                    blob
                ],
                "L": [
                    AttributeValue
                ],
                "M":
                    {
                        "string" :
                            AttributeValue
                    },
                "N": "string",
                "NS": [
                    "string"
                ],
                "NULL": boolean,
                "S": "string",
                "SS": [
                    "string"
                ]
            }
    },
"Key":
    {
        "string" :
```

```
                {
                    "B": blob,
                    "BOOL": boolean,
                    "BS": [
                        blob
                    ],
                    "L": [
                        AttributeValue
                    ],
                    "M":
                        {
                            "string" :
                                AttributeValue
                        },
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
                        "string"
                    ]
                }
        },
    "ReturnConsumedCapacity": "string",
    "ReturnItemCollectionMetrics": "string",
    "ReturnValues": "string",
    "TableName": "string"
}
```

Request Parameters

The request requires the following data in JSON format.

Note
In the following list, the required parameters are described first.

Key
A map of attribute names to *AttributeValue* objects, representing the primary key of the item to delete.

For the primary key, you must provide all of the attributes. For example, with a hash type primary key, you only need to provide the hash attribute. For a hash-and-range type primary key, you must provide both the hash attribute and the range attribute.

Type: String to AttributeValue (p. 130) object map

Required: Yes

TableName
The name of the table from which to delete the item.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

ConditionalOperator

Important
This is a legacy parameter, for backward compatibility. New applications should use ConditionExpression instead. Do not combine legacy parameters and expression parameters in a single API call; otherwise, DynamoDB will return a ValidationException exception.

A logical operator to apply to the conditions in the *Expected* map:

- `AND` - If all of the conditions evaluate to true, then the entire map evaluates to true.
- `OR` - If at least one of the conditions evaluate to true, then the entire map evaluates to true.

If you omit *ConditionalOperator*, then `AND` is the default.

The operation will succeed only if the entire map evaluates to true.

Note
This parameter does not support attributes of type List or Map.

Type: String

Valid Values: `AND | OR`

Required: No

ConditionExpression

A condition that must be satisfied in order for a conditional *DeleteItem* to succeed.

An expression can contain any of the following:

- Functions: `attribute_exists | attribute_not_exists | attribute_type | contains | begins_with | size`

  These function names are case-sensitive.
- Comparison operators: `= | <> | < | > | <= | >= | BETWEEN | IN`
- Logical operators: `AND | OR | NOT`

For more information on condition expressions, see Specifying Conditions in the *Amazon DynamoDB Developer Guide*.

Note
ConditionExpression replaces the legacy ConditionalOperator and Expected parameters.

Type: String

Required: No

Expected

Important
This is a legacy parameter, for backward compatibility. New applications should use ConditionExpression instead. Do not combine legacy parameters and expression parameters in a single API call; otherwise, DynamoDB will return a ValidationException exception.

A map of attribute/condition pairs. *Expected* provides a conditional block for the *DeleteItem* operation.

Each element of *Expected* consists of an attribute name, a comparison operator, and one or more values. DynamoDB compares the attribute with the value(s) you supplied, using the comparison operator. For each *Expected* element, the result of the evaluation is either true or false.

If you specify more than one element in the *Expected* map, then by default all of the conditions must evaluate to true. In other words, the conditions are ANDed together. (You can use the *ConditionalOperator* parameter to OR the conditions instead. If you do this, then at least one of the conditions must evaluate to true, rather than all of them.)

If the *Expected* map evaluates to true, then the conditional operation succeeds; otherwise, it fails.

Expected contains the following:

- *AttributeValueList* - One or more values to evaluate against the supplied attribute. The number of values in the list depends on the *ComparisonOperator* being used.

  For type Number, value comparisons are numeric.

  String value comparisons for greater than, equals, or less than are based on ASCII character code values. For example, `a` is greater than `A`, and `a` is greater than `B`. For a list of code values, see http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters.

  For type Binary, DynamoDB treats each byte of the binary data as unsigned when it compares binary values.
- *ComparisonOperator* - A comparator for evaluating attributes in the *AttributeValueList*. When performing the comparison, DynamoDB uses strongly consistent reads.

  The following comparison operators are available:

  `EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN`

  The following are descriptions of each comparison operator.

  - `EQ` : Equal. `EQ` is supported for all datatypes, including lists and maps.

    AttributeValueList can contain only one *AttributeValue* element of type String, Number, Binary, String Set, Number Set, or Binary Set. If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not equal `{"NS":["6", "2", "1"]}`.

  - `NE` : Not equal. `NE` is supported for all datatypes, including lists and maps.

    AttributeValueList can contain only one *AttributeValue* of type String, Number, Binary, String Set, Number Set, or Binary Set. If an item contains an *AttributeValue* of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not equal `{"NS":["6", "2", "1"]}`.

  - `LE` : Less than or equal.

    AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

  - `LT` : Less than.

    AttributeValueList can contain only one *AttributeValue* of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

  - `GE` : Greater than or equal.

    AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `GT` : Greater than.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `NOT_NULL` : The attribute exists. `NOT_NULL` is supported for all datatypes, including lists and maps.

  Note
  This operator tests for the existence of an attribute, not its data type. If the data type of attribute "a" is null, and you evaluate it using NOT_NULL, the result is a Boolean true. This result is because the attribute "a" exists; its data type is not relevant to the NOT_NULL comparison operator.

- `NULL` : The attribute does not exist. `NULL` is supported for all datatypes, including lists and maps.

  Note
  This operator tests for the nonexistence of an attribute, not its data type. If the data type of attribute "a" is null, and you evaluate it using NULL, the result is a Boolean false. This is because the attribute "a" exists; its data type is not relevant to the NULL comparison operator.

- `CONTAINS` : Checks for a subsequence, or value in a set.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If the target attribute of the comparison is of type String, then the operator checks for a substring match. If the target attribute of the comparison is of type Binary, then the operator looks for a subsequence of the target that matches the input. If the target attribute of the comparison is a set ("SS", "NS", or "BS"), then the operator evaluates to true if it finds an exact match with any member of the set.

  CONTAINS is supported for lists: When evaluating "a `CONTAINS` b", "a" can be a list; however, "b" cannot be a set, a map, or a list.

- `NOT_CONTAINS` : Checks for absence of a subsequence, or absence of a value in a set.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If the target attribute of the comparison is a String, then the operator checks for the absence of a substring match. If the target attribute of the comparison is Binary, then the operator checks for the absence of a subsequence of the target that matches the input. If the target attribute of the comparison is a set ("SS", "NS", or "BS"), then the operator evaluates to true if it *does not* find an exact match with any member of the set.

  NOT_CONTAINS is supported for lists: When evaluating "a `NOT CONTAINS` b", "a" can be a list; however, "b" cannot be a set, a map, or a list.

- `BEGINS_WITH` : Checks for a prefix.

  AttributeValueList can contain only one *AttributeValue* of type String or Binary (not a Number or a set type). The target attribute of the comparison must be of type String or Binary (not a Number or a set type).

- `IN` : Checks for matching elements within two sets.

  AttributeValueList can contain one or more *AttributeValue* elements of type String, Number, or Binary (not a set type). These attributes are compared against an existing set type attribute of an item. If any elements of the input set are present in the item attribute, the expression evaluates to true.

- `BETWEEN` : Greater than or equal to the first value, and less than or equal to the second value.

AttributeValueList must contain two *AttributeValue* elements of the same type, either String, Number, or Binary (not a set type). A target attribute matches if the target value is greater than, or equal to, the first element and less than, or equal to, the second element. If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not compare to `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`

For usage examples of *AttributeValueList* and *ComparisonOperator*, see Legacy Conditional Parameters in the *Amazon DynamoDB Developer Guide*.

For backward compatibility with previous DynamoDB releases, the following parameters can be used instead of *AttributeValueList* and *ComparisonOperator*:

- *Value* - A value for DynamoDB to compare with an attribute.
- *Exists* - A Boolean value that causes DynamoDB to evaluate the value before attempting the conditional operation:
  - If *Exists* is `true`, DynamoDB will check to see if that attribute value already exists in the table. If it is found, then the condition evaluates to true; otherwise the condition evaluate to false.
  - If *Exists* is `false`, DynamoDB assumes that the attribute value does *not* exist in the table. If in fact the value does not exist, then the assumption is valid and the condition evaluates to true. If the value is found, despite the assumption that it does not exist, the condition evaluates to false.

  Note that the default value for *Exists* is `true`.

The *Value* and *Exists* parameters are incompatible with *AttributeValueList* and *ComparisonOperator*. Note that if you use both sets of parameters at once, DynamoDB will return a *ValidationException* exception.

Note
This parameter does not support attributes of type List or Map.

Type: String to ExpectedAttributeValue (p. 139) object map

Required: No

ExpressionAttributeNames

One or more substitution tokens for attribute names in an expression. The following are some use cases for using *ExpressionAttributeNames*:

- To access an attribute whose name conflicts with a DynamoDB reserved word.
- To create a placeholder for repeating occurrences of an attribute name in an expression.
- To prevent special characters in an attribute name from being misinterpreted in an expression.

Use the # character in an expression to dereference an attribute name. For example, consider the following attribute name:

- `Percentile`

The name of this attribute conflicts with a reserved word, so it cannot be used directly in an expression. (For the complete list of reserved words, see Reserved Words in the *Amazon DynamoDB Developer Guide*). To work around this, you could specify the following for *ExpressionAttributeNames*:

- `{"#P":"Percentile"}`

You could then use this substitution in an expression, as in this example:

- `#P = :val`

Note
Tokens that begin with the : character are expression attribute values, which are placeholders for the actual value at runtime.

For more information on expression attribute names, see Accessing Item Attributes in the *Amazon DynamoDB Developer Guide*.

Type: String to String map

Required: No

ExpressionAttributeValues

One or more values that can be substituted in an expression.

Use the **:** (colon) character in an expression to dereference an attribute value. For example, suppose that you wanted to check whether the value of the *ProductStatus* attribute was one of the following:

```
Available | Backordered | Discontinued
```

You would first need to specify *ExpressionAttributeValues* as follows:

```
{ ":avail":{"S":"Available"}, ":back":{"S":"Backordered"},
":disc":{"S":"Discontinued"} }
```

You could then use these values in an expression, such as this:

```
ProductStatus IN (:avail, :back, :disc)
```

For more information on expression attribute values, see Specifying Conditions in the *Amazon DynamoDB Developer Guide*.

Type: String to AttributeValue (p. 130) object map

Required: No

ReturnConsumedCapacity

Determines the level of detail about provisioned throughput consumption that is returned in the response:

- *INDEXES* - The response includes the aggregate *ConsumedCapacity* for the operation, together with *ConsumedCapacity* for each table and secondary index that was accessed.

  Note that some operations, such as *GetItem* and *BatchGetItem*, do not access any indexes at all. In these cases, specifying *INDEXES* will only return *ConsumedCapacity* information for table(s).
- *TOTAL* - The response includes only the aggregate *ConsumedCapacity* for the operation.
- *NONE* - No *ConsumedCapacity* details are included in the response.

Type: String

Valid Values: `INDEXES | TOTAL | NONE`

Required: No

ReturnItemCollectionMetrics

Determines whether item collection metrics are returned. If set to `SIZE`, the response includes statistics about item collections, if any, that were modified during the operation are returned in the response. If set to `NONE` (the default), no statistics are returned.

Type: String

Valid Values: `SIZE | NONE`

Required: No

ReturnValues

Use *ReturnValues* if you want to get the item attributes as they appeared before they were deleted. For *DeleteItem*, the valid values are:

- `NONE` - If *ReturnValues* is not specified, or if its value is `NONE`, then nothing is returned. (This setting is the default for *ReturnValues*.)
- `ALL_OLD` - The content of the old item is returned.

Type: String

Valid Values: `NONE | ALL_OLD | UPDATED_OLD | ALL_NEW | UPDATED_NEW`

Required: No

Response Syntax

```
{
    "Attributes":
        {
            "string" :
                {
                    "B": blob,
                    "BOOL": boolean,
                    "BS": [
                        blob
                    ],
                    "L": [
                        AttributeValue
                    ],
                    "M":
                        {
                            "string" :
                                AttributeValue
                        },
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
                        "string"
                    ]
                }
        },
    "ConsumedCapacity": {
        "CapacityUnits": number,
        "GlobalSecondaryIndexes":
            {
                "string" :
                    {
                        "CapacityUnits": number
                    }
            },
        "LocalSecondaryIndexes":
            {
                "string" :
                    {
                        "CapacityUnits": number
```

```
                    }
            },
        "Table": {
            "CapacityUnits": number
        },
        "TableName": "string"
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey":
            {
                "string" :
                    {
                        "B": blob,
                        "BOOL": boolean,
                        "BS": [
                            blob
                        ],
                        "L": [
                            AttributeValue
                        ],
                        "M":
                            {
                                "string" :
                                    AttributeValue
                            },
                        "N": "string",
                        "NS": [
                            "string"
                        ],
                        "NULL": boolean,
                        "S": "string",
                        "SS": [
                            "string"
                        ]
                    }
            },
        "SizeEstimateRangeGB": [
            number
        ]
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Attributes
A map of attribute names to *AttributeValue* objects, representing the item as it appeared before the *DeleteItem* operation. This map appears in the response only if *ReturnValues* was specified as `ALL_OLD` in the request.

Type: String to AttributeValue (p. 130) object map

ConsumedCapacity
The capacity units consumed by an operation. The data returned includes the total provisioned throughput consumed, along with statistics for the table and any indexes involved in the operation.

ConsumedCapacity is only returned if the request asked for it. For more information, see Provisioned Throughput in the *Amazon DynamoDB Developer Guide*.

Type: ConsumedCapacity (p. 136) object

ItemCollectionMetrics

Information about item collections, if any, that were affected by the operation. *ItemCollectionMetrics* is only returned if the request asked for it. If the table does not have any local secondary indexes, this information is not returned in the response.

Each *ItemCollectionMetrics* element consists of:

- *ItemCollectionKey* - The hash key value of the item collection. This is the same as the hash key of the item.
- *SizeEstimateRange* - An estimate of item collection size, in gigabytes. This value is a two-element array containing a lower bound and an upper bound for the estimate. The estimate includes the size of all the items in the table, plus the size of all attributes projected into all of the local secondary indexes on that table. Use this estimate to measure whether a local secondary index is approaching its size limit.

  The estimate is subject to change over time; therefore, do not rely on the precision or accuracy of the estimate.

Type: ItemCollectionMetrics (p. 145) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 159).

ConditionalCheckFailedException

A condition specified in the operation could not be evaluated.

HTTP Status Code: 400

InternalServerError

An error occurred on the server side.

HTTP Status Code: 500

ItemCollectionSizeLimitExceededException

An item collection is too large. This exception is only returned for tables that have one or more local secondary indexes.

HTTP Status Code: 400

ProvisionedThroughputExceededException

Your request rate is too high. The AWS SDKs for DynamoDB automatically retry requests that receive this exception. Your request is eventually successful, unless your retry queue is too large to finish. Reduce the frequency of requests and use exponential backoff. For more information, go to Error Retries and Exponential Backoff in the *Amazon DynamoDB Developer Guide*.

HTTP Status Code: 400

ResourceNotFoundException

The operation tried to access a nonexistent table or index. The resource might not be specified correctly, or its status might not be `ACTIVE`.

HTTP Status Code: 400

Examples

Delete an Item

The following example deletes an item from the Thread table, but only if that item does not already have an attribute named Replies. Because ReturnValues is set to ALL_OLD, the response contains the item as it appeared before the delete.

Sample Request

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date> X-Amz-Target: DynamoDB_20120810.DeleteItem

{
    "TableName": "Thread",
    "Key": {
        "ForumName": {
            "S": "Amazon DynamoDB"
        },
        "Subject": {
            "S": "How do I update multiple items?"
        }
    },
    "ConditionExpression": "attribute_not_exists(Replies)",
    "ReturnValues": "ALL_OLD"
}
```

Sample Response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
 {
    "Attributes": {
        "LastPostedBy": {
            "S": "fred@example.com"
        },
        "ForumName": {
            "S": "Amazon DynamoDB"
        },
        "LastPostDateTime": {
            "S": "201303201023"
```

```
        },
        "Tags": {
            "SS": ["Update","Multiple Items","HelpMe"]
        },
        "Subject": {
            "S": "How do I update multiple items?"
        },
        "Message": {
            "S": "I want to update multiple items in a single API call. What's
 the best way to do that?"
        }
    }
}
```

DeleteTable

The *DeleteTable* operation deletes a table and all of its items. After a *DeleteTable* request, the specified table is in the `DELETING` state until DynamoDB completes the deletion. If the table is in the `ACTIVE` state, you can delete it. If a table is in `CREATING` or `UPDATING` states, then DynamoDB returns a *ResourceInUseException*. If the specified table does not exist, DynamoDB returns a *ResourceNotFoundException*. If table is already in the `DELETING` state, no error is returned.

> **Note**
> DynamoDB might continue to accept data read and write operations, such as GetItem and PutItem, on a table in the DELETING state until the table deletion is complete.

When you delete a table, any indexes on that table are also deleted.

If you have DynamoDB Streams enabled on the table, then the corresponding stream on that table goes into the `DISABLED` state, and the stream is automatically deleted after 24 hours.

Use the *DescribeTable* API to check the status of the table.

Request Syntax

```
{
    "TableName": "string"
}
```

Request Parameters

The request requires the following data in JSON format.

> **Note**
> In the following list, the required parameters are described first.

TableName
The name of the table to delete.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

Response Syntax

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "string",
                "AttributeType": "string"
            }
```

```
        ],
        "CreationDateTime": number,
        "GlobalSecondaryIndexes": [
            {
                "Backfilling": boolean,
                "IndexArn": "string",
                "IndexName": "string",
                "IndexSizeBytes": number,
                "IndexStatus": "string",
                "ItemCount": number,
                "KeySchema": [
                    {
                        "AttributeName": "string",
                        "KeyType": "string"
                    }
                ],
                "Projection": {
                    "NonKeyAttributes": [
                        "string"
                    ],
                    "ProjectionType": "string"
                },
                "ProvisionedThroughput": {
                    "LastDecreaseDateTime": number,
                    "LastIncreaseDateTime": number,
                    "NumberOfDecreasesToday": number,
                    "ReadCapacityUnits": number,
                    "WriteCapacityUnits": number
                }
            }
        ],
        "ItemCount": number,
        "KeySchema": [
            {
                "AttributeName": "string",
                "KeyType": "string"
            }
        ],
        "LatestStreamArn": "string",
        "LatestStreamLabel": "string",
        "LocalSecondaryIndexes": [
            {
                "IndexArn": "string",
                "IndexName": "string",
                "IndexSizeBytes": number,
                "ItemCount": number,
                "KeySchema": [
                    {
                        "AttributeName": "string",
                        "KeyType": "string"
                    }
                ],
                "Projection": {
                    "NonKeyAttributes": [
                        "string"
                    ],
                    "ProjectionType": "string"
                }
```

```
            }
        ],
        "ProvisionedThroughput": {
            "LastDecreaseDateTime": number,
            "LastIncreaseDateTime": number,
            "NumberOfDecreasesToday": number,
            "ReadCapacityUnits": number,
            "WriteCapacityUnits": number
        },
        "StreamSpecification": {
            "StreamEnabled": boolean,
            "StreamViewType": "string"
        },
        "TableArn": "string",
        "TableName": "string",
        "TableSizeBytes": number,
        "TableStatus": "string"
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TableDescription
Represents the properties of a table.

Type: TableDescription (p. 153) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 159).

InternalServerError
An error occurred on the server side.

HTTP Status Code: 500

LimitExceededException
The number of concurrent table requests (cumulative number of tables in the `CREATING`, `DELETING` or `UPDATING` state) exceeds the maximum allowed of 10.

Also, for tables with secondary indexes, only one of those tables can be in the `CREATING` state at any point in time. Do not attempt to create more than one such table simultaneously.

The total limit of tables in the `ACTIVE` state is 250.

HTTP Status Code: 400

ResourceInUseException
The operation conflicts with the resource's availability. For example, you attempted to recreate an existing table, or tried to delete a table currently in the `CREATING` state.

HTTP Status Code: 400

ResourceNotFoundException
The operation tried to access a nonexistent table or index. The resource might not be specified correctly, or its status might not be ACTIVE.

HTTP Status Code: 400

Examples

Delete a Table

This example deletes the Reply table.

Sample Request

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date> X-Amz-Target: DynamoDB_20120810.DeleteTable

{
    "TableName": "Reply"
}
```

Sample Response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
 {
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/Reply",
        "ItemCount": 0,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableName": "Reply",
        "TableSizeBytes": 0,
        "TableStatus": "DELETING"
    }
}
```

DescribeTable

Returns information about the table, including the current status of the table, when it was created, the primary key schema, and any indexes on the table.

> **Note**
> If you issue a DescribeTable request immediately after a CreateTable request, DynamoDB might return a ResourceNotFoundException. This is because DescribeTable uses an eventually consistent query, and the metadata for your table might not be available at that moment. Wait for a few seconds, and then try the DescribeTable request again.

Request Syntax

```
{
    "TableName": "string"
}
```

Request Parameters

The request requires the following data in JSON format.

> **Note**
> In the following list, the required parameters are described first.

TableName

The name of the table to describe.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

Response Syntax

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "string",
                "AttributeType": "string"
            }
        ],
        "CreationDateTime": number,
        "GlobalSecondaryIndexes": [
            {
                "Backfilling": boolean,
                "IndexArn": "string",
                "IndexName": "string",
                "IndexSizeBytes": number,
```

```
            "IndexStatus": "string",
            "ItemCount": number,
            "KeySchema": [
                {
                    "AttributeName": "string",
                    "KeyType": "string"
                }
            ],
            "Projection": {
                "NonKeyAttributes": [
                    "string"
                ],
                "ProjectionType": "string"
            },
            "ProvisionedThroughput": {
                "LastDecreaseDateTime": number,
                "LastIncreaseDateTime": number,
                "NumberOfDecreasesToday": number,
                "ReadCapacityUnits": number,
                "WriteCapacityUnits": number
            }
        }
    ],
    "ItemCount": number,
    "KeySchema": [
        {
            "AttributeName": "string",
            "KeyType": "string"
        }
    ],
    "LatestStreamArn": "string",
    "LatestStreamLabel": "string",
    "LocalSecondaryIndexes": [
        {
            "IndexArn": "string",
            "IndexName": "string",
            "IndexSizeBytes": number,
            "ItemCount": number,
            "KeySchema": [
                {
                    "AttributeName": "string",
                    "KeyType": "string"
                }
            ],
            "Projection": {
                "NonKeyAttributes": [
                    "string"
                ],
                "ProjectionType": "string"
            }
        }
    ],
    "ProvisionedThroughput": {
        "LastDecreaseDateTime": number,
        "LastIncreaseDateTime": number,
        "NumberOfDecreasesToday": number,
        "ReadCapacityUnits": number,
        "WriteCapacityUnits": number
```

```
        },
        "StreamSpecification": {
            "StreamEnabled": boolean,
            "StreamViewType": "string"
        },
        "TableArn": "string",
        "TableName": "string",
        "TableSizeBytes": number,
        "TableStatus": "string"
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Table
Represents the properties of a table.

Type: TableDescription (p. 153) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 159).

InternalServerError
An error occurred on the server side.

HTTP Status Code: 500

ResourceNotFoundException
The operation tried to access a nonexistent table or index. The resource might not be specified correctly, or its status might not be `ACTIVE`.

HTTP Status Code: 400

Examples

Describe a Table

This example describes the Thread table.

Sample Request

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
```

```
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date> X-Amz-Target: DynamoDB_20120810.DescribeTable

{
    "TableName":"Thread"
}
```

Sample Response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
 {
    "Table": {
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/Thread",
        "AttributeDefinitions": [
            {
                "AttributeName": "ForumName",
                "AttributeType": "S"
            },
            {
                "AttributeName": "LastPostDateTime",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Subject",
                "AttributeType": "S"
            }
        ],
        "CreationDateTime": 1.363729002358E9,
        "ItemCount": 0,
        "KeySchema": [
            {
                "AttributeName": "ForumName",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "Subject",
                "KeyType": "RANGE"
            }
        ],
        "LocalSecondaryIndexes": [
            {
                "IndexArn": "arn:aws:dynamodb:us-west-
2:123456789012:table/Thread/index/LastPostIndex",
                "IndexName": "LastPostIndex",
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "KeySchema": [
                    {
```

```
                        "AttributeName": "ForumName",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "LastPostDateTime",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "KEYS_ONLY"
                }
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableName": "Thread",
        "TableSizeBytes": 0,
        "TableStatus": "ACTIVE"
    }
}
```

GetItem

The *GetItem* operation returns a set of attributes for the item with the given primary key. If there is no matching item, *GetItem* does not return any data.

GetItem provides an eventually consistent read by default. If your application requires a strongly consistent read, set *ConsistentRead* to `true`. Although a strongly consistent read might take more time than an eventually consistent read, it always returns the last updated value.

Request Syntax

```
{
    "AttributesToGet": [
        "string"
    ],
    "ConsistentRead": boolean,
    "ExpressionAttributeNames":
        {
            "string" :
                "string"
        },
    "Key":
        {
            "string" :
                {
                    "B": blob,
                    "BOOL": boolean,
                    "BS": [
                        blob
                    ],
                    "L": [
                        AttributeValue
                    ],
                    "M":
                        {
                            "string" :
                                AttributeValue
                        },
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
                        "string"
                    ]
                }
        },
    "ProjectionExpression": "string",
    "ReturnConsumedCapacity": "string",
    "TableName": "string"
}
```

Request Parameters

The request requires the following data in JSON format.

Note
In the following list, the required parameters are described first.

Key
A map of attribute names to *AttributeValue* objects, representing the primary key of the item to retrieve.

For the primary key, you must provide all of the attributes. For example, with a hash type primary key, you only need to provide the hash attribute. For a hash-and-range type primary key, you must provide both the hash attribute and the range attribute.

Type: String to AttributeValue (p. 130) object map

Required: Yes

TableName
The name of the table containing the requested item.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

AttributesToGet

Important
This is a legacy parameter, for backward compatibility. New applications should use ProjectionExpression instead. Do not combine legacy parameters and expression parameters in a single API call; otherwise, DynamoDB will return a ValidationException exception. This parameter allows you to retrieve attributes of type List or Map; however, it cannot retrieve individual elements within a List or a Map.

The names of one or more attributes to retrieve. If no attribute names are provided, then all attributes will be returned. If any of the requested attributes are not found, they will not appear in the result.

Note that *AttributesToGet* has no effect on provisioned throughput consumption. DynamoDB determines capacity units consumed based on item size, not on the amount of data that is returned to an application.

Type: array of Strings

Length constraints: Minimum of 1 item(s) in the list.

Required: No

ConsistentRead
Determines the read consistency model: If set to `true`, then the operation uses strongly consistent reads; otherwise, the operation uses eventually consistent reads.

Type: Boolean

Required: No

ExpressionAttributeNames
One or more substitution tokens for attribute names in an expression. The following are some use cases for using *ExpressionAttributeNames*:

- To access an attribute whose name conflicts with a DynamoDB reserved word.

- To create a placeholder for repeating occurrences of an attribute name in an expression.
- To prevent special characters in an attribute name from being misinterpreted in an expression.

Use the # character in an expression to dereference an attribute name. For example, consider the following attribute name:

- `Percentile`

The name of this attribute conflicts with a reserved word, so it cannot be used directly in an expression. (For the complete list of reserved words, see Reserved Words in the *Amazon DynamoDB Developer Guide*). To work around this, you could specify the following for *ExpressionAttributeNames*:

- `{"#P":"Percentile"}`

You could then use this substitution in an expression, as in this example:

- `#P = :val`

  Note
  Tokens that begin with the : character are expression attribute values, which are placeholders for the actual value at runtime.

For more information on expression attribute names, see Accessing Item Attributes in the *Amazon DynamoDB Developer Guide*.

Type: String to String map

Required: No

ProjectionExpression

A string that identifies one or more attributes to retrieve from the table. These attributes can include scalars, sets, or elements of a JSON document. The attributes in the expression must be separated by commas.

If no attribute names are specified, then all attributes will be returned. If any of the requested attributes are not found, they will not appear in the result.

For more information, see Accessing Item Attributes in the *Amazon DynamoDB Developer Guide*.

Note
ProjectionExpression replaces the legacy AttributesToGet parameter.

Type: String

Required: No

ReturnConsumedCapacity

Determines the level of detail about provisioned throughput consumption that is returned in the response:

- *INDEXES* - The response includes the aggregate *ConsumedCapacity* for the operation, together with *ConsumedCapacity* for each table and secondary index that was accessed.

  Note that some operations, such as *GetItem* and *BatchGetItem*, do not access any indexes at all. In these cases, specifying *INDEXES* will only return *ConsumedCapacity* information for table(s).
- *TOTAL* - The response includes only the aggregate *ConsumedCapacity* for the operation.
- *NONE* - No *ConsumedCapacity* details are included in the response.

Type: String

Valid Values: `INDEXES | TOTAL | NONE`

Required: No

Response Syntax

```
{
    "ConsumedCapacity": {
        "CapacityUnits": number,
        "GlobalSecondaryIndexes":
            {
                "string" :
                    {
                        "CapacityUnits": number
                    }
            },
        "LocalSecondaryIndexes":
            {
                "string" :
                    {
                        "CapacityUnits": number
                    }
            },
        "Table": {
            "CapacityUnits": number
        },
        "TableName": "string"
    },
    "Item":
        {
            "string" :
                {
                    "B": blob,
                    "BOOL": boolean,
                    "BS": [
                        blob
                    ],
                    "L": [
                        AttributeValue
                    ],
                    "M":
                        {
                            "string" :
                                AttributeValue
                        },
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
                        "string"
                    ]
                }
        }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ConsumedCapacity
The capacity units consumed by an operation. The data returned includes the total provisioned throughput consumed, along with statistics for the table and any indexes involved in the operation. *ConsumedCapacity* is only returned if the request asked for it. For more information, see Provisioned Throughput in the *Amazon DynamoDB Developer Guide*.

Type: ConsumedCapacity (p. 136) object

Item
A map of attribute names to *AttributeValue* objects, as specified by *AttributesToGet*.

Type: String to AttributeValue (p. 130) object map

Errors

For information about the errors that are common to all actions, see Common Errors (p. 159).

InternalServerError
An error occurred on the server side.

HTTP Status Code: 500

ProvisionedThroughputExceededException
Your request rate is too high. The AWS SDKs for DynamoDB automatically retry requests that receive this exception. Your request is eventually successful, unless your retry queue is too large to finish. Reduce the frequency of requests and use exponential backoff. For more information, go to Error Retries and Exponential Backoff in the *Amazon DynamoDB Developer Guide*.

HTTP Status Code: 400

ResourceNotFoundException
The operation tried to access a nonexistent table or index. The resource might not be specified correctly, or its status might not be `ACTIVE`.

HTTP Status Code: 400

Examples

Retrieve Item Attributes

The following example retrieves three attributes from the Thread table. In the response, the ConsumedCapacityUnits value is 1, because ConsistentRead is set to true. If ConsistentRead had been set to false (or not specified) for the same request, an eventually consistent read would have been used and ConsumedCapacityUnits would have been 0.5.

Sample Request

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
```

```
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date> X-Amz-Target: DynamoDB_20120810.GetItem

{
    "TableName": "Thread",
    "Key": {
        "ForumName": {
            "S": "Amazon DynamoDB"
        },
        "Subject": {
            "S": "How do I update multiple items?"
        }
    },
    "ProjectionExpression":"LastPostDateTime, Message, Tags",
    "ConsistentRead": true,
    "ReturnConsumedCapacity": "TOTAL"
}
```

Sample Response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
 {
    "ConsumedCapacity": {
        "CapacityUnits": 1,
        "TableName": "Thread"
    },
    "Item": {
        "Tags": {
            "SS": ["Update","Multiple Items","HelpMe"]
        },
        "LastPostDateTime": {
            "S": "201303190436"
        },
        "Message": {
            "S": "I want to update multiple items in a single API call. What's
 the best way to do that?"
        }
    }
}
```

ListTables

Returns an array of table names associated with the current account and endpoint. The output from *ListTables* is paginated, with each page returning a maximum of 100 table names.

Request Syntax

```
{
    "ExclusiveStartTableName": "string",
    "Limit": number
}
```

Request Parameters

The request requires the following data in JSON format.

Note
In the following list, the required parameters are described first.

ExclusiveStartTableName
The first table name that this operation will evaluate. Use the value that was returned for *LastEvaluatedTableName* in a previous operation, so that you can obtain the next page of results.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: No

Limit
A maximum number of table names to return. If this parameter is not specified, the limit is 100.

Type: Number

Valid range: Minimum value of 1. Maximum value of 100.

Required: No

Response Syntax

```
{
    "LastEvaluatedTableName": "string",
    "TableNames": [
        "string"
    ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

LastEvaluatedTableName
The name of the last table in the current page of results. Use this value as the *ExclusiveStartTableName* in a new request to obtain the next page of results, until all the table names are returned.

If you do not receive a *LastEvaluatedTableName* value in the response, this means that there are no more table names to be retrieved.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

TableNames
The names of the tables associated with the current account at the current endpoint. The maximum size of this array is 100.

If *LastEvaluatedTableName* also appears in the output, you can use this value as the *ExclusiveStartTableName* parameter in a subsequent *ListTables* request and obtain the next page of results.

Type: array of Strings

Errors

For information about the errors that are common to all actions, see Common Errors (p. 159).

InternalServerError
An error occurred on the server side.

HTTP Status Code: 500

Examples

List Tables

This example requests a list of tables, starting with a table named Forum and ending after three table names have been returned.

Sample Request

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date> X-Amz-Target: DynamoDB_20120810.ListTables

{
```

```
    "ExclusiveStartTableName": "Forum",
    "Limit": 3
}
```

Sample Response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
 {
    "LastEvaluatedTableName": "Thread",
    "TableNames": ["Forum","Reply","Thread"]
}
```

PutItem

Creates a new item, or replaces an old item with a new item. If an item that has the same primary key as the new item already exists in the specified table, the new item completely replaces the existing item. You can perform a conditional put operation (add a new item if one with the specified primary key doesn't exist), or replace an existing item if it has certain attribute values.

In addition to putting an item, you can also return the item's attribute values in the same operation, using the *ReturnValues* parameter.

When you add an item, the primary key attribute(s) are the only required attributes. Attribute values cannot be null. String and Binary type attributes must have lengths greater than zero. Set type attributes cannot be empty. Requests with empty values will be rejected with a *ValidationException* exception.

You can request that *PutItem* return either a copy of the original item (before the update) or a copy of the updated item (after the update). For more information, see the *ReturnValues* description below.

> **Note**
> To prevent a new item from replacing an existing item, use a conditional expression that contains the attribute_not_exists function with the name of the attribute being used as the HASH key for the table. Since every record must contain that attribute, the attribute_not_exists function will only succeed if no matching item exists.

For more information about using this API, see Working with Items in the *Amazon DynamoDB Developer Guide*.

Request Syntax

```
{
    "ConditionalOperator": "string",
    "ConditionExpression": "string",
    "Expected":
        {
            "string" :
                {
                    "AttributeValueList": [
                        {
                            "B": blob,
                            "BOOL": boolean,
                            "BS": [
                                blob
                            ],
                            "L": [
                                AttributeValue
                            ],
                            "M":
                                {
                                    "string" :
                                        AttributeValue
                                },
                            "N": "string",
                            "NS": [
                                "string"
                            ],
                            "NULL": boolean,
```

```
                        "S": "string",
                        "SS": [
                            "string"
                        ]
                    }
                ],
                "ComparisonOperator": "string",
                "Exists": boolean,
                "Value": {
                    "B": blob,
                    "BOOL": boolean,
                    "BS": [
                        blob
                    ],
                    "L": [
                        AttributeValue
                    ],
                    "M":
                        {
                            "string" :
                                AttributeValue
                        },
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
                        "string"
                    ]
                }
            }
    },
"ExpressionAttributeNames":
    {
        "string" :
            "string"
    },
"ExpressionAttributeValues":
    {
        "string" :
            {
                "B": blob,
                "BOOL": boolean,
                "BS": [
                    blob
                ],
                "L": [
                    AttributeValue
                ],
                "M":
                    {
                        "string" :
                            AttributeValue
                    },
                "N": "string",
                "NS": [
```

```
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
                        "string"
                    ]
                }
        },
    "Item":
        {
            "string" :
                {
                    "B": blob,
                    "BOOL": boolean,
                    "BS": [
                        blob
                    ],
                    "L": [
                        AttributeValue
                    ],
                    "M":
                        {
                            "string" :
                                AttributeValue
                        },
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
                        "string"
                    ]
                }
        },
    "ReturnConsumedCapacity": "string",
    "ReturnItemCollectionMetrics": "string",
    "ReturnValues": "string",
    "TableName": "string"
}
```

Request Parameters

The request requires the following data in JSON format.

Note
In the following list, the required parameters are described first.

Item

A map of attribute name/value pairs, one for each attribute. Only the primary key attributes are required; you can optionally provide other attribute name-value pairs for the item.

You must provide all of the attributes for the primary key. For example, with a hash type primary key, you only need to provide the hash attribute. For a hash-and-range type primary key, you must provide both the hash attribute and the range attribute.

If you specify any attributes that are part of an index key, then the data types for those attributes must match those of the schema in the table's attribute definition.

For more information about primary keys, see Primary Key in the *Amazon DynamoDB Developer Guide*.

Each element in the *Item* map is an *AttributeValue* object.

Type: String to AttributeValue (p. 130) object map

Required: Yes

TableName

The name of the table to contain the item.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

ConditionalOperator

> **Important**
> This is a legacy parameter, for backward compatibility. New applications should use ConditionExpression instead. Do not combine legacy parameters and expression parameters in a single API call; otherwise, DynamoDB will return a ValidationException exception.

A logical operator to apply to the conditions in the *Expected* map:

- `AND` - If all of the conditions evaluate to true, then the entire map evaluates to true.
- `OR` - If at least one of the conditions evaluate to true, then the entire map evaluates to true.

If you omit *ConditionalOperator*, then `AND` is the default.

The operation will succeed only if the entire map evaluates to true.

> **Note**
> This parameter does not support attributes of type List or Map.

Type: String

Valid Values: `AND | OR`

Required: No

ConditionExpression

A condition that must be satisfied in order for a conditional *PutItem* operation to succeed.

An expression can contain any of the following:

- Functions: `attribute_exists | attribute_not_exists | attribute_type | contains | begins_with | size`

  These function names are case-sensitive.
- Comparison operators: `= | <> | < | > | <= | >= | BETWEEN | IN`
- Logical operators: `AND | OR | NOT`

For more information on condition expressions, see Specifying Conditions in the *Amazon DynamoDB Developer Guide*.

> **Note**
> ConditionExpression replaces the legacy ConditionalOperator and Expected parameters.

Type: String

Required: No

Expected

Important
This is a legacy parameter, for backward compatibility. New applications should use ConditionExpression instead. Do not combine legacy parameters and expression parameters in a single API call; otherwise, DynamoDB will return a ValidationException exception.

A map of attribute/condition pairs. *Expected* provides a conditional block for the *PutItem* operation.

Note
This parameter does not support attributes of type List or Map.

Each element of *Expected* consists of an attribute name, a comparison operator, and one or more values. DynamoDB compares the attribute with the value(s) you supplied, using the comparison operator. For each *Expected* element, the result of the evaluation is either true or false.

If you specify more than one element in the *Expected* map, then by default all of the conditions must evaluate to true. In other words, the conditions are ANDed together. (You can use the *ConditionalOperator* parameter to OR the conditions instead. If you do this, then at least one of the conditions must evaluate to true, rather than all of them.)

If the *Expected* map evaluates to true, then the conditional operation succeeds; otherwise, it fails.

Expected contains the following:

- *AttributeValueList* - One or more values to evaluate against the supplied attribute. The number of values in the list depends on the *ComparisonOperator* being used.

  For type Number, value comparisons are numeric.

  String value comparisons for greater than, equals, or less than are based on ASCII character code values. For example, `a` is greater than `A`, and `a` is greater than `B`. For a list of code values, see http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters.

  For type Binary, DynamoDB treats each byte of the binary data as unsigned when it compares binary values.
- *ComparisonOperator* - A comparator for evaluating attributes in the *AttributeValueList*. When performing the comparison, DynamoDB uses strongly consistent reads.

  The following comparison operators are available:

  ```
  EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS |
  BEGINS_WITH | IN | BETWEEN
  ```

  The following are descriptions of each comparison operator.

  - `EQ` : Equal. `EQ` is supported for all datatypes, including lists and maps.

    AttributeValueList can contain only one *AttributeValue* element of type String, Number, Binary, String Set, Number Set, or Binary Set. If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not equal `{"NS":["6", "2", "1"]}`.

  - `NE` : Not equal. `NE` is supported for all datatypes, including lists and maps.

    AttributeValueList can contain only one *AttributeValue* of type String, Number, Binary, String Set, Number Set, or Binary Set. If an item contains an *AttributeValue* of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not equal `{"NS":["6", "2", "1"]}`.

- `LE` : Less than or equal.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `LT` : Less than.

  AttributeValueList can contain only one *AttributeValue* of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `GE` : Greater than or equal.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `GT` : Greater than.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `NOT_NULL` : The attribute exists. `NOT_NULL` is supported for all datatypes, including lists and maps.

  Note
  This operator tests for the existence of an attribute, not its data type. If the data type of attribute "a" is null, and you evaluate it using NOT_NULL, the result is a Boolean true. This result is because the attribute "a" exists; its data type is not relevant to the NOT_NULL comparison operator.

- `NULL` : The attribute does not exist. `NULL` is supported for all datatypes, including lists and maps.

  Note
  This operator tests for the nonexistence of an attribute, not its data type. If the data type of attribute "a" is null, and you evaluate it using NULL, the result is a Boolean false. This is because the attribute "a" exists; its data type is not relevant to the NULL comparison operator.

- `CONTAINS` : Checks for a subsequence, or value in a set.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If the target attribute of the comparison is of type String, then the operator checks for a substring match. If the target attribute of the comparison is of type Binary, then the operator looks for a subsequence of the target that matches the input. If the target attribute of the comparison is a set ("SS", "NS", or "BS"), then the operator evaluates to true if it finds an exact match with any member of the set.

  CONTAINS is supported for lists: When evaluating "a `CONTAINS` b", "a" can be a list; however, "b" cannot be a set, a map, or a list.

- `NOT_CONTAINS` : Checks for absence of a subsequence, or absence of a value in a set.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If the target attribute of the comparison is a String, then the operator checks for

the absence of a substring match. If the target attribute of the comparison is Binary, then the operator checks for the absence of a subsequence of the target that matches the input. If the target attribute of the comparison is a set ("SS", "NS", or "BS"), then the operator evaluates to true if it *does not* find an exact match with any member of the set.

NOT_CONTAINS is supported for lists: When evaluating "a `NOT CONTAINS` b", "a" can be a list; however, "b" cannot be a set, a map, or a list.

- `BEGINS_WITH` : Checks for a prefix.

  AttributeValueList can contain only one *AttributeValue* of type String or Binary (not a Number or a set type). The target attribute of the comparison must be of type String or Binary (not a Number or a set type).

- `IN` : Checks for matching elements within two sets.

  AttributeValueList can contain one or more *AttributeValue* elements of type String, Number, or Binary (not a set type). These attributes are compared against an existing set type attribute of an item. If any elements of the input set are present in the item attribute, the expression evaluates to true.

- `BETWEEN` : Greater than or equal to the first value, and less than or equal to the second value.

  AttributeValueList must contain two *AttributeValue* elements of the same type, either String, Number, or Binary (not a set type). A target attribute matches if the target value is greater than, or equal to, the first element and less than, or equal to, the second element. If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not compare to `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`

For usage examples of *AttributeValueList* and *ComparisonOperator*, see Legacy Conditional Parameters in the *Amazon DynamoDB Developer Guide*.

For backward compatibility with previous DynamoDB releases, the following parameters can be used instead of *AttributeValueList* and *ComparisonOperator*:

- *Value* - A value for DynamoDB to compare with an attribute.
- *Exists* - A Boolean value that causes DynamoDB to evaluate the value before attempting the conditional operation:
  - If *Exists* is `true`, DynamoDB will check to see if that attribute value already exists in the table. If it is found, then the condition evaluates to true; otherwise the condition evaluate to false.
  - If *Exists* is `false`, DynamoDB assumes that the attribute value does *not* exist in the table. If in fact the value does not exist, then the assumption is valid and the condition evaluates to true. If the value is found, despite the assumption that it does not exist, the condition evaluates to false.

  Note that the default value for *Exists* is `true`.

The *Value* and *Exists* parameters are incompatible with *AttributeValueList* and *ComparisonOperator*. Note that if you use both sets of parameters at once, DynamoDB will return a *ValidationException* exception.

Type: String to ExpectedAttributeValue (p. 139) object map

Required: No

ExpressionAttributeNames

One or more substitution tokens for attribute names in an expression. The following are some use cases for using *ExpressionAttributeNames*:

- To access an attribute whose name conflicts with a DynamoDB reserved word.
- To create a placeholder for repeating occurrences of an attribute name in an expression.
- To prevent special characters in an attribute name from being misinterpreted in an expression.

Use the # character in an expression to dereference an attribute name. For example, consider the following attribute name:

- `Percentile`

The name of this attribute conflicts with a reserved word, so it cannot be used directly in an expression. (For the complete list of reserved words, see Reserved Words in the *Amazon DynamoDB Developer Guide*). To work around this, you could specify the following for *ExpressionAttributeNames*:

- `{"#P":"Percentile"}`

You could then use this substitution in an expression, as in this example:

- `#P = :val`

  Note
  Tokens that begin with the : character are expression attribute values, which are placeholders for the actual value at runtime.

For more information on expression attribute names, see Accessing Item Attributes in the *Amazon DynamoDB Developer Guide*.

Type: String to String map

Required: No

ExpressionAttributeValues

One or more values that can be substituted in an expression.

Use the **:** (colon) character in an expression to dereference an attribute value. For example, suppose that you wanted to check whether the value of the *ProductStatus* attribute was one of the following:

```
Available | Backordered | Discontinued
```

You would first need to specify *ExpressionAttributeValues* as follows:

```
{ ":avail":{"S":"Available"}, ":back":{"S":"Backordered"},
":disc":{"S":"Discontinued"} }
```

You could then use these values in an expression, such as this:

```
ProductStatus IN (:avail, :back, :disc)
```

For more information on expression attribute values, see Specifying Conditions in the *Amazon DynamoDB Developer Guide*.

Type: String to AttributeValue (p. 130) object map

Required: No

ReturnConsumedCapacity

Determines the level of detail about provisioned throughput consumption that is returned in the response:

- *INDEXES* - The response includes the aggregate *ConsumedCapacity* for the operation, together with *ConsumedCapacity* for each table and secondary index that was accessed.

  Note that some operations, such as *GetItem* and *BatchGetItem*, do not access any indexes at all. In these cases, specifying *INDEXES* will only return *ConsumedCapacity* information for table(s).
- *TOTAL* - The response includes only the aggregate *ConsumedCapacity* for the operation.
- *NONE* - No *ConsumedCapacity* details are included in the response.

Type: String

Valid Values: `INDEXES | TOTAL | NONE`

Required: No

ReturnItemCollectionMetrics

Determines whether item collection metrics are returned. If set to `SIZE`, the response includes statistics about item collections, if any, that were modified during the operation are returned in the response. If set to `NONE` (the default), no statistics are returned.

Type: String

Valid Values: `SIZE | NONE`

Required: No

ReturnValues

Use *ReturnValues* if you want to get the item attributes as they appeared before they were updated with the *PutItem* request. For *PutItem*, the valid values are:

- `NONE` - If *ReturnValues* is not specified, or if its value is `NONE`, then nothing is returned. (This setting is the default for *ReturnValues*.)
- `ALL_OLD` - If *PutItem* overwrote an attribute name-value pair, then the content of the old item is returned.

  Note
  Other "Valid Values" are not relevant to PutItem.

Type: String

Valid Values: `NONE | ALL_OLD | UPDATED_OLD | ALL_NEW | UPDATED_NEW`

Required: No

Response Syntax

```
{
    "Attributes":
        {
            "string" :
                {
                    "B": blob,
                    "BOOL": boolean,
                    "BS": [
                        blob
                    ],
                    "L": [
                        AttributeValue
                    ],
                    "M":
                        {
                            "string" :
                                AttributeValue
                        },
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
```

```
                    "string"
                ]
            }
    },
"ConsumedCapacity": {
    "CapacityUnits": number,
    "GlobalSecondaryIndexes":
        {
            "string" :
                {
                    "CapacityUnits": number
                }
        },
    "LocalSecondaryIndexes":
        {
            "string" :
                {
                    "CapacityUnits": number
                }
        },
    "Table": {
        "CapacityUnits": number
    },
    "TableName": "string"
},
"ItemCollectionMetrics": {
    "ItemCollectionKey":
        {
            "string" :
                {
                    "B": blob,
                    "BOOL": boolean,
                    "BS": [
                        blob
                    ],
                    "L": [
                        AttributeValue
                    ],
                    "M":
                        {
                            "string" :
                                AttributeValue
                        },
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
                        "string"
                    ]
                }
        },
    "SizeEstimateRangeGB": [
        number
    ]
```

```
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Attributes
The attribute values as they appeared before the *PutItem* operation, but only if *ReturnValues* is specified as `ALL_OLD` in the request. Each element consists of an attribute name and an attribute value.

Type: String to AttributeValue (p. 130) object map

ConsumedCapacity
The capacity units consumed by an operation. The data returned includes the total provisioned throughput consumed, along with statistics for the table and any indexes involved in the operation. *ConsumedCapacity* is only returned if the request asked for it. For more information, see Provisioned Throughput in the *Amazon DynamoDB Developer Guide*.

Type: ConsumedCapacity (p. 136) object

ItemCollectionMetrics
Information about item collections, if any, that were affected by the operation. *ItemCollectionMetrics* is only returned if the request asked for it. If the table does not have any local secondary indexes, this information is not returned in the response.

Each *ItemCollectionMetrics* element consists of:

- *ItemCollectionKey* - The hash key value of the item collection. This is the same as the hash key of the item.
- *SizeEstimateRange* - An estimate of item collection size, in gigabytes. This value is a two-element array containing a lower bound and an upper bound for the estimate. The estimate includes the size of all the items in the table, plus the size of all attributes projected into all of the local secondary indexes on that table. Use this estimate to measure whether a local secondary index is approaching its size limit.

  The estimate is subject to change over time; therefore, do not rely on the precision or accuracy of the estimate.

Type: ItemCollectionMetrics (p. 145) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 159).

ConditionalCheckFailedException
A condition specified in the operation could not be evaluated.

HTTP Status Code: 400

InternalServerError
An error occurred on the server side.

HTTP Status Code: 500

ItemCollectionSizeLimitExceededException
An item collection is too large. This exception is only returned for tables that have one or more local secondary indexes.

HTTP Status Code: 400

ProvisionedThroughputExceededException
Your request rate is too high. The AWS SDKs for DynamoDB automatically retry requests that receive this exception. Your request is eventually successful, unless your retry queue is too large to finish. Reduce the frequency of requests and use exponential backoff. For more information, go to Error Retries and Exponential Backoff in the *Amazon DynamoDB Developer Guide*.

HTTP Status Code: 400

ResourceNotFoundException
The operation tried to access a nonexistent table or index. The resource might not be specified correctly, or its status might not be `ACTIVE`.

HTTP Status Code: 400

Examples

Put an Item

The following example puts a new item into the Thread table, but only if there is not already an item in the table with the same key.

Sample Request

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date> X-Amz-Target: DynamoDB_20120810.PutItem

{
    "TableName": "Thread",
    "Item": {
        "LastPostDateTime": {
            "S": "201303190422"
        },
        "Tags": {
            "SS": ["Update","Multiple Items","HelpMe"]
        },
        "ForumName": {
            "S": "Amazon DynamoDB"
        },
        "Message": {
            "S": "I want to update multiple items in a single API call. What's
 the best way to do that?"
        },
        "Subject": {
```

```
            "S": "How do I update multiple items?"
        },
        "LastPostedBy": {
            "S": "fred@example.com"
        }
    },
    "ConditionExpression": "ForumName <> :f and Subject <> :s",
    "ExpressionAttributeValues": {
        ":f": {"S": "Amazon DynamoDB"},
        ":s": {"S": "How do I update multiple items?"}
    }
}
```

Sample Response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
 {
}
```

Query

A *Query* operation uses the primary key of a table or a secondary index to directly access items from that table or index.

Use the *KeyConditionExpression* parameter to provide a specific hash key value. The *Query* operation will return all of the items from the table or index with that hash key value. You can optionally narrow the scope of the *Query* operation by specifying a range key value and a comparison operator in *KeyConditionExpression*. You can use the *ScanIndexForward* parameter to get results in forward or reverse order, by range key or by index key.

Queries that do not return results consume the minimum number of read capacity units for that type of read operation.

If the total number of items meeting the query criteria exceeds the result set size limit of 1 MB, the query stops and results are returned to the user with the *LastEvaluatedKey* element to continue the query in a subsequent operation. Unlike a *Scan* operation, a *Query* operation never returns both an empty result set and a *LastEvaluatedKey* value. *LastEvaluatedKey* is only provided if the results exceed 1 MB, or if you have used the *Limit* parameter.

You can query a table, a local secondary index, or a global secondary index. For a query on a table or on a local secondary index, you can set the *ConsistentRead* parameter to `true` and obtain a strongly consistent result. Global secondary indexes support eventually consistent reads only, so do not specify *ConsistentRead* when querying a global secondary index.

Request Syntax

```
{
    "AttributesToGet": [
        "string"
    ],
    "ConditionalOperator": "string",
    "ConsistentRead": boolean,
    "ExclusiveStartKey":
        {
            "string" :
                {
                    "B": blob,
                    "BOOL": boolean,
                    "BS": [
                        blob
                    ],
                    "L": [
                        AttributeValue
                    ],
                    "M":
                        {
                            "string" :
                                AttributeValue
                        },
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
```

```
                "S": "string",
                "SS": [
                    "string"
                ]
            }
    },
"ExpressionAttributeNames":
    {
        "string" :
            "string"
    },
"ExpressionAttributeValues":
    {
        "string" :
            {
                "B": blob,
                "BOOL": boolean,
                "BS": [
                    blob
                ],
                "L": [
                    AttributeValue
                ],
                "M":
                    {
                        "string" :
                            AttributeValue
                    },
                "N": "string",
                "NS": [
                    "string"
                ],
                "NULL": boolean,
                "S": "string",
                "SS": [
                    "string"
                ]
            }
    },
"FilterExpression": "string",
"IndexName": "string",
"KeyConditionExpression": "string",
"KeyConditions":
    {
        "string" :
            {
                "AttributeValueList": [
                    {
                        "B": blob,
                        "BOOL": boolean,
                        "BS": [
                            blob
                        ],
                        "L": [
                            AttributeValue
                        ],
                        "M":
                            {
```

```
                                    "string" :
                                        AttributeValue
                                },
                            "N": "string",
                            "NS": [
                                "string"
                            ],
                            "NULL": boolean,
                            "S": "string",
                            "SS": [
                                "string"
                            ]
                        }
                    ],
                    "ComparisonOperator": "string"
                }
        },
    "Limit": number,
    "ProjectionExpression": "string",
    "QueryFilter":
        {
            "string" :
                {
                    "AttributeValueList": [
                        {
                            "B": blob,
                            "BOOL": boolean,
                            "BS": [
                                blob
                            ],
                            "L": [
                                AttributeValue
                            ],
                            "M":
                                {
                                    "string" :
                                        AttributeValue
                                },
                            "N": "string",
                            "NS": [
                                "string"
                            ],
                            "NULL": boolean,
                            "S": "string",
                            "SS": [
                                "string"
                            ]
                        }
                    ],
                    "ComparisonOperator": "string"
                }
        },
    "ReturnConsumedCapacity": "string",
    "ScanIndexForward": boolean,
    "Select": "string",
    "TableName": "string"
}
```

Request Parameters

The request requires the following data in JSON format.

Note
In the following list, the required parameters are described first.

TableName
The name of the table containing the requested items.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

AttributesToGet

Important
This is a legacy parameter, for backward compatibility. New applications should use ProjectionExpression instead. Do not combine legacy parameters and expression parameters in a single API call; otherwise, DynamoDB will return a ValidationException exception. This parameter allows you to retrieve attributes of type List or Map; however, it cannot retrieve individual elements within a List or a Map.

The names of one or more attributes to retrieve. If no attribute names are provided, then all attributes will be returned. If any of the requested attributes are not found, they will not appear in the result.

Note that *AttributesToGet* has no effect on provisioned throughput consumption. DynamoDB determines capacity units consumed based on item size, not on the amount of data that is returned to an application.

You cannot use both *AttributesToGet* and *Select* together in a *Query* request, *unless* the value for *Select* is `SPECIFIC_ATTRIBUTES`. (This usage is equivalent to specifying *AttributesToGet* without any value for *Select*.)

If you query a local secondary index and request only attributes that are projected into that index, the operation will read only the index and not the table. If any of the requested attributes are not projected into the local secondary index, DynamoDB will fetch each of these attributes from the parent table. This extra fetching incurs additional throughput cost and latency.

If you query a global secondary index, you can only request attributes that are projected into the index. Global secondary index queries cannot fetch attributes from the parent table.

Type: array of Strings

Length constraints: Minimum of 1 item(s) in the list.

Required: No

ConditionalOperator

Important
This is a legacy parameter, for backward compatibility. New applications should use FilterExpression instead. Do not combine legacy parameters and expression parameters in a single API call; otherwise, DynamoDB will return a ValidationException exception.

A logical operator to apply to the conditions in a *QueryFilter* map:

- `AND` - If all of the conditions evaluate to true, then the entire map evaluates to true.
- `OR` - If at least one of the conditions evaluate to true, then the entire map evaluates to true.

If you omit *ConditionalOperator*, then `AND` is the default.

The operation will succeed only if the entire map evaluates to true.

Note
This parameter does not support attributes of type List or Map.

Type: String

Valid Values: `AND` | `OR`

Required: No

ConsistentRead
Determines the read consistency model: If set to `true`, then the operation uses strongly consistent reads; otherwise, the operation uses eventually consistent reads.

Strongly consistent reads are not supported on global secondary indexes. If you query a global secondary index with *ConsistentRead* set to `true`, you will receive a *ValidationException*.

Type: Boolean

Required: No

ExclusiveStartKey
The primary key of the first item that this operation will evaluate. Use the value that was returned for *LastEvaluatedKey* in the previous operation.

The data type for *ExclusiveStartKey* must be String, Number or Binary. No set data types are allowed.

Type: String to AttributeValue (p. 130) object map

Required: No

ExpressionAttributeNames
One or more substitution tokens for attribute names in an expression. The following are some use cases for using *ExpressionAttributeNames*:

- To access an attribute whose name conflicts with a DynamoDB reserved word.
- To create a placeholder for repeating occurrences of an attribute name in an expression.
- To prevent special characters in an attribute name from being misinterpreted in an expression.

Use the # character in an expression to dereference an attribute name. For example, consider the following attribute name:

- `Percentile`

The name of this attribute conflicts with a reserved word, so it cannot be used directly in an expression. (For the complete list of reserved words, see Reserved Words in the *Amazon DynamoDB Developer Guide*). To work around this, you could specify the following for *ExpressionAttributeNames*:

- `{"#P":"Percentile"}`

You could then use this substitution in an expression, as in this example:

- `#P = :val`

Note
Tokens that begin with the : character are expression attribute values, which are placeholders for the actual value at runtime.

For more information on expression attribute names, see Accessing Item Attributes in the *Amazon DynamoDB Developer Guide*.

Type: String to String map

Required: No

ExpressionAttributeValues

One or more values that can be substituted in an expression.

Use the **:** (colon) character in an expression to dereference an attribute value. For example, suppose that you wanted to check whether the value of the *ProductStatus* attribute was one of the following:

```
Available | Backordered | Discontinued
```

You would first need to specify *ExpressionAttributeValues* as follows:

```
{ ":avail":{"S":"Available"}, ":back":{"S":"Backordered"},
":disc":{"S":"Discontinued"} }
```

You could then use these values in an expression, such as this:

```
ProductStatus IN (:avail, :back, :disc)
```

For more information on expression attribute values, see Specifying Conditions in the *Amazon DynamoDB Developer Guide*.

Type: String to AttributeValue (p. 130) object map

Required: No

FilterExpression

A string that contains conditions that DynamoDB applies after the *Query* operation, but before the data is returned to you. Items that do not satisfy the *FilterExpression* criteria are not returned.

> **Note**
> A FilterExpression is applied after the items have already been read; the process of filtering does not consume any additional read capacity units.

For more information, see Filter Expressions in the *Amazon DynamoDB Developer Guide*.

> **Note**
> FilterExpression replaces the legacy QueryFilter and ConditionalOperator parameters.

Type: String

Required: No

IndexName

The name of an index to query. This index can be any local secondary index or global secondary index on the table. Note that if you use the *IndexName* parameter, you must also provide *TableName.*

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: No

KeyConditionExpression

The condition that specifies the key value(s) for items to be retrieved by the *Query* action.

The condition must perform an equality test on a single hash key value. The condition can also perform one of several comparison tests on a single range key value. *Query* can use *KeyConditionExpression* to retrieve one item with a given hash and range key value, or several items that have the same hash key value but different range key values.

The hash key equality test is required, and must be specified in the following format:

`hashAttributeName = :hashval`

If you also want to provide a range key condition, it must be combined using *AND* with the hash key condition. Following is an example, using the **=** comparison operator for the range key:

`hashAttributeName = :hashval` *AND* `rangeAttributeName = :rangeval`

Valid comparisons for the range key condition are as follows:

- `rangeAttributeName = :rangeval` - true if the range key is equal to `:rangeval`.
- `rangeAttributeName < :rangeval` - true if the range key is less than `:rangeval`.
- `rangeAttributeName <= :rangeval` - true if the range key is less than or equal to `:rangeval`.
- `rangeAttributeName > :rangeval` - true if the range key is greater than `:rangeval`.
- `rangeAttributeName >= :rangeval` - true if the range key is greater than or equal to `:rangeval`.
- `rangeAttributeName` *BETWEEN* `:rangeval1` *AND* `:rangeval2` - true if the range key is greater than or equal to `:rangeval1`, and less than or equal to `:rangeval2`.
- *begins_with (* `rangeAttributeName`, `:rangeval` *)* - true if the range key begins with a particular operand. (You cannot use this function with a range key that is of type Number.) Note that the function name `begins_with` is case-sensitive.

Use the *ExpressionAttributeValues* parameter to replace tokens such as `:hashval` and `:rangeval` with actual values at runtime.

You can optionally use the *ExpressionAttributeNames* parameter to replace the names of the hash and range attributes with placeholder tokens. This option might be necessary if an attribute name conflicts with a DynamoDB reserved word. For example, the following *KeyConditionExpression* parameter causes an error because *Size* is a reserved word:

- `Size = :myval`

To work around this, define a placeholder (such a `#S`) to represent the attribute name *Size*. *KeyConditionExpression* then is as follows:

- `#S = :myval`

For a list of reserved words, see Reserved Words in the *Amazon DynamoDB Developer Guide*.

For more information on *ExpressionAttributeNames* and *ExpressionAttributeValues*, see Using Placeholders for Attribute Names and Values in the *Amazon DynamoDB Developer Guide*.

> **Note**
> KeyConditionExpression replaces the legacy KeyConditions parameter.

Type: String

Required: No

KeyConditions

> **Important**
> This is a legacy parameter, for backward compatibility. New applications should use KeyConditionExpression instead. Do not combine legacy parameters and expression parameters in a single API call; otherwise, DynamoDB will return a ValidationException exception.

The selection criteria for the query. For a query on a table, you can have conditions only on the table primary key attributes. You must provide the hash key attribute name and value as an `EQ` condition. You can optionally provide a second condition, referring to the range key attribute.

Note
If you don't provide a range key condition, all of the items that match the hash key will be retrieved. If a FilterExpression or QueryFilter is present, it will be applied after the items are retrieved.

For a query on an index, you can have conditions only on the index key attributes. You must provide the index hash attribute name and value as an `EQ` condition. You can optionally provide a second condition, referring to the index key range attribute.

Each *KeyConditions* element consists of an attribute name to compare, along with the following:

- *AttributeValueList* - One or more values to evaluate against the supplied attribute. The number of values in the list depends on the *ComparisonOperator* being used.

  For type Number, value comparisons are numeric.

  String value comparisons for greater than, equals, or less than are based on ASCII character code values. For example, `a` is greater than `A`, and `a` is greater than `B`. For a list of code values, see http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters.

  For Binary, DynamoDB treats each byte of the binary data as unsigned when it compares binary values.
- *ComparisonOperator* - A comparator for evaluating attributes, for example, equals, greater than, less than, and so on.

  For *KeyConditions*, only the following comparison operators are supported:

  ```
  EQ | LE | LT | GE | GT | BEGINS_WITH | BETWEEN
  ```

  The following are descriptions of these comparison operators.

  - `EQ` : Equal.

    AttributeValueList can contain only one *AttributeValue* of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one specified in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not equal `{"NS":["6", "2", "1"]}`.

  - `LE` : Less than or equal.

    AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

  - `LT` : Less than.

    AttributeValueList can contain only one *AttributeValue* of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

  - `GE` : Greater than or equal.

    AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

  - `GT` : Greater than.

AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `BEGINS_WITH` : Checks for a prefix.

  AttributeValueList can contain only one *AttributeValue* of type String or Binary (not a Number or a set type). The target attribute of the comparison must be of type String or Binary (not a Number or a set type).

- `BETWEEN` : Greater than or equal to the first value, and less than or equal to the second value.

  AttributeValueList must contain two *AttributeValue* elements of the same type, either String, Number, or Binary (not a set type). A target attribute matches if the target value is greater than, or equal to, the first element and less than, or equal to, the second element. If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not compare to `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`

For usage examples of *AttributeValueList* and *ComparisonOperator*, see Legacy Conditional Parameters in the *Amazon DynamoDB Developer Guide*.

Type: String to Condition (p. 134) object map

Required: No

Limit

The maximum number of items to evaluate (not necessarily the number of matching items). If DynamoDB processes the number of items up to the limit while processing the results, it stops the operation and returns the matching values up to that point, and a key in *LastEvaluatedKey* to apply in a subsequent operation, so that you can pick up where you left off. Also, if the processed data set size exceeds 1 MB before DynamoDB reaches this limit, it stops the operation and returns the matching values up to the limit, and a key in *LastEvaluatedKey* to apply in a subsequent operation to continue the operation. For more information, see Query and Scan in the *Amazon DynamoDB Developer Guide*.

Type: Number

Valid range: Minimum value of 1.

Required: No

ProjectionExpression

A string that identifies one or more attributes to retrieve from the table. These attributes can include scalars, sets, or elements of a JSON document. The attributes in the expression must be separated by commas.

If no attribute names are specified, then all attributes will be returned. If any of the requested attributes are not found, they will not appear in the result.

For more information, see Accessing Item Attributes in the *Amazon DynamoDB Developer Guide*.

Note
ProjectionExpression replaces the legacy AttributesToGet parameter.

Type: String

Required: No

QueryFilter

Important
This is a legacy parameter, for backward compatibility. New applications should use FilterExpression instead. Do not combine legacy parameters and expression parameters in a single API call; otherwise, DynamoDB will return a ValidationException exception.

A condition that evaluates the query results after the items are read and returns only the desired values.

This parameter does not support attributes of type List or Map.

Note
A QueryFilter is applied after the items have already been read; the process of filtering does not consume any additional read capacity units.

If you provide more than one condition in the *QueryFilter* map, then by default all of the conditions must evaluate to true. In other words, the conditions are ANDed together. (You can use the *ConditionalOperator* parameter to OR the conditions instead. If you do this, then at least one of the conditions must evaluate to true, rather than all of them.)

Note that *QueryFilter* does not allow key attributes. You cannot define a filter condition on a hash key or range key.

Each *QueryFilter* element consists of an attribute name to compare, along with the following:

- *AttributeValueList* - One or more values to evaluate against the supplied attribute. The number of values in the list depends on the operator specified in *ComparisonOperator*.

  For type Number, value comparisons are numeric.

  String value comparisons for greater than, equals, or less than are based on ASCII character code values. For example, a is greater than A, and a is greater than B. For a list of code values, see http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters.

  For type Binary, DynamoDB treats each byte of the binary data as unsigned when it compares binary values.

  For information on specifying data types in JSON, see JSON Data Format in the *Amazon DynamoDB Developer Guide*.
- *ComparisonOperator* - A comparator for evaluating attributes. For example, equals, greater than, less than, etc.

  The following comparison operators are available:

  ```
  EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS |
  BEGINS_WITH | IN | BETWEEN
  ```

  For complete descriptions of all comparison operators, see the Condition data type.

Type: String to Condition (p. 134) object map

Required: No

ReturnConsumedCapacity

Determines the level of detail about provisioned throughput consumption that is returned in the response:

- *INDEXES* - The response includes the aggregate *ConsumedCapacity* for the operation, together with *ConsumedCapacity* for each table and secondary index that was accessed.

  Note that some operations, such as *GetItem* and *BatchGetItem*, do not access any indexes at all. In these cases, specifying *INDEXES* will only return *ConsumedCapacity* information for table(s).

- *TOTAL* - The response includes only the aggregate *ConsumedCapacity* for the operation.
- *NONE* - No *ConsumedCapacity* details are included in the response.

Type: String

Valid Values: `INDEXES | TOTAL | NONE`

Required: No

ScanIndexForward

Specifies the order for index traversal: If `true` (default), the traversal is performed in ascending order; if `false`, the traversal is performed in descending order.

Items with the same hash key are stored in sorted order by range key. If the range key data type is Number, the results are stored in numeric order. For type String, the results are stored in order of ASCII character code values. For type Binary, DynamoDB treats each byte of the binary data as unsigned.

If *ScanIndexForward* is `true`, DynamoDB returns the results in the order in which they are stored (by range key). This is the default behavior. If *ScanIndexForward* is `false`, DynamoDB reads the results in reverse order by range key, and then returns the results to the client.

Type: Boolean

Required: No

Select

The attributes to be returned in the result. You can retrieve all item attributes, specific item attributes, the count of matching items, or in the case of an index, some or all of the attributes projected into the index.

- `ALL_ATTRIBUTES` - Returns all of the item attributes from the specified table or index. If you query a local secondary index, then for each matching item in the index DynamoDB will fetch the entire item from the parent table. If the index is configured to project all item attributes, then all of the data can be obtained from the local secondary index, and no fetching is required.
- `ALL_PROJECTED_ATTRIBUTES` - Allowed only when querying an index. Retrieves all attributes that have been projected into the index. If the index is configured to project all attributes, this return value is equivalent to specifying `ALL_ATTRIBUTES`.
- `COUNT` - Returns the number of matching items, rather than the matching items themselves.
- `SPECIFIC_ATTRIBUTES` - Returns only the attributes listed in *AttributesToGet*. This return value is equivalent to specifying *AttributesToGet* without specifying any value for *Select*.

  If you query a local secondary index and request only attributes that are projected into that index, the operation will read only the index and not the table. If any of the requested attributes are not projected into the local secondary index, DynamoDB will fetch each of these attributes from the parent table. This extra fetching incurs additional throughput cost and latency.

  If you query a global secondary index, you can only request attributes that are projected into the index. Global secondary index queries cannot fetch attributes from the parent table.

If neither *Select* nor *AttributesToGet* are specified, DynamoDB defaults to `ALL_ATTRIBUTES` when accessing a table, and `ALL_PROJECTED_ATTRIBUTES` when accessing an index. You cannot use both *Select* and *AttributesToGet* together in a single request, unless the value for *Select* is `SPECIFIC_ATTRIBUTES`. (This usage is equivalent to specifying *AttributesToGet* without any value for *Select*.)

> **Note**
> If you use the ProjectionExpression parameter, then the value for Select can only be SPECIFIC_ATTRIBUTES. Any other value for Select will return an error.

Type: String

Valid Values: `ALL_ATTRIBUTES | ALL_PROJECTED_ATTRIBUTES | SPECIFIC_ATTRIBUTES | COUNT`

Required: No

Response Syntax

```
{
    "ConsumedCapacity": {
        "CapacityUnits": number,
        "GlobalSecondaryIndexes":
            {
                "string" :
                    {
                        "CapacityUnits": number
                    }
            },
        "LocalSecondaryIndexes":
            {
                "string" :
                    {
                        "CapacityUnits": number
                    }
            },
        "Table": {
            "CapacityUnits": number
        },
        "TableName": "string"
    },
    "Count": number,
    "Items": [

            {
                "string" :
                    {
                        "B": blob,
                        "BOOL": boolean,
                        "BS": [
                            blob
                        ],
                        "L": [
                            AttributeValue
                        ],
                        "M":
                            {
                                "string" :
                                    AttributeValue
                            },
                        "N": "string",
                        "NS": [
                            "string"
                        ],
                        "NULL": boolean,
                        "S": "string",
                        "SS": [
```

```
                            "string"
                        ]
                    }
            }
    ],
    "LastEvaluatedKey":
        {
            "string" :
                {
                    "B": blob,
                    "BOOL": boolean,
                    "BS": [
                        blob
                    ],
                    "L": [
                        AttributeValue
                    ],
                    "M":
                        {
                            "string" :
                                AttributeValue
                        },
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
                        "string"
                    ]
                }
        },
    "ScannedCount": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ConsumedCapacity
The capacity units consumed by an operation. The data returned includes the total provisioned throughput consumed, along with statistics for the table and any indexes involved in the operation. *ConsumedCapacity* is only returned if the request asked for it. For more information, see Provisioned Throughput in the *Amazon DynamoDB Developer Guide*.

Type: ConsumedCapacity (p. 136) object

Count
The number of items in the response.

If you used a *QueryFilter* in the request, then *Count* is the number of items returned after the filter was applied, and *ScannedCount* is the number of matching items before> the filter was applied.

If you did not use a filter in the request, then *Count* and *ScannedCount* are the same.

Type: Number

Items

An array of item attributes that match the query criteria. Each element in this array consists of an attribute name and the value for that attribute.

Type: array of s

LastEvaluatedKey

The primary key of the item where the operation stopped, inclusive of the previous result set. Use this value to start a new operation, excluding this value in the new request.

If *LastEvaluatedKey* is empty, then the "last page" of results has been processed and there is no more data to be retrieved.

If *LastEvaluatedKey* is not empty, it does not necessarily mean that there is more data in the result set. The only way to know when you have reached the end of the result set is when *LastEvaluatedKey* is empty.

Type: String to AttributeValue (p. 130) object map

ScannedCount

The number of items evaluated, before any *QueryFilter* is applied. A high *ScannedCount* value with few, or no, *Count* results indicates an inefficient *Query* operation. For more information, see Count and ScannedCount in the *Amazon DynamoDB Developer Guide*.

If you did not use a filter in the request, then *ScannedCount* is the same as *Count*.

Type: Number

Errors

For information about the errors that are common to all actions, see Common Errors (p. 159).

InternalServerError

An error occurred on the server side.

HTTP Status Code: 500

ProvisionedThroughputExceededException

Your request rate is too high. The AWS SDKs for DynamoDB automatically retry requests that receive this exception. Your request is eventually successful, unless your retry queue is too large to finish. Reduce the frequency of requests and use exponential backoff. For more information, go to Error Retries and Exponential Backoff in the *Amazon DynamoDB Developer Guide*.

HTTP Status Code: 400

ResourceNotFoundException

The operation tried to access a nonexistent table or index. The resource might not be specified correctly, or its status might not be `ACTIVE`.

HTTP Status Code: 400

Examples

Retrieve a Range of Items

The following example queries the Reply table for replies in a forum that were posted by particular users. There is a local secondary index on the Reply table, PostedBy-Index, to facilitate fast lookups on the these attributes. The ProjectionExpression parameter determines which attributes are returned.

Sample Request

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date> X-Amz-Target: DynamoDB_20120810.Query

{
    "TableName": "Reply",
    "IndexName": "PostedBy-Index",
    "Limit": 3,
    "ConsistentRead": true,
    "ProjectionExpression": "Id, PostedBy, ReplyDateTime",
    "KeyConditionExpression": "Id = :v1 AND PostedBy BETWEEN :v2a AND :v2b",
    "ExpressionAttributeValues": {
        ":v1": {"S": "Amazon DynamoDB#DynamoDB Thread 1"},
        ":v2a": {"S": "User A"},
        ":v2b": {"S": "User C"}
    },
    "ReturnConsumedCapacity": "TOTAL"
}
```

Sample Response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
 {
    "ConsumedCapacity": {
        "CapacityUnits": 1,
        "TableName": "Reply"
    },
    "Count": 2,
    "Items": [
        {
            "ReplyDateTime": {"S": "2015-02-18T20:27:36.165Z"},
            "PostedBy": {"S": "User A"},
            "Id": {"S": "Amazon DynamoDB#DynamoDB Thread 1"}
        },
        {
            "ReplyDateTime": {"S": "2015-02-25T20:27:36.165Z"},
            "PostedBy": {"S": "User B"},
            "Id": {"S": "Amazon DynamoDB#DynamoDB Thread 1"}
        }
    ],
```

```
    "ScannedCount": 2
}
```

Count Items

The following example returns the number of items in the Thread table for a particular forum.

Sample Request

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date> X-Amz-Target: DynamoDB_20120810.Query

{
    "TableName": "Thread",
    "ConsistentRead": true,
    "KeyConditionExpression": "ForumName = :val",
    "ExpressionAttributeValues": {":val": {"S": "Amazon DynamoDB"}}
}
```

Sample Response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
 {
          "Count":2,
          "ScannedCount":2
}
```

Scan

The *Scan* operation returns one or more items and item attributes by accessing every item in a table or a secondary index. To have DynamoDB return fewer items, you can provide a *ScanFilter* operation.

If the total number of scanned items exceeds the maximum data set size limit of 1 MB, the scan stops and results are returned to the user as a *LastEvaluatedKey* value to continue the scan in a subsequent operation. The results also include the number of items exceeding the limit. A scan can result in no table data meeting the filter criteria.

By default, *Scan* operations proceed sequentially; however, for faster performance on a large table or secondary index, applications can request a parallel *Scan* operation by providing the *Segment* and *TotalSegments* parameters. For more information, see Parallel Scan in the *Amazon DynamoDB Developer Guide*.

By default, *Scan* uses eventually consistent reads when acessing the data in a table; therefore, the result set might not include the changes to data in the table immediately before the operation began. If you need a consistent copy of the data, as of the time that the Scan begins, you can set the *ConsistentRead* parameter to *true*.

Request Syntax

```
{
    "AttributesToGet": [
        "string"
    ],
    "ConditionalOperator": "string",
    "ConsistentRead": boolean,
    "ExclusiveStartKey":
        {
            "string" :
                {
                    "B": blob,
                    "BOOL": boolean,
                    "BS": [
                        blob
                    ],
                    "L": [
                        AttributeValue
                    ],
                    "M":
                        {
                            "string" :
                                AttributeValue
                        },
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
                        "string"
                    ]
                }
```

```
    },
"ExpressionAttributeNames":
    {
        "string" :
            "string"
    },
"ExpressionAttributeValues":
    {
        "string" :
            {
                "B": blob,
                "BOOL": boolean,
                "BS": [
                    blob
                ],
                "L": [
                    AttributeValue
                ],
                "M":
                    {
                        "string" :
                            AttributeValue
                    },
                "N": "string",
                "NS": [
                    "string"
                ],
                "NULL": boolean,
                "S": "string",
                "SS": [
                    "string"
                ]
            }
    },
"FilterExpression": "string",
"IndexName": "string",
"Limit": number,
"ProjectionExpression": "string",
"ReturnConsumedCapacity": "string",
"ScanFilter":
    {
        "string" :
            {
                "AttributeValueList": [
                    {
                        "B": blob,
                        "BOOL": boolean,
                        "BS": [
                            blob
                        ],
                        "L": [
                            AttributeValue
                        ],
                        "M":
                            {
                                "string" :
                                    AttributeValue
                            },
```

```
                            "N": "string",
                            "NS": [
                                "string"
                            ],
                            "NULL": boolean,
                            "S": "string",
                            "SS": [
                                "string"
                            ]
                        }
                    ],
                    "ComparisonOperator": "string"
                }
        },
    "Segment": number,
    "Select": "string",
    "TableName": "string",
    "TotalSegments": number
}
```

Request Parameters

The request requires the following data in JSON format.

Note
In the following list, the required parameters are described first.

TableName
The name of the table containing the requested items; or, if you provide `IndexName`, the name of the table to which that index belongs.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

AttributesToGet

Important
This is a legacy parameter, for backward compatibility. New applications should use ProjectionExpression instead. Do not combine legacy parameters and expression parameters in a single API call; otherwise, DynamoDB will return a ValidationException exception. This parameter allows you to retrieve attributes of type List or Map; however, it cannot retrieve individual elements within a List or a Map.

The names of one or more attributes to retrieve. If no attribute names are provided, then all attributes will be returned. If any of the requested attributes are not found, they will not appear in the result.

Note that *AttributesToGet* has no effect on provisioned throughput consumption. DynamoDB determines capacity units consumed based on item size, not on the amount of data that is returned to an application.

Type: array of Strings

Length constraints: Minimum of 1 item(s) in the list.

Required: No

ConditionalOperator

Important
This is a legacy parameter, for backward compatibility. New applications should use FilterExpression instead. Do not combine legacy parameters and expression parameters in a single API call; otherwise, DynamoDB will return a ValidationException exception.

A logical operator to apply to the conditions in a *ScanFilter* map:

- `AND` - If all of the conditions evaluate to true, then the entire map evaluates to true.
- `OR` - If at least one of the conditions evaluate to true, then the entire map evaluates to true.

If you omit *ConditionalOperator*, then `AND` is the default.

The operation will succeed only if the entire map evaluates to true.

Note
This parameter does not support attributes of type List or Map.

Type: String

Valid Values: `AND` | `OR`

Required: No

ConsistentRead

A Boolean value that determines the read consistency model during the scan:

- If *ConsistentRead* is `false`, then the data returned from *Scan* might not contain the results from other recently completed write operations (PutItem, UpdateItem or DeleteItem).
- If *ConsistentRead* is `true`, then all of the write operations that completed before the *Scan* began are guaranteed to be contained in the *Scan* response.

The default setting for *ConsistentRead* is `false`.

The *ConsistentRead* parameter is not supported on global secondary indexes. If you scan a global secondary index with *ConsistentRead* set to true, you will receive a *ValidationException*.

Type: Boolean

Required: No

ExclusiveStartKey

The primary key of the first item that this operation will evaluate. Use the value that was returned for *LastEvaluatedKey* in the previous operation.

The data type for *ExclusiveStartKey* must be String, Number or Binary. No set data types are allowed.

In a parallel scan, a *Scan* request that includes *ExclusiveStartKey* must specify the same segment whose previous *Scan* returned the corresponding value of *LastEvaluatedKey*.

Type: String to AttributeValue (p. 130) object map

Required: No

ExpressionAttributeNames

One or more substitution tokens for attribute names in an expression. The following are some use cases for using *ExpressionAttributeNames*:

- To access an attribute whose name conflicts with a DynamoDB reserved word.
- To create a placeholder for repeating occurrences of an attribute name in an expression.
- To prevent special characters in an attribute name from being misinterpreted in an expression.

Use the # character in an expression to dereference an attribute name. For example, consider the following attribute name:

- `Percentile`

The name of this attribute conflicts with a reserved word, so it cannot be used directly in an expression. (For the complete list of reserved words, see Reserved Words in the *Amazon DynamoDB Developer Guide*). To work around this, you could specify the following for *ExpressionAttributeNames*:

- `{"#P":"Percentile"}`

You could then use this substitution in an expression, as in this example:

- `#P = :val`

Note
Tokens that begin with the : character are expression attribute values, which are placeholders for the actual value at runtime.

For more information on expression attribute names, see Accessing Item Attributes in the *Amazon DynamoDB Developer Guide*.

Type: String to String map

Required: No

ExpressionAttributeValues

One or more values that can be substituted in an expression.

Use the **:** (colon) character in an expression to dereference an attribute value. For example, suppose that you wanted to check whether the value of the *ProductStatus* attribute was one of the following:

```
Available | Backordered | Discontinued
```

You would first need to specify *ExpressionAttributeValues* as follows:

```
{ ":avail":{"S":"Available"}, ":back":{"S":"Backordered"},
":disc":{"S":"Discontinued"} }
```

You could then use these values in an expression, such as this:

```
ProductStatus IN (:avail, :back, :disc)
```

For more information on expression attribute values, see Specifying Conditions in the *Amazon DynamoDB Developer Guide*.

Type: String to AttributeValue (p. 130) object map

Required: No

FilterExpression

A string that contains conditions that DynamoDB applies after the *Scan* operation, but before the data is returned to you. Items that do not satisfy the *FilterExpression* criteria are not returned.

Note
A FilterExpression is applied after the items have already been read; the process of filtering does not consume any additional read capacity units.

For more information, see Filter Expressions in the *Amazon DynamoDB Developer Guide*.

Note
FilterExpression replaces the legacy ScanFilter and ConditionalOperator parameters.

Type: String

Required: No

IndexName

The name of a secondary index to scan. This index can be any local secondary index or global secondary index. Note that if you use the `IndexName` parameter, you must also provide `TableName`.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: No

Limit

The maximum number of items to evaluate (not necessarily the number of matching items). If DynamoDB processes the number of items up to the limit while processing the results, it stops the operation and returns the matching values up to that point, and a key in *LastEvaluatedKey* to apply in a subsequent operation, so that you can pick up where you left off. Also, if the processed data set size exceeds 1 MB before DynamoDB reaches this limit, it stops the operation and returns the matching values up to the limit, and a key in *LastEvaluatedKey* to apply in a subsequent operation to continue the operation. For more information, see Query and Scan in the *Amazon DynamoDB Developer Guide*.

Type: Number

Valid range: Minimum value of 1.

Required: No

ProjectionExpression

A string that identifies one or more attributes to retrieve from the specified table or index. These attributes can include scalars, sets, or elements of a JSON document. The attributes in the expression must be separated by commas.

If no attribute names are specified, then all attributes will be returned. If any of the requested attributes are not found, they will not appear in the result.

For more information, see Accessing Item Attributes in the *Amazon DynamoDB Developer Guide*.

> **Note**
> ProjectionExpression replaces the legacy AttributesToGet parameter.

Type: String

Required: No

ReturnConsumedCapacity

Determines the level of detail about provisioned throughput consumption that is returned in the response:

- *INDEXES* - The response includes the aggregate *ConsumedCapacity* for the operation, together with *ConsumedCapacity* for each table and secondary index that was accessed.

  Note that some operations, such as *GetItem* and *BatchGetItem*, do not access any indexes at all. In these cases, specifying *INDEXES* will only return *ConsumedCapacity* information for table(s).
- *TOTAL* - The response includes only the aggregate *ConsumedCapacity* for the operation.
- *NONE* - No *ConsumedCapacity* details are included in the response.

Type: String

Valid Values: `INDEXES | TOTAL | NONE`

Required: No

ScanFilter

Important
This is a legacy parameter, for backward compatibility. New applications should use FilterExpression instead. Do not combine legacy parameters and expression parameters in a single API call; otherwise, DynamoDB will return a ValidationException exception.

A condition that evaluates the scan results and returns only the desired values.

Note
This parameter does not support attributes of type List or Map.

If you specify more than one condition in the *ScanFilter* map, then by default all of the conditions must evaluate to true. In other words, the conditions are ANDed together. (You can use the *ConditionalOperator* parameter to OR the conditions instead. If you do this, then at least one of the conditions must evaluate to true, rather than all of them.)

Each *ScanFilter* element consists of an attribute name to compare, along with the following:

- *AttributeValueList* - One or more values to evaluate against the supplied attribute. The number of values in the list depends on the operator specified in *ComparisonOperator* .

  For type Number, value comparisons are numeric.

  String value comparisons for greater than, equals, or less than are based on ASCII character code values. For example, a is greater than A, and a is greater than B. For a list of code values, see http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters.

  For Binary, DynamoDB treats each byte of the binary data as unsigned when it compares binary values.

  For information on specifying data types in JSON, see JSON Data Format in the *Amazon DynamoDB Developer Guide*.
- *ComparisonOperator* - A comparator for evaluating attributes. For example, equals, greater than, less than, etc.

  The following comparison operators are available:

  ```
  EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS |
  BEGINS_WITH | IN | BETWEEN
  ```

  For complete descriptions of all comparison operators, see Condition.

Type: String to Condition (p. 134) object map

Required: No

Segment

For a parallel *Scan* request, *Segment* identifies an individual segment to be scanned by an application worker.

Segment IDs are zero-based, so the first segment is always 0. For example, if you want to use four application threads to scan a table or an index, then the first thread specifies a *Segment* value of 0, the second thread specifies 1, and so on.

The value of *LastEvaluatedKey* returned from a parallel *Scan* request must be used as *ExclusiveStartKey* with the same segment ID in a subsequent *Scan* operation.

The value for *Segment* must be greater than or equal to 0, and less than the value provided for *TotalSegments*.

If you provide *Segment*, you must also provide *TotalSegments*.

Type: Number

Valid range: Minimum value of 0. Maximum value of 999999.

Required: No

Select

The attributes to be returned in the result. You can retrieve all item attributes, specific item attributes, or the count of matching items.

- `ALL_ATTRIBUTES` - Returns all of the item attributes.
- `COUNT` - Returns the number of matching items, rather than the matching items themselves.
- `SPECIFIC_ATTRIBUTES` - Returns only the attributes listed in *AttributesToGet*. This return value is equivalent to specifying *AttributesToGet* without specifying any value for *Select*.

If neither *Select* nor *AttributesToGet* are specified, DynamoDB defaults to `ALL_ATTRIBUTES`. You cannot use both *AttributesToGet* and *Select* together in a single request, unless the value for *Select* is `SPECIFIC_ATTRIBUTES`. (This usage is equivalent to specifying *AttributesToGet* without any value for *Select*.)

Type: String

Valid Values: `ALL_ATTRIBUTES | ALL_PROJECTED_ATTRIBUTES | SPECIFIC_ATTRIBUTES | COUNT`

Required: No

TotalSegments

For a parallel *Scan* request, *TotalSegments* represents the total number of segments into which the *Scan* operation will be divided. The value of *TotalSegments* corresponds to the number of application workers that will perform the parallel scan. For example, if you want to use four application threads to scan a table or an index, specify a *TotalSegments* value of 4.

The value for *TotalSegments* must be greater than or equal to 1, and less than or equal to 1000000. If you specify a *TotalSegments* value of 1, the *Scan* operation will be sequential rather than parallel.

If you specify *TotalSegments*, you must also specify *Segment*.

Type: Number

Valid range: Minimum value of 1. Maximum value of 1000000.

Required: No

Response Syntax

```
{
    "ConsumedCapacity": {
        "CapacityUnits": number,
        "GlobalSecondaryIndexes":
            {
                "string" :
                    {
                        "CapacityUnits": number
                    }
            },
        "LocalSecondaryIndexes":
            {
```

```
            "string" :
                {
                    "CapacityUnits": number
                }
        },
    "Table": {
        "CapacityUnits": number
    },
    "TableName": "string"
},
"Count": number,
"Items": [

        {
            "string" :
                {
                    "B": blob,
                    "BOOL": boolean,
                    "BS": [
                        blob
                    ],
                    "L": [
                        AttributeValue
                    ],
                    "M":
                        {
                            "string" :
                                AttributeValue
                        },
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
                        "string"
                    ]
                }
        }
],
"LastEvaluatedKey":
    {
        "string" :
            {
                "B": blob,
                "BOOL": boolean,
                "BS": [
                    blob
                ],
                "L": [
                    AttributeValue
                ],
                "M":
                    {
                        "string" :
                            AttributeValue
                    },
```

```
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
                        "string"
                    ]
                }
        },
    "ScannedCount": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ConsumedCapacity
The capacity units consumed by an operation. The data returned includes the total provisioned throughput consumed, along with statistics for the table and any indexes involved in the operation. *ConsumedCapacity* is only returned if the request asked for it. For more information, see Provisioned Throughput in the *Amazon DynamoDB Developer Guide*.

Type: ConsumedCapacity (p. 136) object

Count
The number of items in the response.

If you set *ScanFilter* in the request, then *Count* is the number of items returned after the filter was applied, and *ScannedCount* is the number of matching items before the filter was applied.

If you did not use a filter in the request, then *Count* is the same as *ScannedCount*.

Type: Number

Items
An array of item attributes that match the scan criteria. Each element in this array consists of an attribute name and the value for that attribute.

Type: array of s

LastEvaluatedKey
The primary key of the item where the operation stopped, inclusive of the previous result set. Use this value to start a new operation, excluding this value in the new request.

If *LastEvaluatedKey* is empty, then the "last page" of results has been processed and there is no more data to be retrieved.

If *LastEvaluatedKey* is not empty, it does not necessarily mean that there is more data in the result set. The only way to know when you have reached the end of the result set is when *LastEvaluatedKey* is empty.

Type: String to AttributeValue (p. 130) object map

ScannedCount
The number of items evaluated, before any *ScanFilter* is applied. A high *ScannedCount* value with few, or no, *Count* results indicates an inefficient *Scan* operation. For more information, see Count and ScannedCount in the *Amazon DynamoDB Developer Guide*.

If you did not use a filter in the request, then *ScannedCount* is the same as *Count*.

Type: Number

Errors

For information about the errors that are common to all actions, see Common Errors (p. 159).

InternalServerError
An error occurred on the server side.

HTTP Status Code: 500

ProvisionedThroughputExceededException
Your request rate is too high. The AWS SDKs for DynamoDB automatically retry requests that receive this exception. Your request is eventually successful, unless your retry queue is too large to finish. Reduce the frequency of requests and use exponential backoff. For more information, go to Error Retries and Exponential Backoff in the *Amazon DynamoDB Developer Guide*.

HTTP Status Code: 400

ResourceNotFoundException
The operation tried to access a nonexistent table or index. The resource might not be specified correctly, or its status might not be `ACTIVE`.

HTTP Status Code: 400

Examples

Return All Items

The following example returns all of the items in a table. No scan filter is applied.

Sample Request

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date> X-Amz-Target: DynamoDB_20120810.Scan

{
    "TableName": "Reply",
    "ReturnConsumedCapacity": "TOTAL"
}
```

Sample Response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
 {
    "ConsumedCapacity": {
        "CapacityUnits": 0.5,
        "TableName": "Reply"
    },
    "Count": 4,
    "Items": [
        {
            "PostedBy": {
                "S": "joe@example.com"
            },
            "ReplyDateTime": {
                "S": "20130320115336"
            },
            "Id": {
                "S": "Amazon DynamoDB#How do I update multiple items?"
            },
            "Message": {
                "S": "Have you looked at the BatchWriteItem API?"
            }
        },
        {
            "PostedBy": {
                "S": "fred@example.com"
            },
            "ReplyDateTime": {
                "S": "20130320115342"
            },
            "Id": {
                "S": "Amazon DynamoDB#How do I update multiple items?"
            },
            "Message": {
                "S": "No, I didn't know about that.  Where can I find more in
formation?"
            }
        },
        {
            "PostedBy": {
                "S": "joe@example.com"
            },
            "ReplyDateTime": {
                "S": "20130320115347"
            },
            "Id": {
                "S": "Amazon DynamoDB#How do I update multiple items?"
            },
            "Message": {
                "S": "BatchWriteItem is documented in the Amazon DynamoDB API
```

```
Reference."
            }
        },
        {
            "PostedBy": {
                "S": "fred@example.com"
            },
            "ReplyDateTime": {
                "S": "20130320115352"
            },
            "Id": {
                "S": "Amazon DynamoDB#How do I update multiple items?"
            },
            "Message": {
                "S": "OK, I'll take a look at that.  Thanks!"
            }
        }
    ],
    "ScannedCount": 4
}
```

Use a Filter Expression

The following example returns only those items matching specific criteria.

Sample Request

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date> X-Amz-Target: DynamoDB_20120810.Scan

{
    "TableName": "Reply",
    "FilterExpression": "PostedBy = :val",
    "ExpressionAttributeValues": {":val": {"S": "joe@example.com"}},
    "ReturnConsumedCapacity": "TOTAL"
}
```

Sample Response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
```

```
Date: <Date>
 {
    "ConsumedCapacity": {
        "CapacityUnits": 0.5,
        "TableName": "Reply"
    },
    "Count": 2,
    "Items": [
        {
            "PostedBy": {
                "S": "joe@example.com"
            },
            "ReplyDateTime": {
                "S": "20130320115336"
            },
            "Id": {
                "S": "Amazon DynamoDB#How do I update multiple items?"
            },
            "Message": {
                "S": "Have you looked at the BatchWriteItem API?"
            }
        },
        {
            "PostedBy": {
                "S": "joe@example.com"
            },
            "ReplyDateTime": {
                "S": "20130320115347"
            },
            "Id": {
                "S": "Amazon DynamoDB#How do I update multiple items?"
            },
            "Message": {
                "S": "BatchWriteItem is documented in the Amazon DynamoDB API
Reference."
            }
        }
    ],
    "ScannedCount": 4
}
```

UpdateItem

Edits an existing item's attributes, or adds a new item to the table if it does not already exist. You can put, delete, or add attribute values. You can also perform a conditional update on an existing item (insert a new attribute name-value pair if it doesn't exist, or replace an existing name-value pair if it has certain expected attribute values).

You can also return the item's attribute values in the same *UpdateItem* operation using the *ReturnValues* parameter.

Request Syntax

```
{
    "AttributeUpdates":
        {
            "string" :
                {
                    "Action": "string",
                    "Value": {
                        "B": blob,
                        "BOOL": boolean,
                        "BS": [
                            blob
                        ],
                        "L": [
                            AttributeValue
                        ],
                        "M":
                            {
                                "string" :
                                    AttributeValue
                            },
                        "N": "string",
                        "NS": [
                            "string"
                        ],
                        "NULL": boolean,
                        "S": "string",
                        "SS": [
                            "string"
                        ]
                    }
                }
        },
    "ConditionalOperator": "string",
    "ConditionExpression": "string",
    "Expected":
        {
            "string" :
                {
                    "AttributeValueList": [
                        {
                            "B": blob,
                            "BOOL": boolean,
                            "BS": [
```

```
                        blob
                    ],
                    "L": [
                        AttributeValue
                    ],
                    "M":
                        {
                            "string" :
                                AttributeValue
                        },
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
                        "string"
                    ]
                }
            ],
            "ComparisonOperator": "string",
            "Exists": boolean,
            "Value": {
                "B": blob,
                "BOOL": boolean,
                "BS": [
                    blob
                ],
                "L": [
                    AttributeValue
                ],
                "M":
                    {
                        "string" :
                            AttributeValue
                    },
                "N": "string",
                "NS": [
                    "string"
                ],
                "NULL": boolean,
                "S": "string",
                "SS": [
                    "string"
                ]
            }
        }
    },
"ExpressionAttributeNames":
    {
        "string" :
            "string"
    },
"ExpressionAttributeValues":
    {
        "string" :
            {
```

```
                    "B": blob,
                    "BOOL": boolean,
                    "BS": [
                        blob
                    ],
                    "L": [
                        AttributeValue
                    ],
                    "M":
                        {
                            "string" :
                                AttributeValue
                        },
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
                        "string"
                    ]
                }
        },
    "Key":
        {
            "string" :
                {
                    "B": blob,
                    "BOOL": boolean,
                    "BS": [
                        blob
                    ],
                    "L": [
                        AttributeValue
                    ],
                    "M":
                        {
                            "string" :
                                AttributeValue
                        },
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
                        "string"
                    ]
                }
        },
    "ReturnConsumedCapacity": "string",
    "ReturnItemCollectionMetrics": "string",
    "ReturnValues": "string",
    "TableName": "string",
    "UpdateExpression": "string"
}
```

Request Parameters

The request requires the following data in JSON format.

Note
In the following list, the required parameters are described first.

Key
The primary key of the item to be updated. Each element consists of an attribute name and a value for that attribute.

For the primary key, you must provide all of the attributes. For example, with a hash type primary key, you only need to provide the hash attribute. For a hash-and-range type primary key, you must provide both the hash attribute and the range attribute.

Type: String to AttributeValue (p. 130) object map

Required: Yes

TableName
The name of the table containing the item to update.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

AttributeUpdates

Important
This is a legacy parameter, for backward compatibility. New applications should use UpdateExpression instead. Do not combine legacy parameters and expression parameters in a single API call; otherwise, DynamoDB will return a ValidationException exception. This parameter can be used for modifying top-level attributes; however, it does not support individual list or map elements.

The names of attributes to be modified, the action to perform on each, and the new value for each. If you are updating an attribute that is an index key attribute for any indexes on that table, the attribute type must match the index key type defined in the *AttributesDefinition* of the table description. You can use *UpdateItem* to update any nonkey attributes.

Attribute values cannot be null. String and Binary type attributes must have lengths greater than zero. Set type attributes must not be empty. Requests with empty values will be rejected with a *ValidationException* exception.

Each *AttributeUpdates* element consists of an attribute name to modify, along with the following:

- *Value* - The new value, if applicable, for this attribute.
- *Action* - A value that specifies how to perform the update. This action is only valid for an existing attribute whose data type is Number or is a set; do not use `ADD` for other data types.

  If an item with the specified primary key is found in the table, the following values perform the following actions:

  - `PUT` - Adds the specified attribute to the item. If the attribute already exists, it is replaced by the new value.
  - `DELETE` - Removes the attribute and its value, if no value is specified for `DELETE`. The data type of the specified value must match the existing value's data type.

If a set of values is specified, then those values are subtracted from the old set. For example, if the attribute value was the set `[a,b,c]` and the `DELETE` action specifies `[a,c]`, then the final attribute value is `[b]`. Specifying an empty set is an error.

- `ADD` - Adds the specified value to the item, if the attribute does not already exist. If the attribute does exist, then the behavior of `ADD` depends on the data type of the attribute:
  - If the existing attribute is a number, and if *Value* is also a number, then *Value* is mathematically added to the existing attribute. If *Value* is a negative number, then it is subtracted from the existing attribute.

    Note
    If you use ADD to increment or decrement a number value for an item that doesn't exist before the update, DynamoDB uses 0 as the initial value. Similarly, if you use ADD for an existing item to increment or decrement an attribute value that doesn't exist before the update, DynamoDB uses 0 as the initial value. For example, suppose that the item you want to update doesn't have an attribute named itemcount, but you decide to ADD the number 3 to this attribute anyway. DynamoDB will create the itemcount attribute, set its initial value to 0, and finally add 3 to it. The result will be a new itemcount attribute, with a value of 3.

  - If the existing data type is a set, and if *Value* is also a set, then *Value* is appended to the existing set. For example, if the attribute value is the set `[1,2]`, and the `ADD` action specified `[3]`, then the final attribute value is `[1,2,3]`. An error occurs if an `ADD` action is specified for a set attribute and the attribute type specified does not match the existing set type.

    Both sets must have the same primitive data type. For example, if the existing data type is a set of strings, *Value* must also be a set of strings.

If no item with the specified key is found in the table, the following values perform the following actions:

- `PUT` - Causes DynamoDB to create a new item with the specified primary key, and then adds the attribute.
- `DELETE` - Nothing happens, because attributes cannot be deleted from a nonexistent item. The operation succeeds, but DynamoDB does not create a new item.
- `ADD` - Causes DynamoDB to create an item with the supplied primary key and number (or set of numbers) for the attribute value. The only data types allowed are Number and Number Set.

If you provide any attributes that are part of an index key, then the data types for those attributes must match those of the schema in the table's attribute definition.

Type: String to AttributeValueUpdate (p. 132) object map

Required: No

ConditionalOperator

Important
This is a legacy parameter, for backward compatibility. New applications should use ConditionExpression instead. Do not combine legacy parameters and expression parameters in a single API call; otherwise, DynamoDB will return a ValidationException exception.

A logical operator to apply to the conditions in the *Expected* map:

- `AND` - If all of the conditions evaluate to true, then the entire map evaluates to true.
- `OR` - If at least one of the conditions evaluate to true, then the entire map evaluates to true.

If you omit *ConditionalOperator*, then `AND` is the default.

The operation will succeed only if the entire map evaluates to true.

Note
This parameter does not support attributes of type List or Map.

Type: String

Valid Values: `AND | OR`

Required: No

ConditionExpression

A condition that must be satisfied in order for a conditional update to succeed.

An expression can contain any of the following:

- Functions: `attribute_exists | attribute_not_exists | attribute_type | contains | begins_with | size`

  These function names are case-sensitive.
- Comparison operators: `= | <> | < | > | <= | >= | BETWEEN | IN`
- Logical operators: `AND | OR | NOT`

For more information on condition expressions, see Specifying Conditions in the *Amazon DynamoDB Developer Guide*.

> **Note**
> ConditionExpression replaces the legacy ConditionalOperator and Expected parameters.

Type: String

Required: No

Expected

> **Important**
> This is a legacy parameter, for backward compatibility. New applications should use ConditionExpression instead. Do not combine legacy parameters and expression parameters in a single API call; otherwise, DynamoDB will return a ValidationException exception.

A map of attribute/condition pairs. *Expected* provides a conditional block for the *UpdateItem* operation.

Each element of *Expected* consists of an attribute name, a comparison operator, and one or more values. DynamoDB compares the attribute with the value(s) you supplied, using the comparison operator. For each *Expected* element, the result of the evaluation is either true or false.

If you specify more than one element in the *Expected* map, then by default all of the conditions must evaluate to true. In other words, the conditions are ANDed together. (You can use the *ConditionalOperator* parameter to OR the conditions instead. If you do this, then at least one of the conditions must evaluate to true, rather than all of them.)

If the *Expected* map evaluates to true, then the conditional operation succeeds; otherwise, it fails.

Expected contains the following:

- *AttributeValueList* - One or more values to evaluate against the supplied attribute. The number of values in the list depends on the *ComparisonOperator* being used.

  For type Number, value comparisons are numeric.

  String value comparisons for greater than, equals, or less than are based on ASCII character code values. For example, `a` is greater than `A`, and `a` is greater than `B`. For a list of code values, see http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters.

  For type Binary, DynamoDB treats each byte of the binary data as unsigned when it compares binary values.
- *ComparisonOperator* - A comparator for evaluating attributes in the *AttributeValueList*. When performing the comparison, DynamoDB uses strongly consistent reads.

The following comparison operators are available:

`EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN`

The following are descriptions of each comparison operator.

- `EQ` : Equal. `EQ` is supported for all datatypes, including lists and maps.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, Binary, String Set, Number Set, or Binary Set. If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not equal `{"NS":["6", "2", "1"]}`.

- `NE` : Not equal. `NE` is supported for all datatypes, including lists and maps.

  AttributeValueList can contain only one *AttributeValue* of type String, Number, Binary, String Set, Number Set, or Binary Set. If an item contains an *AttributeValue* of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not equal `{"NS":["6", "2", "1"]}`.

- `LE` : Less than or equal.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `LT` : Less than.

  AttributeValueList can contain only one *AttributeValue* of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `GE` : Greater than or equal.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `GT` : Greater than.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `NOT_NULL` : The attribute exists. `NOT_NULL` is supported for all datatypes, including lists and maps.

  Note
  This operator tests for the existence of an attribute, not its data type. If the data type of attribute "a" is null, and you evaluate it using NOT_NULL, the result is a Boolean true. This result is because the attribute "a" exists; its data type is not relevant to the NOT_NULL comparison operator.

- `NULL` : The attribute does not exist. `NULL` is supported for all datatypes, including lists and maps.

Note
This operator tests for the nonexistence of an attribute, not its data type. If the data type of attribute "a" is null, and you evaluate it using NULL, the result is a Boolean false. This is because the attribute "a" exists; its data type is not relevant to the NULL comparison operator.

- `CONTAINS` : Checks for a subsequence, or value in a set.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If the target attribute of the comparison is of type String, then the operator checks for a substring match. If the target attribute of the comparison is of type Binary, then the operator looks for a subsequence of the target that matches the input. If the target attribute of the comparison is a set ("SS", "NS", or "BS"), then the operator evaluates to true if it finds an exact match with any member of the set.

  CONTAINS is supported for lists: When evaluating "a `CONTAINS` b", "a" can be a list; however, "b" cannot be a set, a map, or a list.

- `NOT_CONTAINS` : Checks for absence of a subsequence, or absence of a value in a set.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If the target attribute of the comparison is a String, then the operator checks for the absence of a substring match. If the target attribute of the comparison is Binary, then the operator checks for the absence of a subsequence of the target that matches the input. If the target attribute of the comparison is a set ("SS", "NS", or "BS"), then the operator evaluates to true if it *does not* find an exact match with any member of the set.

  NOT_CONTAINS is supported for lists: When evaluating "a `NOT CONTAINS` b", "a" can be a list; however, "b" cannot be a set, a map, or a list.

- `BEGINS_WITH` : Checks for a prefix.

  AttributeValueList can contain only one *AttributeValue* of type String or Binary (not a Number or a set type). The target attribute of the comparison must be of type String or Binary (not a Number or a set type).

- `IN` : Checks for matching elements within two sets.

  AttributeValueList can contain one or more *AttributeValue* elements of type String, Number, or Binary (not a set type). These attributes are compared against an existing set type attribute of an item. If any elements of the input set are present in the item attribute, the expression evaluates to true.

- `BETWEEN` : Greater than or equal to the first value, and less than or equal to the second value.

  AttributeValueList must contain two *AttributeValue* elements of the same type, either String, Number, or Binary (not a set type). A target attribute matches if the target value is greater than, or equal to, the first element and less than, or equal to, the second element. If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not compare to `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`

For usage examples of *AttributeValueList* and *ComparisonOperator*, see Legacy Conditional Parameters in the *Amazon DynamoDB Developer Guide*.

For backward compatibility with previous DynamoDB releases, the following parameters can be used instead of *AttributeValueList* and *ComparisonOperator*:

- *Value* - A value for DynamoDB to compare with an attribute.
- *Exists* - A Boolean value that causes DynamoDB to evaluate the value before attempting the conditional operation:

- If *Exists* is `true`, DynamoDB will check to see if that attribute value already exists in the table. If it is found, then the condition evaluates to true; otherwise the condition evaluate to false.
- If *Exists* is `false`, DynamoDB assumes that the attribute value does *not* exist in the table. If in fact the value does not exist, then the assumption is valid and the condition evaluates to true. If the value is found, despite the assumption that it does not exist, the condition evaluates to false.

Note that the default value for *Exists* is `true`.

The *Value* and *Exists* parameters are incompatible with *AttributeValueList* and *ComparisonOperator*. Note that if you use both sets of parameters at once, DynamoDB will return a *ValidationException* exception.

Note
This parameter does not support attributes of type List or Map.

Type: String to ExpectedAttributeValue (p. 139) object map

Required: No

ExpressionAttributeNames

One or more substitution tokens for attribute names in an expression. The following are some use cases for using *ExpressionAttributeNames*:

- To access an attribute whose name conflicts with a DynamoDB reserved word.
- To create a placeholder for repeating occurrences of an attribute name in an expression.
- To prevent special characters in an attribute name from being misinterpreted in an expression.

Use the # character in an expression to dereference an attribute name. For example, consider the following attribute name:

- `Percentile`

The name of this attribute conflicts with a reserved word, so it cannot be used directly in an expression. (For the complete list of reserved words, see Reserved Words in the *Amazon DynamoDB Developer Guide*). To work around this, you could specify the following for *ExpressionAttributeNames*:

- `{"#P":"Percentile"}`

You could then use this substitution in an expression, as in this example:

- `#P = :val`

Note
Tokens that begin with the : character are expression attribute values, which are placeholders for the actual value at runtime.

For more information on expression attribute names, see Accessing Item Attributes in the *Amazon DynamoDB Developer Guide*.

Type: String to String map

Required: No

ExpressionAttributeValues

One or more values that can be substituted in an expression.

Use the **:** (colon) character in an expression to dereference an attribute value. For example, suppose that you wanted to check whether the value of the *ProductStatus* attribute was one of the following:

```
Available | Backordered | Discontinued
```

You would first need to specify *ExpressionAttributeValues* as follows:

```
{ ":avail":{"S":"Available"}, ":back":{"S":"Backordered"},
":disc":{"S":"Discontinued"} }
```

You could then use these values in an expression, such as this:

```
ProductStatus IN (:avail, :back, :disc)
```

For more information on expression attribute values, see Specifying Conditions in the *Amazon DynamoDB Developer Guide*.

Type: String to AttributeValue (p. 130) object map

Required: No

ReturnConsumedCapacity

Determines the level of detail about provisioned throughput consumption that is returned in the response:

- *INDEXES* - The response includes the aggregate *ConsumedCapacity* for the operation, together with *ConsumedCapacity* for each table and secondary index that was accessed.

  Note that some operations, such as *GetItem* and *BatchGetItem*, do not access any indexes at all. In these cases, specifying *INDEXES* will only return *ConsumedCapacity* information for table(s).
- *TOTAL* - The response includes only the aggregate *ConsumedCapacity* for the operation.
- *NONE* - No *ConsumedCapacity* details are included in the response.

Type: String

Valid Values: `INDEXES | TOTAL | NONE`

Required: No

ReturnItemCollectionMetrics

Determines whether item collection metrics are returned. If set to `SIZE`, the response includes statistics about item collections, if any, that were modified during the operation are returned in the response. If set to `NONE` (the default), no statistics are returned.

Type: String

Valid Values: `SIZE | NONE`

Required: No

ReturnValues

Use *ReturnValues* if you want to get the item attributes as they appeared either before or after they were updated. For *UpdateItem*, the valid values are:

- `NONE` - If *ReturnValues* is not specified, or if its value is `NONE`, then nothing is returned. (This setting is the default for *ReturnValues*.)
- `ALL_OLD` - If *UpdateItem* overwrote an attribute name-value pair, then the content of the old item is returned.
- `UPDATED_OLD` - The old versions of only the updated attributes are returned.
- `ALL_NEW` - All of the attributes of the new version of the item are returned.
- `UPDATED_NEW` - The new versions of only the updated attributes are returned.

Type: String

Valid Values: `NONE | ALL_OLD | UPDATED_OLD | ALL_NEW | UPDATED_NEW`

Required: No

UpdateExpression

An expression that defines one or more attributes to be updated, the action to be performed on them, and new value(s) for them.

The following action values are available for *UpdateExpression*.

- `SET` - Adds one or more attributes and values to an item. If any of these attribute already exist, they are replaced by the new values. You can also use `SET` to add or subtract from an attribute that is of type Number. For example: `SET myNum = myNum + :val`

  `SET` supports the following functions:

  - `if_not_exists (path, operand)` - if the item does not contain an attribute at the specified path, then `if_not_exists` evaluates to operand; otherwise, it evaluates to path. You can use this function to avoid overwriting an attribute that may already be present in the item.
  - `list_append (operand, operand)` - evaluates to a list with a new element added to it. You can append the new element to the start or the end of the list by reversing the order of the operands.

  These function names are case-sensitive.
- `REMOVE` - Removes one or more attributes from an item.
- `ADD` - Adds the specified value to the item, if the attribute does not already exist. If the attribute does exist, then the behavior of `ADD` depends on the data type of the attribute:
  - If the existing attribute is a number, and if *Value* is also a number, then *Value* is mathematically added to the existing attribute. If *Value* is a negative number, then it is subtracted from the existing attribute.

    Note
    If you use ADD to increment or decrement a number value for an item that doesn't exist before the update, DynamoDB uses 0 as the initial value. Similarly, if you use ADD for an existing item to increment or decrement an attribute value that doesn't exist before the update, DynamoDB uses 0 as the initial value. For example, suppose that the item you want to update doesn't have an attribute named itemcount, but you decide to ADD the number 3 to this attribute anyway. DynamoDB will create the itemcount attribute, set its initial value to 0, and finally add 3 to it. The result will be a new itemcount attribute in the item, with a value of 3.
  - If the existing data type is a set and if *Value* is also a set, then *Value* is added to the existing set. For example, if the attribute value is the set `[1,2]`, and the `ADD` action specified `[3]`, then the final attribute value is `[1,2,3]`. An error occurs if an `ADD` action is specified for a set attribute and the attribute type specified does not match the existing set type.

    Both sets must have the same primitive data type. For example, if the existing data type is a set of strings, the *Value* must also be a set of strings.

    Important
    The ADD action only supports Number and set data types. In addition, ADD can only be used on top-level attributes, not nested attributes.
- `DELETE` - Deletes an element from a set.

  If a set of values is specified, then those values are subtracted from the old set. For example, if the attribute value was the set `[a,b,c]` and the `DELETE` action specifies `[a,c]`, then the final attribute value is `[b]`. Specifying an empty set is an error.

  Important
  The DELETE action only supports set data types. In addition, DELETE can only be used on top-level attributes, not nested attributes.

You can have many actions in a single expression, such as the following: `SET a=:value1, b=:value2 DELETE :value3, :value4, :value5`

For more information on update expressions, see Modifying Items and Attributes in the *Amazon DynamoDB Developer Guide*.

Note
UpdateExpression replaces the legacy AttributeUpdates parameter.

Type: String

Required: No

Response Syntax

```
{
    "Attributes":
        {
            "string" :
                {
                    "B": blob,
                    "BOOL": boolean,
                    "BS": [
                        blob
                    ],
                    "L": [
                        AttributeValue
                    ],
                    "M":
                        {
                            "string" :
                                AttributeValue
                        },
                    "N": "string",
                    "NS": [
                        "string"
                    ],
                    "NULL": boolean,
                    "S": "string",
                    "SS": [
                        "string"
                    ]
                }
        },
    "ConsumedCapacity": {
        "CapacityUnits": number,
        "GlobalSecondaryIndexes":
            {
                "string" :
                    {
                        "CapacityUnits": number
                    }
            },
        "LocalSecondaryIndexes":
            {
                "string" :
                    {
                        "CapacityUnits": number
                    }
            },
        "Table": {
            "CapacityUnits": number
        },
        "TableName": "string"
```

```
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey":
            {
                "string" :
                    {
                        "B": blob,
                        "BOOL": boolean,
                        "BS": [
                            blob
                        ],
                        "L": [
                            AttributeValue
                        ],
                        "M":
                            {
                                "string" :
                                    AttributeValue
                            },
                        "N": "string",
                        "NS": [
                            "string"
                        ],
                        "NULL": boolean,
                        "S": "string",
                        "SS": [
                            "string"
                        ]
                    }
            },
        "SizeEstimateRangeGB": [
            number
        ]
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Attributes
A map of attribute values as they appeared before the *UpdateItem* operation. This map only appears if *ReturnValues* was specified as something other than `NONE` in the request. Each element represents one attribute.

Type: String to AttributeValue (p. 130) object map

ConsumedCapacity
The capacity units consumed by an operation. The data returned includes the total provisioned throughput consumed, along with statistics for the table and any indexes involved in the operation. *ConsumedCapacity* is only returned if the request asked for it. For more information, see Provisioned Throughput in the *Amazon DynamoDB Developer Guide*.

Type: ConsumedCapacity (p. 136) object

ItemCollectionMetrics
Information about item collections, if any, that were affected by the operation. *ItemCollectionMetrics* is only returned if the request asked for it. If the table does not have any local secondary indexes, this information is not returned in the response.

Type: ItemCollectionMetrics (p. 145) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 159).

ConditionalCheckFailedException
A condition specified in the operation could not be evaluated.

HTTP Status Code: 400

InternalServerError
An error occurred on the server side.

HTTP Status Code: 500

ItemCollectionSizeLimitExceededException
An item collection is too large. This exception is only returned for tables that have one or more local secondary indexes.

HTTP Status Code: 400

ProvisionedThroughputExceededException
Your request rate is too high. The AWS SDKs for DynamoDB automatically retry requests that receive this exception. Your request is eventually successful, unless your retry queue is too large to finish. Reduce the frequency of requests and use exponential backoff. For more information, go to Error Retries and Exponential Backoff in the *Amazon DynamoDB Developer Guide*.

HTTP Status Code: 400

ResourceNotFoundException
The operation tried to access a nonexistent table or index. The resource might not be specified correctly, or its status might not be `ACTIVE`.

HTTP Status Code: 400

Examples

Conditional Update

This example updates the Thread table, changing the LastPostedBy attribute - but only if LastPostedBy is currently "fred@example.com". All of the item's attributes, as they appear after the update, are returned in the response.

Sample Request

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
```

```
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date> X-Amz-Target: DynamoDB_20120810.UpdateItem

{
    "TableName": "Thread",
    "Key": {
        "ForumName": {
            "S": "Amazon DynamoDB"
        },
        "Subject": {
            "S": "Maximum number of items?"
        }
    },
    "UpdateExpression": "set LastPostedBy = :val1",
    "ConditionExpression": "LastPostedBy = :val2",
    "ExpressionAttributeValues": {
        ":val1": {"S": "alice@example.com"},
        ":val2": {"S": "fred@example.com"}
    },
    "ReturnValues": "ALL_NEW"
}
```

Sample Response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
 {
    "Attributes": {
        "LastPostedBy": {
            "S": "alice@example.com"
        },
        "ForumName": {
            "S": "Amazon DynamoDB"
        },
        "LastPostDateTime": {
            "S": "20130320010350"
        },
        "Tags": {
            "SS": ["Update","Multiple Items","HelpMe"]
        },
        "Subject": {
            "S": "Maximum number of items?"
        },
        "Views": {
            "N": "5"
        },
        "Message": {
            "S": "I want to put 10 million data items to an Amazon DynamoDB
```

```
table.  Is there an upper limit?"
        }
    }
}
```

Atomic Counter

The following example increments the Replies attribute in the Thread table whenever someone posts a reply. Because ReturnValues is set to NONE, no output appears in the response payload.

Sample Request

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date> X-Amz-Target: DynamoDB_20120810.UpdateItem

{
    "TableName": "Thread",
    "Key": {
        "ForumName": {
            "S": "Amazon DynamoDB"
        },
        "Subject": {
            "S": "A question about updates"
        }
    },
    "UpdateExpression": "set Replies = Replies + :num",
    "ExpressionAttributeValues": {
        ":num": {"N": "1"}
    },
    "ReturnValues" : "NONE"
}
```

Sample Response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
 {
}
```

UpdateTable

Modifies the provisioned throughput settings, global secondary indexes, or DynamoDB Streams settings for a given table.

You can only perform one of the following operations at once:

- Modify the provisioned throughput settings of the table.
- Enable or disable Streams on the table.
- Remove a global secondary index from the table.
- Create a new global secondary index on the table. Once the index begins backfilling, you can use *UpdateTable* to perform other operations.

UpdateTable is an asynchronous operation; while it is executing, the table status changes from `ACTIVE` to `UPDATING`. While it is `UPDATING`, you cannot issue another *UpdateTable* request. When the table returns to the `ACTIVE` state, the *UpdateTable* operation is complete.

Request Syntax

```
{
    "AttributeDefinitions": [
        {
            "AttributeName": "string",
            "AttributeType": "string"
        }
    ],
    "GlobalSecondaryIndexUpdates": [
        {
            "Create": {
                "IndexName": "string",
                "KeySchema": [
                    {
                        "AttributeName": "string",
                        "KeyType": "string"
                    }
                ],
                "Projection": {
                    "NonKeyAttributes": [
                        "string"
                    ],
                    "ProjectionType": "string"
                },
                "ProvisionedThroughput": {
                    "ReadCapacityUnits": number,
                    "WriteCapacityUnits": number
                }
            },
            "Delete": {
                "IndexName": "string"
            },
            "Update": {
                "IndexName": "string",
                "ProvisionedThroughput": {
```

```
                    "ReadCapacityUnits": number,
                    "WriteCapacityUnits": number
                }
            }
        }
    ],
    "ProvisionedThroughput": {
        "ReadCapacityUnits": number,
        "WriteCapacityUnits": number
    },
    "StreamSpecification": {
        "StreamEnabled": boolean,
        "StreamViewType": "string"
    },
    "TableName": "string"
}
```

Request Parameters

The request requires the following data in JSON format.

Note
In the following list, the required parameters are described first.

TableName
The name of the table to be updated.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

AttributeDefinitions
An array of attributes that describe the key schema for the table and indexes. If you are adding a new global secondary index to the table, *AttributeDefinitions* must include the key element(s) of the new index.

Type: array of AttributeDefinition (p. 130) objects

Required: No

GlobalSecondaryIndexUpdates
An array of one or more global secondary indexes for the table. For each index in the array, you can request one action:

- *Create* - add a new global secondary index to the table.
- *Update* - modify the provisioned throughput settings of an existing global secondary index.
- *Delete* - remove a global secondary index from the table.

For more information, see Managing Global Secondary Indexes in the *Amazon DynamoDB Developer Guide*.

Type: array of GlobalSecondaryIndexUpdate (p. 145) objects

Required: No

ProvisionedThroughput
Represents the provisioned throughput settings for a specified table or index. The settings can be modified using the *UpdateTable* operation.

For current minimum and maximum provisioned throughput values, see Limits in the *Amazon DynamoDB Developer Guide*.

Type: ProvisionedThroughput (p. 151) object

Required: No

StreamSpecification
Represents the DynamoDB Streams configuration for the table.

Note
You will receive a ResourceInUseException if you attempt to enable a stream on a table that already has a stream, or if you attempt to disable a stream on a table which does not have a stream.

Type: StreamSpecification (p. 153) object

Required: No

Response Syntax

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "string",
                "AttributeType": "string"
            }
        ],
        "CreationDateTime": number,
        "GlobalSecondaryIndexes": [
            {
                "Backfilling": boolean,
                "IndexArn": "string",
                "IndexName": "string",
                "IndexSizeBytes": number,
                "IndexStatus": "string",
                "ItemCount": number,
                "KeySchema": [
                    {
                        "AttributeName": "string",
                        "KeyType": "string"
                    }
                ],
                "Projection": {
                    "NonKeyAttributes": [
                        "string"
                    ],
                    "ProjectionType": "string"
                },
                "ProvisionedThroughput": {
                    "LastDecreaseDateTime": number,
                    "LastIncreaseDateTime": number,
```

```
                    "NumberOfDecreasesToday": number,
                    "ReadCapacityUnits": number,
                    "WriteCapacityUnits": number
                }
            }
        ],
        "ItemCount": number,
        "KeySchema": [
            {
                "AttributeName": "string",
                "KeyType": "string"
            }
        ],
        "LatestStreamArn": "string",
        "LatestStreamLabel": "string",
        "LocalSecondaryIndexes": [
            {
                "IndexArn": "string",
                "IndexName": "string",
                "IndexSizeBytes": number,
                "ItemCount": number,
                "KeySchema": [
                    {
                        "AttributeName": "string",
                        "KeyType": "string"
                    }
                ],
                "Projection": {
                    "NonKeyAttributes": [
                        "string"
                    ],
                    "ProjectionType": "string"
                }
            }
        ],
        "ProvisionedThroughput": {
            "LastDecreaseDateTime": number,
            "LastIncreaseDateTime": number,
            "NumberOfDecreasesToday": number,
            "ReadCapacityUnits": number,
            "WriteCapacityUnits": number
        },
        "StreamSpecification": {
            "StreamEnabled": boolean,
            "StreamViewType": "string"
        },
        "TableArn": "string",
        "TableName": "string",
        "TableSizeBytes": number,
        "TableStatus": "string"
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TableDescription
Represents the properties of a table.

Type: TableDescription (p. 153) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 159).

InternalServerError
An error occurred on the server side.

HTTP Status Code: 500

LimitExceededException
The number of concurrent table requests (cumulative number of tables in the `CREATING`, `DELETING` or `UPDATING` state) exceeds the maximum allowed of 10.

Also, for tables with secondary indexes, only one of those tables can be in the `CREATING` state at any point in time. Do not attempt to create more than one such table simultaneously.

The total limit of tables in the `ACTIVE` state is 250.

HTTP Status Code: 400

ResourceInUseException
The operation conflicts with the resource's availability. For example, you attempted to recreate an existing table, or tried to delete a table currently in the `CREATING` state.

HTTP Status Code: 400

ResourceNotFoundException
The operation tried to access a nonexistent table or index. The resource might not be specified correctly, or its status might not be `ACTIVE`.

HTTP Status Code: 400

Examples

Modify Provisioned Write Throughput

This example changes both the provisioned read and write throughput of the Thread table to 10 capacity units.

Sample Request

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
```

```
X-Amz-Date: <Date> X-Amz-Target: DynamoDB_20120810.UpdateTable

{
    "TableName": "Thread",
    "ProvisionedThroughput": {
        "ReadCapacityUnits": 10,
        "WriteCapacityUnits": 10
    }
}
```

Sample Response

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
 {
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/Thread",
        "AttributeDefinitions": [
            {
                "AttributeName": "ForumName",
                "AttributeType": "S"
            },
            {
                "AttributeName": "LastPostDateTime",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Subject",
                "AttributeType": "S"
            }
        ],
        "CreationDateTime": 1.363801528686E9,
        "ItemCount": 0,
        "KeySchema": [
            {
                "AttributeName": "ForumName",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "Subject",
                "KeyType": "RANGE"
            }
        ],
        "LocalSecondaryIndexes": [
            {
                "IndexName": "LastPostIndex",
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "KeySchema": [
                    {
```

```
                        "AttributeName": "ForumName",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "LastPostDateTime",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "KEYS_ONLY"
                }
            }
        ],
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": 1.363801701282E9,
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableName": "Thread",
        "TableSizeBytes": 0,
        "TableStatus": "UPDATING"
    }
}
```

Data Types

The Amazon DynamoDB API contains several data types that various actions use. This section describes each data type in detail.

> **Note**
> The order of each element in the response is not guaranteed. Applications should not assume a particular order.

The following data types are supported:

- AttributeDefinition (p. 130)
- AttributeValue (p. 130)
- AttributeValueUpdate (p. 132)
- Capacity (p. 133)
- Condition (p. 134)
- ConsumedCapacity (p. 136)
- CreateGlobalSecondaryIndexAction (p. 137)
- DeleteGlobalSecondaryIndexAction (p. 138)
- DeleteRequest (p. 138)
- ExpectedAttributeValue (p. 139)
- GlobalSecondaryIndex (p. 142)
- GlobalSecondaryIndexDescription (p. 143)
- GlobalSecondaryIndexUpdate (p. 145)
- ItemCollectionMetrics (p. 145)
- KeysAndAttributes (p. 146)
- KeySchemaElement (p. 148)
- LocalSecondaryIndex (p. 148)
- LocalSecondaryIndexDescription (p. 149)
- Projection (p. 150)
- ProvisionedThroughput (p. 151)
- ProvisionedThroughputDescription (p. 151)
- PutRequest (p. 152)
- StreamSpecification (p. 153)
- TableDescription (p. 153)

- UpdateGlobalSecondaryIndexAction (p. 157)
- WriteRequest (p. 157)

AttributeDefinition

Description

Represents an attribute for describing the key schema for the table and indexes.

Contents

Note
In the following list, the required parameters are described first.

AttributeName
A name for the attribute.

Type: String

Length constraints: Minimum length of 1. Maximum length of 255.

Required: Yes

AttributeType
The data type for the attribute.

Type: String

Valid Values: `S` | `N` | `B`

Required: Yes

AttributeValue

Description

Represents the data for an attribute. You can set one, and only one, of the elements.

Each attribute in an item is a name-value pair. An attribute can be single-valued or multi-valued set. For example, a book item can have title and authors attributes. Each book has one title but can have many authors. The multi-valued attribute is a set; duplicate values are not allowed.

Contents

Note
In the following list, the required parameters are described first.

B
A Binary data type.

Type: Blob

Required: No

BOOL
A Boolean data type.

Type: Boolean

Required: No

BS
A Binary Set data type.

Type: array of Blobs

Required: No

L
A List of attribute values.

Type: array of AttributeValue (p. 130) objects

Required: No

M
A Map of attribute values.

Type: String to AttributeValue (p. 130) object map

Required: No

N
A Number data type.

Type: String

Required: No

NS
A Number Set data type.

Type: array of Strings

Required: No

NULL
A Null data type.

Type: Boolean

Required: No

S
A String data type.

Type: String

Required: No

SS
A String Set data type.

Type: array of Strings

Required: No

AttributeValueUpdate

Description

For the *UpdateItem* operation, represents the attributes to be modified, the action to perform on each, and the new value for each.

> **Note**
> You cannot use UpdateItem to update any primary key attributes. Instead, you will need to delete the item, and then use PutItem to create a new item with new attributes.

Attribute values cannot be null; string and binary type attributes must have lengths greater than zero; and set type attributes must not be empty. Requests with empty values will be rejected with a *ValidationException* exception.

Contents

> **Note**
> In the following list, the required parameters are described first.

Action

Specifies how to perform the update. Valid values are `PUT` (default), `DELETE`, and `ADD`. The behavior depends on whether the specified primary key already exists in the table.

If an item with the specified Key is found in the table:

- `PUT` - Adds the specified attribute to the item. If the attribute already exists, it is replaced by the new value.
- `DELETE` - If no value is specified, the attribute and its value are removed from the item. The data type of the specified value must match the existing value's data type.

  If a *set* of values is specified, then those values are subtracted from the old set. For example, if the attribute value was the set `[a,b,c]` and the *DELETE* action specified `[a,c]`, then the final attribute value would be `[b]`. Specifying an empty set is an error.
- `ADD` - If the attribute does not already exist, then the attribute and its values are added to the item. If the attribute does exist, then the behavior of `ADD` depends on the data type of the attribute:
  - If the existing attribute is a number, and if *Value* is also a number, then the *Value* is mathematically added to the existing attribute. If *Value* is a negative number, then it is subtracted from the existing attribute.

    > **Note**
    > If you use ADD to increment or decrement a number value for an item that doesn't exist before the update, DynamoDB uses 0 as the initial value. In addition, if you use ADD to update an existing item, and intend to increment or decrement an attribute value which does not yet exist, DynamoDB uses 0 as the initial value. For example, suppose that the item you want to update does not yet have an attribute named itemcount, but you decide to ADD the number 3 to this attribute anyway, even though it currently does not exist. DynamoDB will create the itemcount attribute, set its initial value to 0, and finally add 3 to it. The result will be a new itemcount attribute in the item, with a value of 3.
  - If the existing data type is a set, and if the *Value* is also a set, then the *Value* is added to the existing set. (This is a *set* operation, not mathematical addition.) For example, if the attribute value was the set `[1,2]`, and the `ADD` action specified `[3]`, then the final attribute value would be `[1,2,3]`. An error occurs if an Add action is specified for a set attribute and the attribute type specified does not match the existing set type.

Both sets must have the same primitive data type. For example, if the existing data type is a set of strings, the *Value* must also be a set of strings. The same holds true for number sets and binary sets.

This action is only valid for an existing attribute whose data type is number or is a set. Do not use `ADD` for any other data types.

If no item with the specified Key is found:

- `PUT` - DynamoDB creates a new item with the specified primary key, and then adds the attribute.
- `DELETE` - Nothing happens; there is no attribute to delete.
- `ADD` - DynamoDB creates an item with the supplied primary key and number (or set of numbers) for the attribute value. The only data types allowed are number and number set; no other data types can be specified.

Type: String

Valid Values: `ADD` | `PUT` | `DELETE`

Required: No

Value

Represents the data for an attribute. You can set one, and only one, of the elements.

Each attribute in an item is a name-value pair. An attribute can be single-valued or multi-valued set. For example, a book item can have title and authors attributes. Each book has one title but can have many authors. The multi-valued attribute is a set; duplicate values are not allowed.

Type: AttributeValue (p. 130) object

Required: No

Capacity

Description

Represents the amount of provisioned throughput capacity consumed on a table or an index.

Contents

Note

In the following list, the required parameters are described first.

CapacityUnits

The total number of capacity units consumed on a table or an index.

Type: Double

Required: No

Condition

Description

Represents the selection criteria for a *Query* or *Scan* operation:

- For a *Query* operation, *Condition* is used for specifying the *KeyConditions* to use when querying a table or an index. For *KeyConditions*, only the following comparison operators are supported:

  `EQ | LE | LT | GE | GT | BEGINS_WITH | BETWEEN`

  Condition is also used in a *QueryFilter*, which evaluates the query results and returns only the desired values.
- For a *Scan* operation, *Condition* is used in a *ScanFilter*, which evaluates the scan results and returns only the desired values.

Contents

Note
In the following list, the required parameters are described first.

ComparisonOperator
A comparator for evaluating attributes. For example, equals, greater than, less than, etc.

The following comparison operators are available:

`EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN`

The following are descriptions of each comparison operator.

- `EQ` : Equal. `EQ` is supported for all datatypes, including lists and maps.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, Binary, String Set, Number Set, or Binary Set. If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not equal `{"NS":["6", "2", "1"]}`.

- `NE` : Not equal. `NE` is supported for all datatypes, including lists and maps.

  AttributeValueList can contain only one *AttributeValue* of type String, Number, Binary, String Set, Number Set, or Binary Set. If an item contains an *AttributeValue* of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not equal `{"NS":["6", "2", "1"]}`.

- `LE` : Less than or equal.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `LT` : Less than.

  AttributeValueList can contain only one *AttributeValue* of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the

request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `GE` : Greater than or equal.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `GT` : Greater than.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `NOT_NULL` : The attribute exists. `NOT_NULL` is supported for all datatypes, including lists and maps.

  Note
  This operator tests for the existence of an attribute, not its data type. If the data type of attribute "a" is null, and you evaluate it using NOT_NULL, the result is a Boolean true. This result is because the attribute "a" exists; its data type is not relevant to the NOT_NULL comparison operator.

- `NULL` : The attribute does not exist. `NULL` is supported for all datatypes, including lists and maps.

  Note
  This operator tests for the nonexistence of an attribute, not its data type. If the data type of attribute "a" is null, and you evaluate it using NULL, the result is a Boolean false. This is because the attribute "a" exists; its data type is not relevant to the NULL comparison operator.

- `CONTAINS` : Checks for a subsequence, or value in a set.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If the target attribute of the comparison is of type String, then the operator checks for a substring match. If the target attribute of the comparison is of type Binary, then the operator looks for a subsequence of the target that matches the input. If the target attribute of the comparison is a set ("SS", "NS", or "BS"), then the operator evaluates to true if it finds an exact match with any member of the set.

  CONTAINS is supported for lists: When evaluating "a `CONTAINS` b", "a" can be a list; however, "b" cannot be a set, a map, or a list.

- `NOT_CONTAINS` : Checks for absence of a subsequence, or absence of a value in a set.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If the target attribute of the comparison is a String, then the operator checks for the absence of a substring match. If the target attribute of the comparison is Binary, then the operator checks for the absence of a subsequence of the target that matches the input. If the target attribute of the comparison is a set ("SS", "NS", or "BS"), then the operator evaluates to true if it *does not* find an exact match with any member of the set.

  NOT_CONTAINS is supported for lists: When evaluating "a `NOT CONTAINS` b", "a" can be a list; however, "b" cannot be a set, a map, or a list.

- `BEGINS_WITH` : Checks for a prefix.

  AttributeValueList can contain only one *AttributeValue* of type String or Binary (not a Number or a set type). The target attribute of the comparison must be of type String or Binary (not a Number or a set type).

- `IN` : Checks for matching elements within two sets.

  AttributeValueList can contain one or more *AttributeValue* elements of type String, Number, or Binary (not a set type). These attributes are compared against an existing set type attribute of an item. If any elements of the input set are present in the item attribute, the expression evaluates to true.
- `BETWEEN` : Greater than or equal to the first value, and less than or equal to the second value.

  AttributeValueList must contain two *AttributeValue* elements of the same type, either String, Number, or Binary (not a set type). A target attribute matches if the target value is greater than, or equal to, the first element and less than, or equal to, the second element. If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not compare to `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`

For usage examples of *AttributeValueList* and *ComparisonOperator*, see Legacy Conditional Parameters in the *Amazon DynamoDB Developer Guide*.

Type: String

Valid Values: `EQ | NE | IN | LE | LT | GE | GT | BETWEEN | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH`

Required: Yes

AttributeValueList

One or more values to evaluate against the supplied attribute. The number of values in the list depends on the *ComparisonOperator* being used.

For type Number, value comparisons are numeric.

String value comparisons for greater than, equals, or less than are based on ASCII character code values. For example, `a` is greater than `A`, and `a` is greater than `B`. For a list of code values, see http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters.

For Binary, DynamoDB treats each byte of the binary data as unsigned when it compares binary values.

Type: array of AttributeValue (p. 130) objects

Required: No

ConsumedCapacity

Description

The capacity units consumed by an operation. The data returned includes the total provisioned throughput consumed, along with statistics for the table and any indexes involved in the operation. *ConsumedCapacity* is only returned if the request asked for it. For more information, see Provisioned Throughput in the *Amazon DynamoDB Developer Guide*.

Contents

Note

In the following list, the required parameters are described first.

CapacityUnits
The total number of capacity units consumed by the operation.

Type: Double

Required: No

GlobalSecondaryIndexes
The amount of throughput consumed on each global index affected by the operation.

Type: String to Capacity (p. 133) object map

Required: No

LocalSecondaryIndexes
The amount of throughput consumed on each local index affected by the operation.

Type: String to Capacity (p. 133) object map

Required: No

Table
The amount of throughput consumed on the table affected by the operation.

Type: Capacity (p. 133) object

Required: No

TableName
The name of the table that was affected by the operation.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: No

CreateGlobalSecondaryIndexAction

Description

Represents a new global secondary index to be added to an existing table.

Contents

Note
In the following list, the required parameters are described first.

IndexName
The name of the global secondary index to be created.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

KeySchema
The key schema for the global secondary index.

Type: array of KeySchemaElement (p. 148) objects

Length constraints: Minimum of 1 item(s) in the list. Maximum of 2 item(s) in the list.

Required: Yes

Projection
Represents attributes that are copied (projected) from the table into an index. These are in addition to the primary key attributes and index key attributes, which are automatically projected.

Type: Projection (p. 150) object

Required: Yes

ProvisionedThroughput
Represents the provisioned throughput settings for a specified table or index. The settings can be modified using the *UpdateTable* operation.

For current minimum and maximum provisioned throughput values, see Limits in the *Amazon DynamoDB Developer Guide*.

Type: ProvisionedThroughput (p. 151) object

Required: Yes

DeleteGlobalSecondaryIndexAction

Description

Represents a global secondary index to be deleted from an existing table.

Contents

Note
In the following list, the required parameters are described first.

IndexName
The name of the global secondary index to be deleted.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

DeleteRequest

Description

Represents a request to perform a *DeleteItem* operation on an item.

Contents

Note
In the following list, the required parameters are described first.

Key
A map of attribute name to attribute values, representing the primary key of the item to delete. All of the table's primary key attributes must be specified, and their data types must match those of the table's key schema.

Type: String to AttributeValue (p. 130) object map

Required: Yes

ExpectedAttributeValue

Description

Represents a condition to be compared with an attribute value. This condition can be used with *DeleteItem*, *PutItem* or *UpdateItem* operations; if the comparison evaluates to true, the operation succeeds; if not, the operation fails. You can use *ExpectedAttributeValue* in one of two different ways:

- Use *AttributeValueList* to specify one or more values to compare against an attribute. Use *ComparisonOperator* to specify how you want to perform the comparison. If the comparison evaluates to true, then the conditional operation succeeds.
- Use *Value* to specify a value that DynamoDB will compare against an attribute. If the values match, then *ExpectedAttributeValue* evaluates to true and the conditional operation succeeds. Optionally, you can also set *Exists* to false, indicating that you *do not* expect to find the attribute value in the table. In this case, the conditional operation succeeds only if the comparison evaluates to false.

Value and *Exists* are incompatible with *AttributeValueList* and *ComparisonOperator*. Note that if you use both sets of parameters at once, DynamoDB will return a *ValidationException* exception.

Contents

Note
In the following list, the required parameters are described first.

AttributeValueList
One or more values to evaluate against the supplied attribute. The number of values in the list depends on the *ComparisonOperator* being used.

For type Number, value comparisons are numeric.

String value comparisons for greater than, equals, or less than are based on ASCII character code values. For example, a is greater than A, and a is greater than B. For a list of code values, see http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters.

For Binary, DynamoDB treats each byte of the binary data as unsigned when it compares binary values.

For information on specifying data types in JSON, see JSON Data Format in the *Amazon DynamoDB Developer Guide*.

Type: array of AttributeValue (p. 130) objects

Required: No

ComparisonOperator

A comparator for evaluating attributes in the *AttributeValueList*. For example, equals, greater than, less than, etc.

The following comparison operators are available:

`EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN`

The following are descriptions of each comparison operator.

- `EQ` : Equal. `EQ` is supported for all datatypes, including lists and maps.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, Binary, String Set, Number Set, or Binary Set. If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not equal `{"NS":["6", "2", "1"]}`.

- `NE` : Not equal. `NE` is supported for all datatypes, including lists and maps.

  AttributeValueList can contain only one *AttributeValue* of type String, Number, Binary, String Set, Number Set, or Binary Set. If an item contains an *AttributeValue* of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not equal `{"NS":["6", "2", "1"]}`.

- `LE` : Less than or equal.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `LT` : Less than.

  AttributeValueList can contain only one *AttributeValue* of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `GE` : Greater than or equal.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `GT` : Greater than.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not equal `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`.

- `NOT_NULL` : The attribute exists. `NOT_NULL` is supported for all datatypes, including lists and maps.

  Note

  This operator tests for the existence of an attribute, not its data type. If the data type of attribute "a" is null, and you evaluate it using NOT_NULL, the result is a Boolean true.

This result is because the attribute "a" exists; its data type is not relevant to the NOT_NULL comparison operator.

- `NULL` : The attribute does not exist. `NULL` is supported for all datatypes, including lists and maps.

  Note
  This operator tests for the nonexistence of an attribute, not its data type. If the data type of attribute "a" is null, and you evaluate it using NULL, the result is a Boolean false. This is because the attribute "a" exists; its data type is not relevant to the NULL comparison operator.

- `CONTAINS` : Checks for a subsequence, or value in a set.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If the target attribute of the comparison is of type String, then the operator checks for a substring match. If the target attribute of the comparison is of type Binary, then the operator looks for a subsequence of the target that matches the input. If the target attribute of the comparison is a set ("`SS`", "`NS`", or "`BS`"), then the operator evaluates to true if it finds an exact match with any member of the set.

  CONTAINS is supported for lists: When evaluating "a `CONTAINS` b", "a" can be a list; however, "b" cannot be a set, a map, or a list.

- `NOT_CONTAINS` : Checks for absence of a subsequence, or absence of a value in a set.

  AttributeValueList can contain only one *AttributeValue* element of type String, Number, or Binary (not a set type). If the target attribute of the comparison is a String, then the operator checks for the absence of a substring match. If the target attribute of the comparison is Binary, then the operator checks for the absence of a subsequence of the target that matches the input. If the target attribute of the comparison is a set ("`SS`", "`NS`", or "`BS`"), then the operator evaluates to true if it *does not* find an exact match with any member of the set.

  NOT_CONTAINS is supported for lists: When evaluating "a `NOT CONTAINS` b", "a" can be a list; however, "b" cannot be a set, a map, or a list.

- `BEGINS_WITH` : Checks for a prefix.

  AttributeValueList can contain only one *AttributeValue* of type String or Binary (not a Number or a set type). The target attribute of the comparison must be of type String or Binary (not a Number or a set type).

- `IN` : Checks for matching elements within two sets.

  AttributeValueList can contain one or more *AttributeValue* elements of type String, Number, or Binary (not a set type). These attributes are compared against an existing set type attribute of an item. If any elements of the input set are present in the item attribute, the expression evaluates to true.

- `BETWEEN` : Greater than or equal to the first value, and less than or equal to the second value.

  AttributeValueList must contain two *AttributeValue* elements of the same type, either String, Number, or Binary (not a set type). A target attribute matches if the target value is greater than, or equal to, the first element and less than, or equal to, the second element. If an item contains an *AttributeValue* element of a different type than the one provided in the request, the value does not match. For example, `{"S":"6"}` does not compare to `{"N":"6"}`. Also, `{"N":"6"}` does not compare to `{"NS":["6", "2", "1"]}`

Type: String

Valid Values: `EQ | NE | IN | LE | LT | GE | GT | BETWEEN | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH`

Required: No

Exists

Causes DynamoDB to evaluate the value before attempting a conditional operation:

- If *Exists* is `true`, DynamoDB will check to see if that attribute value already exists in the table. If it is found, then the operation succeeds. If it is not found, the operation fails with a *ConditionalCheckFailedException*.
- If *Exists* is `false`, DynamoDB assumes that the attribute value does not exist in the table. If in fact the value does not exist, then the assumption is valid and the operation succeeds. If the value is found, despite the assumption that it does not exist, the operation fails with a *ConditionalCheckFailedException*.

The default setting for *Exists* is `true`. If you supply a *Value* all by itself, DynamoDB assumes the attribute exists: You don't have to set *Exists* to `true`, because it is implied.

DynamoDB returns a *ValidationException* if:

- *Exists* is `true` but there is no *Value* to check. (You expect a value to exist, but don't specify what that value is.)
- *Exists* is `false` but you also provide a *Value*. (You cannot expect an attribute to have a value, while also expecting it not to exist.)

Type: Boolean

Required: No

Value

Represents the data for an attribute. You can set one, and only one, of the elements.

Each attribute in an item is a name-value pair. An attribute can be single-valued or multi-valued set. For example, a book item can have title and authors attributes. Each book has one title but can have many authors. The multi-valued attribute is a set; duplicate values are not allowed.

Type: AttributeValue (p. 130) object

Required: No

GlobalSecondaryIndex

Description

Represents the properties of a global secondary index.

Contents

Note

In the following list, the required parameters are described first.

IndexName

The name of the global secondary index. The name must be unique among all other indexes on this table.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

KeySchema
The complete key schema for a global secondary index, which consists of one or more pairs of attribute names and key types (HASH or RANGE).

Type: array of KeySchemaElement (p. 148) objects

Length constraints: Minimum of 1 item(s) in the list. Maximum of 2 item(s) in the list.

Required: Yes

Projection
Represents attributes that are copied (projected) from the table into an index. These are in addition to the primary key attributes and index key attributes, which are automatically projected.

Type: Projection (p. 150) object

Required: Yes

ProvisionedThroughput
Represents the provisioned throughput settings for a specified table or index. The settings can be modified using the *UpdateTable* operation.

For current minimum and maximum provisioned throughput values, see Limits in the *Amazon DynamoDB Developer Guide*.

Type: ProvisionedThroughput (p. 151) object

Required: Yes

GlobalSecondaryIndexDescription

Description

Represents the properties of a global secondary index.

Contents

Note
In the following list, the required parameters are described first.

Backfilling
Indicates whether the index is currently backfilling. *Backfilling* is the process of reading items from the table and determining whether they can be added to the index. (Not all items will qualify: For example, a hash key attribute cannot have any duplicates.) If an item can be added to the index, DynamoDB will do so. After all items have been processed, the backfilling operation is complete and *Backfilling* is false.

Note
For indexes that were created during a CreateTable operation, the Backfilling attribute does not appear in the DescribeTable output.

Type: Boolean

Required: No

IndexArn
The Amazon Resource Name (ARN) that uniquely identifies the index.

Type: String

Required: No

IndexName
The name of the global secondary index.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: No

IndexSizeBytes
The total size of the specified index, in bytes. DynamoDB updates this value approximately every six hours. Recent changes might not be reflected in this value.

Type: Long

Required: No

IndexStatus
The current state of the global secondary index:

- *CREATING* - The index is being created.
- *UPDATING* - The index is being updated.
- *DELETING* - The index is being deleted.
- *ACTIVE* - The index is ready for use.

Type: String

Valid Values: `CREATING | UPDATING | DELETING | ACTIVE`

Required: No

ItemCount
The number of items in the specified index. DynamoDB updates this value approximately every six hours. Recent changes might not be reflected in this value.

Type: Long

Required: No

KeySchema
The complete key schema for the global secondary index, consisting of one or more pairs of attribute names and key types (`HASH` or `RANGE`).

Type: array of KeySchemaElement (p. 148) objects

Length constraints: Minimum of 1 item(s) in the list. Maximum of 2 item(s) in the list.

Required: No

Projection
Represents attributes that are copied (projected) from the table into an index. These are in addition to the primary key attributes and index key attributes, which are automatically projected.

Type: Projection (p. 150) object

Required: No

ProvisionedThroughput
Represents the provisioned throughput settings for the table, consisting of read and write capacity units, along with data about increases and decreases.

Type: ProvisionedThroughputDescription (p. 151) object

Required: No

GlobalSecondaryIndexUpdate

Description

Represents one of the following:

- A new global secondary index to be added to an existing table.
- New provisioned throughput parameters for an existing global secondary index.
- An existing global secondary index to be removed from an existing table.

Contents

Note
In the following list, the required parameters are described first.

Create
The parameters required for creating a global secondary index on an existing table:

- `IndexName`
- `KeySchema`
- `AttributeDefinitions`
- `Projection`
- `ProvisionedThroughput`

Type: CreateGlobalSecondaryIndexAction (p. 137) object

Required: No

Delete
The name of an existing global secondary index to be removed.

Type: DeleteGlobalSecondaryIndexAction (p. 138) object

Required: No

Update
The name of an existing global secondary index, along with new provisioned throughput settings to be applied to that index.

Type: UpdateGlobalSecondaryIndexAction (p. 157) object

Required: No

ItemCollectionMetrics

Description

Information about item collections, if any, that were affected by the operation. *ItemCollectionMetrics* is only returned if the request asked for it. If the table does not have any local secondary indexes, this information is not returned in the response.

Contents

Note
In the following list, the required parameters are described first.

ItemCollectionKey
The hash key value of the item collection. This value is the same as the hash key of the item.

Type: String to AttributeValue (p. 130) object map

Required: No

SizeEstimateRangeGB
An estimate of item collection size, in gigabytes. This value is a two-element array containing a lower bound and an upper bound for the estimate. The estimate includes the size of all the items in the table, plus the size of all attributes projected into all of the local secondary indexes on that table. Use this estimate to measure whether a local secondary index is approaching its size limit.

The estimate is subject to change over time; therefore, do not rely on the precision or accuracy of the estimate.

Type: array of Doubles

Required: No

KeysAndAttributes

Description

Represents a set of primary keys and, for each key, the attributes to retrieve from the table.

For each primary key, you must provide *all* of the key attributes. For example, with a hash type primary key, you only need to provide the hash attribute. For a hash-and-range type primary key, you must provide *both* the hash attribute and the range attribute.

Contents

Note
In the following list, the required parameters are described first.

Keys
The primary key attribute values that define the items and the attributes associated with the items.

Type: array of s

Length constraints: Minimum of 1 item(s) in the list. Maximum of 100 item(s) in the list.

Required: Yes

AttributesToGet
One or more attributes to retrieve from the table or index. If no attribute names are specified then all attributes will be returned. If any of the specified attributes are not found, they will not appear in the result.

Type: array of Strings

Length constraints: Minimum of 1 item(s) in the list.

Required: No

ConsistentRead

The consistency of a read operation. If set to `true`, then a strongly consistent read is used; otherwise, an eventually consistent read is used.

Type: Boolean

Required: No

ExpressionAttributeNames

One or more substitution tokens for attribute names in an expression. The following are some use cases for using *ExpressionAttributeNames*:

- To access an attribute whose name conflicts with a DynamoDB reserved word.
- To create a placeholder for repeating occurrences of an attribute name in an expression.
- To prevent special characters in an attribute name from being misinterpreted in an expression.

Use the # character in an expression to dereference an attribute name. For example, consider the following attribute name:

- `Percentile`

The name of this attribute conflicts with a reserved word, so it cannot be used directly in an expression. (For the complete list of reserved words, see Reserved Words in the *Amazon DynamoDB Developer Guide*). To work around this, you could specify the following for *ExpressionAttributeNames*:

- `{"#P":"Percentile"}`

You could then use this substitution in an expression, as in this example:

- `#P = :val`

> **Note**
> Tokens that begin with the : character are expression attribute values, which are placeholders for the actual value at runtime.

For more information on expression attribute names, see Accessing Item Attributes in the *Amazon DynamoDB Developer Guide*.

Type: String to String map

Required: No

ProjectionExpression

A string that identifies one or more attributes to retrieve from the table. These attributes can include scalars, sets, or elements of a JSON document. The attributes in the *ProjectionExpression* must be separated by commas.

If no attribute names are specified, then all attributes will be returned. If any of the requested attributes are not found, they will not appear in the result.

For more information, see Accessing Item Attributes in the *Amazon DynamoDB Developer Guide*.

> **Note**
> ProjectionExpression replaces the legacy AttributesToGet parameter.

Type: String

Required: No

KeySchemaElement

Description

Represents *a single element* of a key schema. A key schema specifies the attributes that make up the primary key of a table, or the key attributes of an index.

A *KeySchemaElement* represents exactly one attribute of the primary key. For example, a hash type primary key would be represented by one *KeySchemaElement*. A hash-and-range type primary key would require one *KeySchemaElement* for the hash attribute, and another *KeySchemaElement* for the range attribute.

Contents

Note
In the following list, the required parameters are described first.

AttributeName
The name of a key attribute.

Type: String

Length constraints: Minimum length of 1. Maximum length of 255.

Required: Yes

KeyType
The attribute data, consisting of the data type and the attribute value itself.

Type: String

Valid Values: `HASH | RANGE`

Required: Yes

LocalSecondaryIndex

Description

Represents the properties of a local secondary index.

Contents

Note
In the following list, the required parameters are described first.

IndexName
The name of the local secondary index. The name must be unique among all other indexes on this table.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

KeySchema
The complete key schema for the local secondary index, consisting of one or more pairs of attribute names and key types (`HASH` or `RANGE`).

Type: array of KeySchemaElement (p. 148) objects

Length constraints: Minimum of 1 item(s) in the list. Maximum of 2 item(s) in the list.

Required: Yes

Projection
Represents attributes that are copied (projected) from the table into an index. These are in addition to the primary key attributes and index key attributes, which are automatically projected.

Type: Projection (p. 150) object

Required: Yes

LocalSecondaryIndexDescription

Description

Represents the properties of a local secondary index.

Contents

Note
In the following list, the required parameters are described first.

IndexArn
The Amazon Resource Name (ARN) that uniquely identifies the index.

Type: String

Required: No

IndexName
Represents the name of the local secondary index.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: No

IndexSizeBytes
The total size of the specified index, in bytes. DynamoDB updates this value approximately every six hours. Recent changes might not be reflected in this value.

Type: Long

Required: No

ItemCount
The number of items in the specified index. DynamoDB updates this value approximately every six hours. Recent changes might not be reflected in this value.

Type: Long

Required: No

KeySchema
The complete index key schema, which consists of one or more pairs of attribute names and key types (HASH or RANGE).

Type: array of KeySchemaElement (p. 148) objects

Length constraints: Minimum of 1 item(s) in the list. Maximum of 2 item(s) in the list.

Required: No

Projection
Represents attributes that are copied (projected) from the table into an index. These are in addition to the primary key attributes and index key attributes, which are automatically projected.

Type: Projection (p. 150) object

Required: No

Projection

Description

Represents attributes that are copied (projected) from the table into an index. These are in addition to the primary key attributes and index key attributes, which are automatically projected.

Contents

Note
In the following list, the required parameters are described first.

NonKeyAttributes
Represents the non-key attribute names which will be projected into the index.

For local secondary indexes, the total count of *NonKeyAttributes* summed across all of the local secondary indexes, must not exceed 20. If you project the same attribute into two different indexes, this counts as two distinct attributes when determining the total.

Type: array of Strings

Length constraints: Minimum of 1 item(s) in the list. Maximum of 20 item(s) in the list.

Required: No

ProjectionType
The set of attributes that are projected into the index:

- KEYS_ONLY - Only the index and primary keys are projected into the index.
- INCLUDE - Only the specified table attributes are projected into the index. The list of projected attributes are in *NonKeyAttributes*.
- ALL - All of the table attributes are projected into the index.

Type: String

Valid Values: ALL | KEYS_ONLY | INCLUDE

Required: No

ProvisionedThroughput

Description

Represents the provisioned throughput settings for a specified table or index. The settings can be modified using the *UpdateTable* operation.

For current minimum and maximum provisioned throughput values, see Limits in the *Amazon DynamoDB Developer Guide*.

Contents

Note
In the following list, the required parameters are described first.

ReadCapacityUnits
The maximum number of strongly consistent reads consumed per second before DynamoDB returns a *ThrottlingException*. For more information, see Specifying Read and Write Requirements in the *Amazon DynamoDB Developer Guide*.

Type: Long

Valid range: Minimum value of 1.

Required: Yes

WriteCapacityUnits
The maximum number of writes consumed per second before DynamoDB returns a *ThrottlingException*. For more information, see Specifying Read and Write Requirements in the *Amazon DynamoDB Developer Guide*.

Type: Long

Valid range: Minimum value of 1.

Required: Yes

ProvisionedThroughputDescription

Description

Represents the provisioned throughput settings for the table, consisting of read and write capacity units, along with data about increases and decreases.

Contents

Note
In the following list, the required parameters are described first.

LastDecreaseDateTime
The date and time of the last provisioned throughput decrease for this table.

Type: DateTime

Required: No

LastIncreaseDateTime
The date and time of the last provisioned throughput increase for this table.

Type: DateTime

Required: No

NumberOfDecreasesToday
The number of provisioned throughput decreases for this table during this UTC calendar day. For current maximums on provisioned throughput decreases, see Limits in the *Amazon DynamoDB Developer Guide*.

Type: Long

Valid range: Minimum value of 1.

Required: No

ReadCapacityUnits
The maximum number of strongly consistent reads consumed per second before DynamoDB returns a *ThrottlingException*. Eventually consistent reads require less effort than strongly consistent reads, so a setting of 50 *ReadCapacityUnits* per second provides 100 eventually consistent *ReadCapacityUnits* per second.

Type: Long

Valid range: Minimum value of 1.

Required: No

WriteCapacityUnits
The maximum number of writes consumed per second before DynamoDB returns a *ThrottlingException*.

Type: Long

Valid range: Minimum value of 1.

Required: No

PutRequest

Description

Represents a request to perform a *PutItem* operation on an item.

Contents

Note
In the following list, the required parameters are described first.

Item
A map of attribute name to attribute values, representing the primary key of an item to be processed by *PutItem*. All of the table's primary key attributes must be specified, and their data types must match those of the table's key schema. If any attributes are present in the item which are part of an index key schema for the table, their types must match the index key schema.

Type: String to AttributeValue (p. 130) object map

Required: Yes

StreamSpecification

Description

Represents the DynamoDB Streams configuration for a table in DynamoDB.

Contents

Note
In the following list, the required parameters are described first.

StreamEnabled
Indicates whether DynamoDB Streams is enabled (true) or disabled (false) on the table.

Type: Boolean

Required: No

StreamViewType
The DynamoDB Streams settings for the table. These settings consist of:

- *StreamEnabled* - Indicates whether DynamoDB Streams is enabled (true) or disabled (false) on the table.
- *StreamViewType* - When an item in the table is modified, *StreamViewType* determines what information is written to the stream for this table. Valid values for *StreamViewType* are:
  - *KEYS_ONLY* - Only the key attributes of the modified item are written to the stream.
  - *NEW_IMAGE* - The entire item, as it appears after it was modified, is written to the stream.
  - *OLD_IMAGE* - The entire item, as it appeared before it was modified, is written to the stream.
  - *NEW_AND_OLD_IMAGES* - Both the new and the old item images of the item are written to the stream.

Type: String

Valid Values: `NEW_IMAGE | OLD_IMAGE | NEW_AND_OLD_IMAGES | KEYS_ONLY`

Required: No

TableDescription

Description

Represents the properties of a table.

Contents

Note
In the following list, the required parameters are described first.

AttributeDefinitions
An array of *AttributeDefinition* objects. Each of these objects describes one attribute in the table and index key schema.

Each *AttributeDefinition* object in this array is composed of:

- *AttributeName* - The name of the attribute.
- *AttributeType* - The data type for the attribute.

Type: array of AttributeDefinition (p. 130) objects

Required: No

CreationDateTime
The date and time when the table was created, in UNIX epoch time format.

Type: DateTime

Required: No

GlobalSecondaryIndexes
The global secondary indexes, if any, on the table. Each index is scoped to a given hash key value. Each element is composed of:

- *Backfilling* - If true, then the index is currently in the backfilling phase. Backfilling occurs only when a new global secondary index is added to the table; it is the process by which DynamoDB populates the new index with data from the table. (This attribute does not appear for indexes that were created during a *CreateTable* operation.)
- *IndexName* - The name of the global secondary index.
- *IndexSizeBytes* - The total size of the global secondary index, in bytes. DynamoDB updates this value approximately every six hours. Recent changes might not be reflected in this value.
- *IndexStatus* - The current status of the global secondary index:
  - *CREATING* - The index is being created.
  - *UPDATING* - The index is being updated.
  - *DELETING* - The index is being deleted.
  - *ACTIVE* - The index is ready for use.
- *ItemCount* - The number of items in the global secondary index. DynamoDB updates this value approximately every six hours. Recent changes might not be reflected in this value.
- *KeySchema* - Specifies the complete index key schema. The attribute names in the key schema must be between 1 and 255 characters (inclusive). The key schema must begin with the same hash key attribute as the table.
- *Projection* - Specifies attributes that are copied (projected) from the table into the index. These are in addition to the primary key attributes and index key attributes, which are automatically projected. Each attribute specification is composed of:
  - *ProjectionType* - One of the following:
    - `KEYS_ONLY` - Only the index and primary keys are projected into the index.
    - `INCLUDE` - Only the specified table attributes are projected into the index. The list of projected attributes are in *NonKeyAttributes*.
    - `ALL` - All of the table attributes are projected into the index.
  - *NonKeyAttributes* - A list of one or more non-key attribute names that are projected into the secondary index. The total count of attributes provided in *NonKeyAttributes*, summed across all of the secondary indexes, must not exceed 20. If you project the same attribute into two different indexes, this counts as two distinct attributes when determining the total.
- *ProvisionedThroughput* - The provisioned throughput settings for the global secondary index, consisting of read and write capacity units, along with data about increases and decreases.

If the table is in the `DELETING` state, no information about indexes will be returned.

Type: array of GlobalSecondaryIndexDescription (p. 143) objects

Required: No

ItemCount
The number of items in the specified table. DynamoDB updates this value approximately every six hours. Recent changes might not be reflected in this value.

Type: Long

Required: No

KeySchema
The primary key structure for the table. Each *KeySchemaElement* consists of:

- *AttributeName* - The name of the attribute.
- *KeyType* - The key type for the attribute. Can be either `HASH` or `RANGE`.

For more information about primary keys, see Primary Key in the *Amazon DynamoDB Developer Guide*.

Type: array of KeySchemaElement (p. 148) objects

Length constraints: Minimum of 1 item(s) in the list. Maximum of 2 item(s) in the list.

Required: No

LatestStreamArn
The Amazon Resource Name (ARN) that uniquely identifies the latest stream for this table.

Type: String

Length constraints: Minimum length of 37. Maximum length of 1024.

Required: No

LatestStreamLabel
A timestamp, in ISO 8601 format, for this stream.

Note that *LatestStreamLabel* is not a unique identifier for the stream, because it is possible that a stream from another table might have the same timestamp. However, the combination of the following three elements is guaranteed to be unique:

- the AWS customer ID.
- the table name.
- the *StreamLabel*.

Type: String

Required: No

LocalSecondaryIndexes
Represents one or more local secondary indexes on the table. Each index is scoped to a given hash key value. Tables with one or more local secondary indexes are subject to an item collection size limit, where the amount of data within a given item collection cannot exceed 10 GB. Each element is composed of:

- *IndexName* - The name of the local secondary index.
- *KeySchema* - Specifies the complete index key schema. The attribute names in the key schema must be between 1 and 255 characters (inclusive). The key schema must begin with the same hash key attribute as the table.
- *Projection* - Specifies attributes that are copied (projected) from the table into the index. These are in addition to the primary key attributes and index key attributes, which are automatically projected. Each attribute specification is composed of:

- *ProjectionType* - One of the following:
  - `KEYS_ONLY` - Only the index and primary keys are projected into the index.
  - `INCLUDE` - Only the specified table attributes are projected into the index. The list of projected attributes are in *NonKeyAttributes*.
  - `ALL` - All of the table attributes are projected into the index.
- *NonKeyAttributes* - A list of one or more non-key attribute names that are projected into the secondary index. The total count of attributes provided in *NonKeyAttributes*, summed across all of the secondary indexes, must not exceed 20. If you project the same attribute into two different indexes, this counts as two distinct attributes when determining the total.
- *IndexSizeBytes* - Represents the total size of the index, in bytes. DynamoDB updates this value approximately every six hours. Recent changes might not be reflected in this value.
- *ItemCount* - Represents the number of items in the index. DynamoDB updates this value approximately every six hours. Recent changes might not be reflected in this value.

If the table is in the `DELETING` state, no information about indexes will be returned.

Type: array of LocalSecondaryIndexDescription (p. 149) objects

Required: No

ProvisionedThroughput
The provisioned throughput settings for the table, consisting of read and write capacity units, along with data about increases and decreases.

Type: ProvisionedThroughputDescription (p. 151) object

Required: No

StreamSpecification
The current DynamoDB Streams configuration for the table.

Type: StreamSpecification (p. 153) object

Required: No

TableArn
The Amazon Resource Name (ARN) that uniquely identifies the table.

Type: String

Required: No

TableName
The name of the table.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: No

TableSizeBytes
The total size of the specified table, in bytes. DynamoDB updates this value approximately every six hours. Recent changes might not be reflected in this value.

Type: Long

Required: No

TableStatus
The current state of the table:

- *CREATING* - The table is being created.
- *UPDATING* - The table is being updated.
- *DELETING* - The table is being deleted.
- *ACTIVE* - The table is ready for use.

Type: String

Valid Values: `CREATING | UPDATING | DELETING | ACTIVE`

Required: No

UpdateGlobalSecondaryIndexAction

Description

Represents the new provisioned throughput settings to be applied to a global secondary index.

Contents

Note
In the following list, the required parameters are described first.

IndexName
The name of the global secondary index to be updated.

Type: String

Length constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.-]+`

Required: Yes

ProvisionedThroughput
Represents the provisioned throughput settings for a specified table or index. The settings can be modified using the *UpdateTable* operation.

For current minimum and maximum provisioned throughput values, see Limits in the *Amazon DynamoDB Developer Guide*.

Type: ProvisionedThroughput (p. 151) object

Required: Yes

WriteRequest

Description

Represents an operation to perform - either *DeleteItem* or *PutItem*. You can only request one of these operations, not both, in a single *WriteRequest*. If you do need to perform both of these operations, you will need to provide two separate *WriteRequest* objects.

Contents

Note
In the following list, the required parameters are described first.

DeleteRequest
A request to perform a *DeleteItem* operation.

Type: DeleteRequest (p. 138) object

Required: No

PutRequest
A request to perform a *PutItem* operation.

Type: PutRequest (p. 152) object

Required: No

Common Errors

This section lists the common errors that all actions return. Any action-specific errors are listed in the topic for the action.

IncompleteSignature
The request signature does not conform to AWS standards.

HTTP Status Code: 400

InternalFailure
The request processing has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

InvalidAction
The action or operation requested is invalid. Verify that the action is typed correctly.

HTTP Status Code: 400

InvalidClientTokenId
The X.509 certificate or AWS access key ID provided does not exist in our records.

HTTP Status Code: 403

InvalidParameterCombination
Parameters that must not be used together were used together.

HTTP Status Code: 400

InvalidParameterValue
An invalid or out-of-range value was supplied for the input parameter.

HTTP Status Code: 400

InvalidQueryParameter
The AWS query string is malformed or does not adhere to AWS standards.

HTTP Status Code: 400

MalformedQueryString
The query string contains a syntax error.

HTTP Status Code: 404

MissingAction
The request is missing an action or a required parameter.

HTTP Status Code: 400

MissingAuthenticationToken
The request must contain either a valid (registered) AWS access key ID or X.509 certificate.

HTTP Status Code: 403

MissingParameter
A required parameter for the specified action is not supplied.

HTTP Status Code: 400

OptInRequired
The AWS access key ID needs a subscription for the service.

HTTP Status Code: 403

RequestExpired
The request reached the service more than 15 minutes after the date stamp on the request or more than 15 minutes after the request expiration date (such as for pre-signed URLs), or the date stamp on the request is more than 15 minutes in the future.

HTTP Status Code: 400

ServiceUnavailable
The request has failed due to a temporary failure of the server.

HTTP Status Code: 503

Throttling
The request was denied due to request throttling.

HTTP Status Code: 400

ValidationError
The input fails to satisfy the constraints specified by an AWS service.

HTTP Status Code: 400