Skip to content

put_item

```
DynamoDB.Table.put_item(**kwargs)
```

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. You can return the item's attribute values in the same operation, using the `ReturnValues` parameter.

When you add an item, the primary key attributes are the only required attributes.

Empty String and Binary attribute values are allowed. Attribute values of type String and Binary must have a length greater than zero if the attribute is used as a key attribute for a table or index. Set type attributes cannot be empty.

Invalid Requests with empty values will be rejected with a `ValidationException` exception.

> **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 partition 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 `PutItem`, see Working with Items in the *Amazon DynamoDB Developer Guide*.

See also: AWS API Documentation

Request Syntax

```
response = table.put_item(
    Item={
        'string': 'string'|123|Binary(b'bytes')|True|None|set(['string'])|set([123])
    },
    Expected={
        'string': {
            'Value': 'string'|123|Binary(b'bytes')|True|None|set(['string'])|set([12
            'Exists': True|False,
            'ComparisonOperator': 'EQ'|'NE'|'IN'|'LE'|'LT'|'GE'|'GT'|'BETWEEN'|'NOT_
            'AttributeValueList': [
                'string'|123|Binary(b'bytes')|True|None|set(['string'])|set([123])|s
            ]
        }
    },
    ReturnValues='NONE'|'ALL_OLD'|'UPDATED_OLD'|'ALL_NEW'|'UPDATED_NEW',
    ReturnConsumedCapacity='INDEXES'|'TOTAL'|'NONE',
    ReturnItemCollectionMetrics='SIZE'|'NONE',
    ConditionalOperator='AND'|'OR',
    ConditionExpression=Attr('myattribute').eq('myvalue'),
    ExpressionAttributeNames={
        'string': 'string'
    },
    ExpressionAttributeValues={
        'string': 'string'|123|Binary(b'bytes')|True|None|set(['string'])|set([123])
    },
    ReturnValuesOnConditionCheckFailure='ALL_OLD'|'NONE'
)
```

PARAMETERS:

- **Item** (*dict*) –

  [REQUIRED]

  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 simple primary key, you only need to provide a value for the partition key. For a composite primary key, you must provide both values for both the partition key and the sort key.

Skip to content

u 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.

Empty String and Binary attribute values are allowed. Attribute values of type String and Binary must have a length greater than zero if the attribute is used as a key attribute for a table or index.

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.

- *(string)* –
  - *(valid DynamoDB type) – *- The value of the attribute. The valid value types are listed in the DynamoDB Reference Guide.

- **Expected** (*dict*) –

This is a legacy parameter. Use `ConditionExpression` instead. For more information, see Expected in the *Amazon DynamoDB Developer Guide*.

- *(string) –*
  - *(dict) –*

    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.

- **Value** *(valid DynamoDB type) – *- The value of the attribute. The valid value types are listed in the DynamoDB Reference Guide.
- **Exists** *(boolean) –*

  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 `ConditionCheckFailedException`.
  - 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 `ConditionCheckFailedException`.

  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.)
- **ComparisonOperator** *(string) –*

  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 data types, 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 data types, 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"]}`.

Skip to content

- `NOT_NULL` : The attribute exists. `NOT_NULL` is supported for all data types, 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 data types, 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 in a list. `AttributeValueList` can contain one or more `AttributeValue` elements of type String, Number, or Binary. These attributes are compared against an existing attribute of an item. If any elements of the input are equal to the item attribute, the expression evaluates to true.

Skip to content

        - `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"]}`

    - **AttributeValueList** *(list)* –

      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*.

      - *(valid DynamoDB type) – *- The value of the attribute. The valid value types are listed in the DynamoDB Reference Guide.

- **ReturnValues** (*string*) –

  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.

  The values returned are strongly consistent.

  There is no additional cost associated with requesting a return value aside from a small network and processing overhead of receiving a larger response. No read capacity units are consumed.

> **Note**
>
> The `ReturnValues` parameter is used by several DynamoDB operations; however, `PutItem` does not recognize any values other than `NONE` or `ALL_OLD`.

- **ReturnConsumedCapacity** (*string*) –

  Determines the level of detail about either provisioned or on-demand 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.

- **ReturnItemCollectionMetrics** (*string*) – 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.
- **ConditionalOperator** (*string*) – This is a legacy parameter. Use `ConditionExpression` instead. For more information, see ConditionalOperator in the *Amazon DynamoDB Developer Guide*.
- **ConditionExpression** (condition from `boto3.dynamodb.conditions.Attr` method) – The condition(s) an attribute(s) must meet. Valid conditions are listed in the DynamoDB Reference Guide.
- **ExpressionAttributeNames** (*dict*) –

  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 Specifying Item Attributes in the *Amazon DynamoDB Developer Guide*.

- *(string)* –
  - *(string)* –

- **ExpressionAttributeValues** (*dict*) –

  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 Condition Expressions in the *Amazon DynamoDB Developer Guide*.

  - *(string)* –

    *(valid DynamoDB type) – *- The value of the attribute. The valid value types are listed in the DynamoDB Reference Guide.

- **ReturnValuesOnConditionCheckFailure** (*string*) –

An optional parameter that returns the item attributes for a `PutItem` operation that failed a condition check.

There is no additional cost associated with requesting a return value aside from the small network and processing overhead of receiving a larger response. No read capacity units are consumed.

RETURN TYPE:

dict

RETURNS:

Response Syntax

```
{
    'Attributes': {
        'string': 'string'|123|Binary(b'bytes')|True|None|set(['string'])|set([
    },
    'ConsumedCapacity': {
        'TableName': 'string',
        'CapacityUnits': 123.0,
        'ReadCapacityUnits': 123.0,
        'WriteCapacityUnits': 123.0,
        'Table': {
            'ReadCapacityUnits': 123.0,
            'WriteCapacityUnits': 123.0,
            'CapacityUnits': 123.0
        },
        'LocalSecondaryIndexes': {
            'string': {
                'ReadCapacityUnits': 123.0,
                'WriteCapacityUnits': 123.0,
                'CapacityUnits': 123.0
            }
        },
        'GlobalSecondaryIndexes': {
            'string': {
                'ReadCapacityUnits': 123.0,
                'WriteCapacityUnits': 123.0,
                'CapacityUnits': 123.0
            }
        }
    },
    'ItemCollectionMetrics': {
        'ItemCollectionKey': {
            'string': 'string'|123|Binary(b'bytes')|True|None|set(['string'])|s
        },
        'SizeEstimateRangeGB': [
            123.0,
        ]
    }
}
```

Response Structure

- *(dict)* –

Represents the output of a `PutItem` operation.

- **Attributes** *(dict)* –

  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.

  - *(string)* –
    - *(valid DynamoDB type) – *- The value of the attribute. The valid value types are listed in the DynamoDB Reference Guide.
- **ConsumedCapacity** *(dict)* –

The capacity units consumed by the `PutItem` 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 `ReturnConsumedCapacity` parameter was specified. For more information, see Capacity unity consumption for write operations in the *Amazon DynamoDB Developer Guide*.

- **TableName** *(string)* –

  The name of the table that was affected by the operation. If you had specified the Amazon Resource Name (ARN) of a table in the input, you'll see the table ARN in the response.

- **CapacityUnits** *(float)* –

  The total number of capacity units consumed by the operation.

- **ReadCapacityUnits** *(float)* –

  The total number of read capacity units consumed by the operation.

- **WriteCapacityUnits** *(float)* –

  The total number of write capacity units consumed by the operation.

- **Table** *(dict)* –

  The amount of throughput consumed on the table affected by the operation.

  - **ReadCapacityUnits** *(float)* –

    The total number of read capacity units consumed on a table or an index.

  - **WriteCapacityUnits** *(float)* –

    The total number of write capacity units consumed on a table or an index.

  - **CapacityUnits** *(float)* –

    The total number of capacity units consumed on a table or an index.

- **LocalSecondaryIndexes** *(dict)* –

The amount of throughput consumed on each local index affected by the operation.

- *(string) –*
  - *(dict) –*

    Represents the amount of provisioned throughput capacity consumed on a table or an index.

    - **ReadCapacityUnits** *(float) –*

      The total number of read capacity units consumed on a table or an index.

    - **WriteCapacityUnits** *(float) –*

      The total number of write capacity units consumed on a table or an index.

    - **CapacityUnits** *(float) –*

      The total number of capacity units consumed on a table or an index.

- **GlobalSecondaryIndexes** *(dict) –*

  The amount of throughput consumed on each global index affected by the operation.

  - *(string) –*
    - *(dict) –*

      Represents the amount of provisioned throughput capacity consumed on a table or an index.

      - **ReadCapacityUnits** *(float) –*

        The total number of read capacity units consumed on a table or an index.

      - **WriteCapacityUnits** *(float) –*

        The total number of write capacity units consumed on a table or an index.

      - **CapacityUnits** *(float) –*

        The total number of capacity units consumed on a table or an index.

- **ItemCollectionMetrics** *(dict) –*

Information about item collections, if any, that were affected by the `PutItem` operation. `ItemCollectionMetrics` is only returned if the `ReturnItemCollectionMetrics` parameter was specified. 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 partition key value of the item collection. This is the same as the partition key value of the item itself.
- `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.
- **ItemCollectionKey** *(dict)* –

  The partition key value of the item collection. This value is the same as the partition key value of the item.

  - *(string)* –
    - *(valid DynamoDB type) – *- The value of the attribute. The valid value types are listed in the DynamoDB Reference Guide.
- **SizeEstimateRangeGB** *(list)* –

  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.

  - *(float)* –