# Welcome to Amazon SimpleDB

Amazon SimpleDB is a web service for running queries on structured data in real time. This service works in close conjunction with Amazon Simple Storage Service (Amazon S3) and Amazon Elastic Compute Cloud (Amazon EC2), collectively providing the ability to store, process and query data sets in the cloud. These services are designed to make web-scale computing easier and more cost-effective for developers.

**Creating a Domain**

The following is an example of creating a domain using REST.

https://sdb.amazonaws.com/

?Action=CreateDomain

&AWSAccessKeyId=[valid access key id]

&DomainName=MyDomain

&SignatureVersion=2

&SignatureMethod=HmacSHA256

&Timestamp=2010-01-25T15%3A01%3A28-07%3A00

&Version=2009-04-15

&Signature=[valid signature]

Amazon SimpleDB returns output similar to the following.

<CreateDomainResponse>

<ResponseMetadata>

<RequestId>2a1305a2-ed1c-43fc-b7c4-e6966b5e2727</RequestId>

<BoxUsage>0.0000219907</BoxUsage>

</ResponseMetadata>

</CreateDomainResponse>

# Enabling SimpleDB service for AWS account

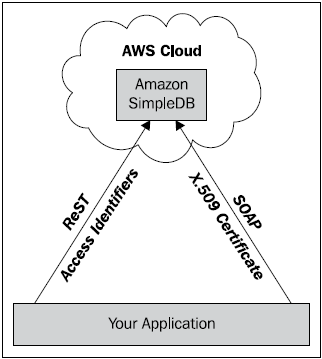
Once you have successfully set up an AWS account, you must follow these steps to enable the SimpleDB service for your account:

1. Log in to your AWS account.
2. Navigate to the SimpleDB home page—http://aws.amazon.com/simpledb/.
3. Click on the **Sign Up For Amazon SimpleDB** button on the right side of the page.
4. Provide the requested credit card information and complete the signup process.



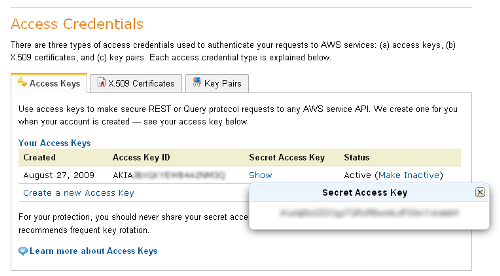
You have now successfully set up your AWS account and enabled it for SimpleDB.

All communication with SimpleDB or any of the Amazon web services must be through either the SOAP interface or the Query/ReST interface. The request messages sent through either of these interfaces is digitally signed by the sending user in order to ensure that the messages have not been tampered within transit, and that they really originate from the sending user. Requests that use the Query/ReST interface will use the access keys for signing the request, whereas requests to the SOAP interface will use the x.509 certificates.

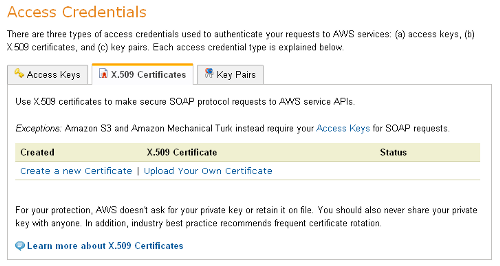


Your new AWS account is associated with the following items:

* A unique 12-digit AWS account number for identifying your account.
* AWS Access Credentials are used for the purpose of authenticating requests made by you through the ReST Request API to any of the web services provided by AWS. An initial set of keys is automatically generated for you by default. You can regenerate the Secret Access Key at any time if you like. Keep in mind that when you generate a new access key, all requests made using the old key will be rejected.
  + An Access Key ID identifies you as the person making requests to a web service.
  + A Secret Access Key is used to calculate the digital signature when you make requests to the web service.
  + Be careful with your Secret Access Key, as it provides full access to the account, including the ability to delete all of your data.



* All requests made to any of the web services provided by AWS using the SOAP protocol use the X.509 security certificate for authentication. There are no default certificates generated automatically for you by AWS. You must generate the certificate by clicking on the **Create a new Certificate** link, then download them to your computer and make them available to the machine that will be making requests to AWS.
  + Public and private key for the x.509 certificate. You can either upload your own x.509 certificate if you already have one, or you can just generate a new certificate and then download it to your computer.



# Query API and authentication

There are two interfaces to SimpleDB. The SOAP interface uses the SOAP protocol for the messages, while the ReST Requests uses HTTP requests with request parameters to describe the various SimpleDB methods and operations. In this book, we will be focusing on using the ReST Requests for talking to SimpleDB, as it is a much simpler protocol and utilizes straightforward HTTP-based requests and responses for communication, and the requests are sent to SimpleDB using either a HTTP *GET* or *POST* method.

The ReST Requests need to be authenticated in order to establish that they are originating from a valid SimpleDB user, and also for accounting and billing purposes. This authentication is performed using your access key identifiers. Every request to SimpleDB must contain a request signature calculated by constructing a string based on the Query API and then calculating an RFC 2104-compliant HMAC-SHA1 hash, using the Secret Access Key.

The basic steps in the authentication of a request by SimpleDB are:

* You construct a request to SimpleDB.
* You use your Secret Access Key to calculate the request signature, a Keyed-Hashing for **Message Authentication code (HMAC)** with an SHA1 hash function.
* You send the request data, the request signature, timestamp, and your Access Key ID to AWS.
* AWS uses the Access Key ID in the request to look up the associated Secret Access Key.
* AWS generates a request signature from the request data using the retrieved Secret Access Key and the same algorithm you used to calculate the signature in the request.
* If the signature generated by AWS matches the one you sent in the request, the request is considered to be authentic. If the signatures are different, the request is discarded, and AWS returns an error response. If the timestamp is older than 15 minutes, the request is rejected.

The procedure for constructing your requests is simple, but tedious and time consuming. This overview was intended to make you familiar with the entire process, but don’t worry—you will not need to go through this laborious process every single time that you interact with SimpleDB. Instead, we will be leveraging one of the available libraries for communicating with SimpleDB, which encapsulates a lot of the repetitive stuff for us and makes it simple to dive straight into playing with and exploring SimpleDB!

## DynamoDB

**Amazon DynamoDB** is a fully managed NoSQL database service that allows to create database tables that can store and retrieve any amount of data. It automatically manages the data traffic of tables over multiple servers and maintains performance. It also relieves the customers from the burden of operating and scaling a distributed database. Hence, hardware provisioning, setup, configuration, replication, software patching, cluster scaling, etc. is managed by Amazon.

## How to Run DynamoDB on Computer?

Following are the steps to set up DynamoDB.

**Step 1** − Following are the steps to set up DynamoDB.

* Download DynamoDB (.jar file) using the following link. It supports multiple Operating Systems like Windows, Linux, Mac, etc.

.tar.gz **format** − http://dynamodb-local.s3-website-us-west2.amazonaws.com/dynamodb\_local\_latest.tar.gz

.zip **format** − http://dynamodb-local.s3-website-us-west2.amazonaws.com/dynamodb\_local\_latest.zip.

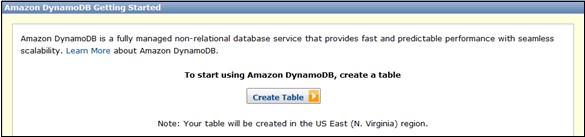
* Once download is complete, extract the contents and copy the extracted directory to a location wherever you want.
* Open the command prompt and navigate to the directory where you extracted DynamoDBLocal.jar, and execute the following command −

java -Djava.library.path=./DynamoDBLocal\_lib -jar DynamoDBLocal.jar -sharedDb

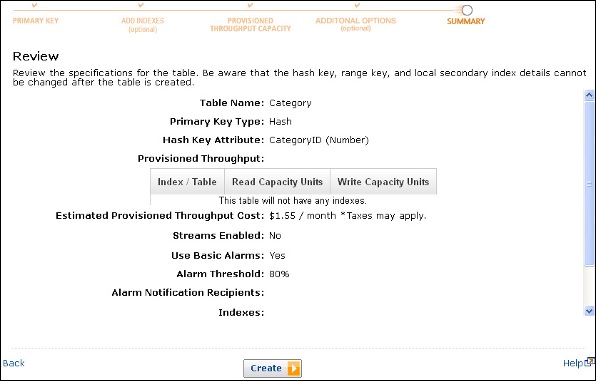
* Now there is access to the build-in javaScript shell.

**Step 2** − Create a Table using the following steps.

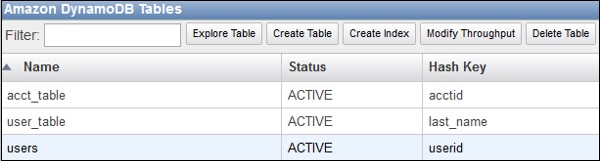
* Open AWS Management Console and select DynamoDB.
* Select the region where the table will be created and click the Create Table button.



* Create Table window opens. Fill the details into their respective fields and click the Continue button.
* Finally, a review page opens where we can view details. Click the Create button.



* Now the Table-name is visible in the in-to the list and Dynamo Table is ready to use.



## Benefits of Amazon DynamoDB

Managed **service** − Amazon DynamoDB is a managed service. There is no need to hire experts to manage NoSQL installation. Developers need not worry about setting up, configuring a distributed database cluster, managing ongoing cluster operations, etc. It handles all the complexities of scaling, partitions and re-partitions data over more machine resources to meet I/O performance requirements.

Scalable − Amazon DynamoDB is designed to scale. There is no need to worry about predefined limits to the amount of data each table can store. Any amount of data can be stored and retrieved. DynamoDB will spread automatically with the amount of data stored as the table grows.

Fast − Amazon DynamoDB provides high throughput at very low latency. As datasets grow, latencies remain stable due to the distributed nature of DynamoDB's data placement and request routing algorithms.

Durable and **h**ighly **a**vailable − Amazon DynamoDB replicates data over at least 3 different data centers’ results. The system operates and serves data even under various failure conditions.

Flexible: Amazon DynamoDB allows creation of dynamic tables, i.e. the table can have any number of attributes, including multi-valued attributes.

Cost-effective: Payment is for what we use without any minimum charges. Its pricing structure is simple and easy to calculate

While SimpleDB has scaling limitations, it may be a good fit for smaller workloads that require query flexibility. Amazon SimpleDB automatically indexes all item attributes and thus supports query flexibility at the cost of performance and scale.

It became obvious that developers [even Amazon engineers] strongly preferred simplicity to fine-grained control as they voted "with their feet" and adopted cloud-based AWS solutions, like Amazon S3 and Amazon SimpleDB, over Dynamo. [addition mine]

We concluded that an ideal solution would combine the best parts of the original Dynamo design (incremental scalability, predictable high performance) with the best parts of SimpleDB (ease of administration of a cloud service, consistency, and a table-based data model that is richer than a pure key-value store).

Amazon DynamoDB is designed to maintain predictably high performance and to be highly cost efficient for workloads of any scale, from the smallest to the largest internet-scale applications.

Commands:

# EXAMPLES

# The following examples assume you have set these environment variables:

|  |
| --- |
| export AWS\_ACCESS\_KEY\_ID=...  export AWS\_SECRET\_ACCESS\_KEY=...  export SDB\_SERVICE\_URL=[https://sdb.eu-west-1.amazonaws.com](https://sdb.eu-west-1.amazonaws.com/) |

# Create a new SimpleDB domain:

|  |
| --- |
| simpledb create-domain mydomain |

# List the domains for this account:

|  |
| --- |
| simpledb list-domains |

# Create some items with attribute name=value pairs:

|  |
| --- |
| simpledb put mydomain item1 key1=valueA key2=value2 x=why    simpledb put mydomain item2 key1=valueB key2=value2 y=zee |

# Add another value for an attribute on an item:

|  |
| --- |
| simpledb put mydomain item2 y=zed when=now who=you |

# Replace all values for specific attributes on an item:

|  |
| --- |
| simpledb put-replace mydomain item1 key1=value1 newkey=newvalue |

# Delete all values for specific attributes on an item:

|  |
| --- |
| simpledb delete mydomain item1 x |

# Delete specific values for specific attributes on an item:

|  |
| --- |
| simpledb delete mydomain item2 who=you |

# List all item names in a domain - note backquotes around domain

|  |
| --- |
| simpledb select 'select itemName() from `my-domain`' |

# List all items and their attributes matching a given select query:

|  |
| --- |
| simpledb select 'select \* from mydomain where key2="value2"' |

# List all attributes on an item:

|  |
| --- |
| simpledb get mydomain item1    simpledb get mydomain item2 |

# Delete the entire SimpleDB domain including all items and attributes:

|  |
| --- |
| simpledb delete-domain mydomain |

# ENVIRONMENT

|  |
| --- |
| AWS\_ACCESS\_KEY\_ID       Default AWS access key id    AWS\_SECRET\_ACCESS\_KEY       Default AWS secret access key    SDB\_SERVICE\_URL       Default <https://sdb.amazonaws.com/>       Alternatives:  <http://docs.aws.amazon.com/general/latest/gr/rande.html>#sdb\_region |

# FILES

|  |
| --- |
| $HOME/.awssecret     If the above fail, then the keys are sought here in the     format expected by the "aws" toolkit (one per line):       access\_key\_id       secret\_access\_key    /etc/passwd-s3fs     If all of the above fail, then the keys are sought     here in the format expected by s3fs (colon separated):       access\_key\_id:secret\_access\_key |

# CAVEATS

As currently written this tool does not support keys containing equal signs (=).

Output will be difficult to parse if the values contain newlines.

# HISTORY

|  |
| --- |
| 2013-09-11 Andrew Solomon <andrew at illywhacker dot net>  - Rebased on SimpleDB::Client  - Removed proxying  - Removed attribute parameters from "get"    2010-04-20 Eric Hammond <ehammond@thinksome.com>  - Removed support for "query".  Please migrate to "select"    2009-09-01 Peter Kaminski <kaminski@istori.com>  - Added utf8 binmode for STDOUT  - Added select method    2009-03-16 Eric Hammond <ehammond@thinksome.com>  - Fix --max options and large result sets without --max  <http://code.google.com/p/amazon-simpledb-cli/issues/detail?id=2>    2008-06-09 Eric Hammond <ehammond@thinksome.com>  - Fallback to finding keys in $HOME/.awssecret or /etc/passwd-s3fs    2008-06-03 Eric Hammond <ehammond@thinksome.com>  - Completed --max option  - bugfix: Corrected --aws-secret-access-key option spelling    2008-05-26 Eric Hammond <ehammond@thinksome.com>  - Original release |

# AWS SimpleDB Boto3 Example

## [Client](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sdb.html#id13)

**class SimpleDB.Client**

A low-level client representing Amazon SimpleDB:

**import** **boto3**

client **=** boto3**.**client('sdb')

These are the available methods:

* [**batch\_delete\_attributes()**](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sdb.html#SimpleDB.Client.batch_delete_attributes)
* [**batch\_put\_attributes()**](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sdb.html#SimpleDB.Client.batch_put_attributes)
* [**can\_paginate()**](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sdb.html#SimpleDB.Client.can_paginate)
* [**create\_domain()**](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sdb.html#SimpleDB.Client.create_domain)
* [**delete\_attributes()**](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sdb.html#SimpleDB.Client.delete_attributes)
* [**delete\_domain()**](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sdb.html#SimpleDB.Client.delete_domain)
* [**domain\_metadata()**](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sdb.html#SimpleDB.Client.domain_metadata)
* [**generate\_presigned\_url()**](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sdb.html#SimpleDB.Client.generate_presigned_url)
* [**get\_attributes()**](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sdb.html#SimpleDB.Client.get_attributes)
* [**get\_paginator()**](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sdb.html#SimpleDB.Client.get_paginator)
* [**get\_waiter()**](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sdb.html#SimpleDB.Client.get_waiter)
* [**list\_domains()**](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sdb.html#SimpleDB.Client.list_domains)
* [**put\_attributes()**](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sdb.html#SimpleDB.Client.put_attributes)
* [**select()**](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sdb.html#SimpleDB.Client.select)

**batch\_delete\_attributes(\*\*kwargs)**

Performs multiple DeleteAttributes operations in a single call, which reduces round trips and latencies. This enables Amazon SimpleDB to optimize requests, which generally yields better throughput.

The following limitations are enforced for this operation:

* 1 MB request size
* 25 item limit per BatchDeleteAttributes operation

**Request Syntax**

response **=** client**.**batch\_delete\_attributes(

DomainName**=**'string',

Items**=**[

{

'Name': 'string',

'Attributes': [

{

'Name': 'string',

'AlternateNameEncoding': 'string',

'Value': 'string',

'AlternateValueEncoding': 'string'

},

]

},

]

)

**Parameters**

* **DomainName** (*string*) -- **[REQUIRED]** The name of the domain in which the attributes are being deleted.
* **Items** (*list*) --

**[REQUIRED]** A list of items on which to perform the operation.

* + *(dict) --*
    - **Name** *(string) --* **[REQUIRED]**
    - **Attributes** *(list) --*
      * *(dict) --*
        + **Name** *(string) --* **[REQUIRED]** The name of the attribute.
        + **AlternateNameEncoding** *(string) --*
        + **Value** *(string) --* **[REQUIRED]** The value of the attribute.
        + **AlternateValueEncoding** *(string) --*

**Returns**

None

**batch\_put\_attributes(*\*\*kwargs*)**

The BatchPutAttributes operation creates or replaces attributes within one or more items. By using this operation, the client can perform multiple PutAttribute operation with a single call. This helps yield savings in round trips and latencies, enabling Amazon SimpleDB to optimize requests and generally produce better throughput.

The client may specify the item name with the Item.X.ItemName parameter. The client may specify new attributes using a combination of the Item.X.Attribute.Y.Name and Item.X.Attribute.Y.Valueparameters. The client may specify the first attribute for the first item using the parameters Item.0.Attribute.0.Name and Item.0.Attribute.0.Value , and for the second attribute for the first item by the parameters Item.0.Attribute.1.Name and Item.0.Attribute.1.Value , and so on.

Attributes are uniquely identified within an item by their name/value combination. For example, a single item can have the attributes { "first\_name", "first\_value" } and { "first\_name","second\_value" } . However, it cannot have two attribute instances where both the Item.X.Attribute.Y.Name and Item.X.Attribute.Y.Value are the same.

Optionally, the requester can supply the Replace parameter for each individual value. Setting this value to true will cause the new attribute values to replace the existing attribute values. For example, if an item I has the attributes { 'a', '1' }, { 'b', '2'} and { 'b', '3' } and the requester does a BatchPutAttributes of {'I', 'b', '4' } with the Replace parameter set to true, the final attributes of the item will be { 'a', '1' } and { 'b', '4' } , replacing the previous values of the 'b' attribute with the new value.

You can execute multiple BatchPutAttributes operations and other operations in parallel. However, large numbers of concurrent BatchPutAttributes calls can result in Service Unavailable (503) responses.

The following limitations are enforced for this operation:

* 256 attribute name-value pairs per item
* 1 MB request size
* 1 billion attributes per domain
* 10 GB of total user data storage per domain
* 25 item limit per BatchPutAttributes operation

**Request Syntax**

response **=** client**.**batch\_put\_attributes(

DomainName**=**'string',

Items**=**[

{

'Name': 'string',

'Attributes': [

{

'Name': 'string',

'Value': 'string',

'Replace': True**|**False

},

]

},

]

)

**Parameters**

* **DomainName** (*string*) -- **[REQUIRED]** The name of the domain in which the attributes are being stored.
* **Items** (*list*) --

**[REQUIRED]** A list of items on which to perform the operation.

* + *(dict) --*
    - **Name** *(string) --* **[REQUIRED]** The name of the replaceable item.
    - **Attributes** *(list) --* **[REQUIRED]** The list of attributes for a replaceable item.
      * *(dict) --*
        + **Name** *(string) --* **[REQUIRED]** The name of the replaceable attribute.
        + **Value** *(string) --* **[REQUIRED]** The value of the replaceable attribute.
        + **Replace** *(boolean) --* A flag specifying whether or not to replace the attribute/value pair or to add a new attribute/value pair. The default setting is false .

**Returns**

None

**can\_paginate(*operation\_name*)**

Check if an operation can be paginated.

**Parameters**

**operation\_name** (*string*) -- The operation name. This is the same name as the method name on the client. For example, if the method name is create\_foo, and you'd normally invoke the operation as client.create\_foo(\*\*kwargs), if the create\_foo operation can be paginated, you can use the call client.get\_paginator("create\_foo").

**Returns**

True if the operation can be paginated, False otherwise.

**create\_domain(*\*\*kwargs*)**

The CreateDomain operation creates a new domain. The domain name should be unique among the domains associated with the Access Key ID provided in the request. The CreateDomain operation may take 10 or more seconds to complete.

The client can create up to 100 domains per account.

**Request Syntax**

response **=** client**.**create\_domain(

DomainName**=**'string'

)

**Parameters**

**DomainName** (*string*) -- **[REQUIRED]** The name of the domain to create. The name can range between 3 and 255 characters and can contain the following characters: a-z, A-Z, 0-9, '\_', '-', and '.'.

**Returns**

None

**delete\_attributes(*\*\*kwargs*)**

Deletes one or more attributes associated with an item. If all attributes of the item are deleted, the item is deleted.

**Request Syntax**

response **=** client**.**delete\_attributes(

DomainName**=**'string',

ItemName**=**'string',

Attributes**=**[

{

'Name': 'string',

'AlternateNameEncoding': 'string',

'Value': 'string',

'AlternateValueEncoding': 'string'

},

],

Expected**=**{

'Name': 'string',

'Value': 'string',

'Exists': True**|**False

}

)

**Parameters**

* **DomainName** (*string*) -- **[REQUIRED]** The name of the domain in which to perform the operation.
* **ItemName** (*string*) -- **[REQUIRED]** The name of the item. Similar to rows on a spreadsheet, items represent individual objects that contain one or more value-attribute pairs.
* **Attributes** (*list*) --

A list of Attributes. Similar to columns on a spreadsheet, attributes represent categories of data that can be assigned to items.

* + *(dict) --*
    - **Name** *(string) --* **[REQUIRED]** The name of the attribute.
    - **AlternateNameEncoding** *(string) --*
    - **Value** *(string) --* **[REQUIRED]** The value of the attribute.
    - **AlternateValueEncoding** *(string) --*
* **Expected** (*dict*) --

The update condition which, if specified, determines whether the specified attributes will be deleted or not. The update condition must be satisfied in order for this request to be processed and the attributes to be deleted.

* + **Name** *(string) --*

The name of the attribute involved in the condition.

* + **Value** *(string) --*

The value of an attribute. This value can only be specified when the Exists parameter is equal to true .

* + **Exists** *(boolean) --*

A value specifying whether or not the specified attribute must exist with the specified value in order for the update condition to be satisfied. Specify true if the attribute must exist for the update condition to be satisfied. Specify false if the attribute should not exist in order for the update condition to be satisfied.

**Returns**

None

**delete\_domain(*\*\*kwargs*)**

The DeleteDomain operation deletes a domain. Any items (and their attributes) in the domain are deleted as well. The DeleteDomain operation might take 10 or more seconds to complete.

**Request Syntax**

response **=** client**.**delete\_domain(

DomainName**=**'string'

)

**Parameters**

**DomainName** (*string*) -- **[REQUIRED]** The name of the domain to delete.

**Returns**

None

**domain\_metadata(*\*\*kwargs*)**

Returns information about the domain, including when the domain was created, the number of items and attributes in the domain, and the size of the attribute names and values.

**Request Syntax**

response **=** client**.**domain\_metadata(

DomainName**=**'string'

)

**Parameters**

**DomainName** (*string*) -- **[REQUIRED]** The name of the domain for which to display the metadata of.

**Return type**

dict

**Returns**

**Response Syntax**

{

'ItemCount': 123,

'ItemNamesSizeBytes': 123,

'AttributeNameCount': 123,

'AttributeNamesSizeBytes': 123,

'AttributeValueCount': 123,

'AttributeValuesSizeBytes': 123,

'Timestamp': 123

}

**Response Structure**

* *(dict) --*
  + **ItemCount** *(integer) --* The number of all items in the domain.
  + **ItemNamesSizeBytes** *(integer) --* The total size of all item names in the domain, in bytes.
  + **AttributeNameCount** *(integer) --* The number of unique attribute names in the domain.
  + **AttributeNamesSizeBytes** *(integer) --* The total size of all unique attribute names in the domain, in bytes.
  + **AttributeValueCount** *(integer) --* The number of all attribute name/value pairs in the domain.
  + **AttributeValuesSizeBytes** *(integer) --* The total size of all attribute values in the domain, in bytes.
  + **Timestamp** *(integer) --* The data and time when metadata was calculated, in Epoch (UNIX) seconds.

**generate\_presigned\_url(*ClientMethod*, *Params=None*, *ExpiresIn=3600*, *HttpMethod=None*)**

Generate a presigned url given a client, its method, and arguments

**Parameters**

* **ClientMethod** (*string*) -- The client method to presign for
* **Params** (*dict*) -- The parameters normally passed to ClientMethod.
* **ExpiresIn** (*int*) -- The number of seconds the presigned url is valid for. By default it expires in an hour (3600 seconds)
* **HttpMethod** (*string*) -- The http method to use on the generated url. By default, the http method is whatever is used in the method's model.

**get\_attributes(*\*\*kwargs*)**

Returns all of the attributes associated with the specified item. Optionally, the attributes returned can be limited to one or more attributes by specifying an attribute name parameter.

If the item does not exist on the replica that was accessed for this operation, an empty set is returned. The system does not return an error as it cannot guarantee the item does not exist on other replicas.

**Request Syntax**

response **=** client**.**get\_attributes(

DomainName**=**'string',

ItemName**=**'string',

AttributeNames**=**[

'string',

],

ConsistentRead**=**True**|**False

)

**Parameters**

* **DomainName** (*string*) -- **[REQUIRED]** The name of the domain in which to perform the operation.
* **ItemName** (*string*) -- **[REQUIRED]** The name of the item.
* **AttributeNames** (*list*) --

The names of the attributes.

* + *(string) --*
* **ConsistentRead** (*boolean*) -- Determines whether or not strong consistency should be enforced when data is read from SimpleDB. If true , any data previously written to SimpleDB will be returned. Otherwise, results will be consistent eventually, and the client may not see data that was written immediately before your read.

**Return type**

dict

**Returns**

**Response Syntax**

{

'Attributes': [

{

'Name': 'string',

'AlternateNameEncoding': 'string',

'Value': 'string',

'AlternateValueEncoding': 'string'

},

]

}

**Response Structure**

* *(dict) --*
  + **Attributes** *(list) --* The list of attributes returned by the operation.
    - *(dict) --*
      * **Name** *(string) --* The name of the attribute.
      * **AlternateNameEncoding** *(string) --*
      * **Value** *(string) --* The value of the attribute.
      * **AlternateValueEncoding** *(string) --*

**list\_domains(*\*\*kwargs*)**

The ListDomains operation lists all domains associated with the Access Key ID. It returns domain names up to the limit set by [MaxNumberOfDomains](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sdb.html#MaxNumberOfDomains) . A [NextToken](https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sdb.html#NextToken) is returned if there are more than MaxNumberOfDomains domains. Calling ListDomains successive times with the NextTokenprovided by the operation returns up to MaxNumberOfDomains more domain names with each successive operation call.

**Request Syntax**

response **=** client**.**list\_domains(

MaxNumberOfDomains**=**123,

NextToken**=**'string'

)

**Parameters**

* **MaxNumberOfDomains** (*integer*) -- The maximum number of domain names you want returned. The range is 1 to 100. The default setting is 100.
* **NextToken** (*string*) -- A string informing Amazon SimpleDB where to start the next list of domain names.

**Return type**

dict

**Returns**

**Response Syntax**

{

'DomainNames': [

'string',

],

'NextToken': 'string'

}

**Response Structure**

* *(dict) --*
  + **DomainNames** *(list) --* A list of domain names that match the expression.
    - *(string) --*
  + **NextToken** *(string) --* An opaque token indicating that there are more domains than the specified MaxNumberOfDomains still available.

**put\_attributes(*\*\*kwargs*)**

The PutAttributes operation creates or replaces attributes in an item. The client may specify new attributes using a combination of the Attribute.X.Name and Attribute.X.Value parameters. The client specifies the first attribute by the parameters Attribute.0.Name and Attribute.0.Value , the second attribute by the parameters Attribute.1.Name and Attribute.1.Value , and so on.

Attributes are uniquely identified in an item by their name/value combination. For example, a single item can have the attributes { "first\_name", "first\_value" } and { "first\_name",second\_value" } . However, it cannot have two attribute instances where both the Attribute.X.Name and Attribute.X.Value are the same.

Optionally, the requestor can supply the Replace parameter for each individual attribute. Setting this value to true causes the new attribute value to replace the existing attribute value(s). For example, if an item has the attributes { 'a', '1' } , { 'b', '2'} and { 'b', '3' } and the requestor calls PutAttributes using the attributes { 'b', '4' } with the Replace parameter set to true, the final attributes of the item are changed to { 'a', '1' } and { 'b', '4' } , which replaces the previous values of the 'b' attribute with the new value.

You cannot specify an empty string as an attribute name.

Because Amazon SimpleDB makes multiple copies of client data and uses an eventual consistency update model, an immediate GetAttributes or Select operation (read) immediately after a PutAttributes or DeleteAttributes operation (write) might not return the updated data.

The following limitations are enforced for this operation:

* 256 total attribute name-value pairs per item
* One billion attributes per domain
* 10 GB of total user data storage per domain

See also: [AWS API Documentation](https://docs.aws.amazon.com/goto/WebAPI/sdb-2009-04-15/PutAttributes)

**Request Syntax**

response **=** client**.**put\_attributes(

DomainName**=**'string',

ItemName**=**'string',

Attributes**=**[

{

'Name': 'string',

'Value': 'string',

'Replace': True**|**False

},

],

Expected**=**{

'Name': 'string',

'Value': 'string',

'Exists': True**|**False

}

)

**Parameters**

* **DomainName** (*string*) -- **[REQUIRED]** The name of the domain in which to perform the operation.
* **ItemName** (*string*) -- **[REQUIRED]** The name of the item.
* **Attributes** (*list*) --

**[REQUIRED]** The list of attributes.

* + *(dict) --*
    - **Name** *(string) --* **[REQUIRED]** The name of the replaceable attribute.
    - **Value** *(string) --* **[REQUIRED]** The value of the replaceable attribute.
    - **Replace** *(boolean) --* A flag specifying whether or not to replace the attribute/value pair or to add a new attribute/value pair. The default setting is false .
* **Expected** (*dict*) --

The update condition which, if specified, determines whether the specified attributes will be updated or not. The update condition must be satisfied in order for this request to be processed and the attributes to be updated.

* + **Name** *(string) --*

The name of the attribute involved in the condition.

* + **Value** *(string) --*

The value of an attribute. This value can only be specified when the Exists parameter is equal to true .

* + **Exists** *(boolean) --*

A value specifying whether or not the specified attribute must exist with the specified value in order for the update condition to be satisfied. Specify true if the attribute must exist for the update condition to be satisfied. Specify false if the attribute should not exist in order for the update condition to be satisfied.

**Returns**

None

**select(*\*\*kwargs*)**

The Select operation returns a set of attributes for ItemNames that match the select expression. Select is similar to the standard SQL SELECT statement.

The total size of the response cannot exceed 1 MB in total size. Amazon SimpleDB automatically adjusts the number of items returned per page to enforce this limit. For example, if the client asks to retrieve 2500 items, but each individual item is 10 kB in size, the system returns 100 items and an appropriate NextToken so the client can access the next page of results.

For information on how to construct select expressions, see Using Select to Create Amazon SimpleDB Queries in the Developer Guide.

See also: [AWS API Documentation](https://docs.aws.amazon.com/goto/WebAPI/sdb-2009-04-15/Select)

**Request Syntax**

response **=** client**.**select(

SelectExpression**=**'string',

NextToken**=**'string',

ConsistentRead**=**True**|**False

)

**Parameters**

* **SelectExpression** (*string*) -- **[REQUIRED]** The expression used to query the domain.
* **NextToken** (*string*) -- A string informing Amazon SimpleDB where to start the next list of ItemNames .
* **ConsistentRead** (*boolean*) -- Determines whether or not strong consistency should be enforced when data is read from SimpleDB. If true , any data previously written to SimpleDB will be returned. Otherwise, results will be consistent eventually, and the client may not see data that was written immediately before your read.

**Return type**

dict

**Returns**

**Response Syntax**

{

'Items': [

{

'Name': 'string',

'AlternateNameEncoding': 'string',

'Attributes': [

{

'Name': 'string',

'AlternateNameEncoding': 'string',

'Value': 'string',

'AlternateValueEncoding': 'string'

},

]

},

],

'NextToken': 'string'

}

**Response Structure**

* *(dict) --*
  + **Items** *(list) --* A list of items that match the select expression.
    - *(dict) --*
      * **Name** *(string) --* The name of the item.
      * **AlternateNameEncoding** *(string) --*
      * **Attributes** *(list) --* A list of attributes.
        + *(dict) --*

**Name** *(string) --* The name of the attribute.

**AlternateNameEncoding** *(string) --*

**Value** *(string) --* The value of the attribute.

**AlternateValueEncoding** *(string) --*

* + **NextToken** *(string) --* An opaque token indicating that more items than MaxNumberOfItemswere matched, the response size exceeded 1 megabyte, or the execution time exceeded 5 seconds.

from \_\_future\_\_ import print\_function

import boto3

def quote(string):

return string.replace("'", "''").replace('"', '""').replace('`', '``')

def put\_attributes(sdb, domain, id, color):

response = sdb.put\_attributes(

DomainName=domain,

ItemName=id,

Attributes=[

{

'Name': 'color',

'Value': color,

'Replace': True

},

],

)

print(response)

if \_\_name\_\_ == "\_\_main\_\_":

domain = "TEST\_DOMAIN"

sdb = boto3.client('sdb')

response = sdb.create\_domain(

DomainName=domain

)

print(response)

response = sdb.list\_domains(

)

print("Current domains: %s" % response['DomainNames'])

put\_attributes(sdb, domain, "id1", "red")

put\_attributes(sdb, domain, "id2", "redblue")

put\_attributes(sdb, domain, "id3", "blue")

response = sdb.select(

SelectExpression='select \* from %s where color like "%%%s%%"' % (domain, quote('blue')),

)

print(response)

response = sdb.delete\_domain(

DomainName=domain

)

print(response)