



# **TIBCO® Data Virtualization**

## **Amazon DynamoDB Adapter Guide**

Version 8.7.0 | October 2023

# Contents

---

<b>Contents</b>	<b>2</b>
<b>Amazon DynamoDB Adapter</b>	<b>4</b>
Getting Started	4
Basic Tab	5
Logging	11
Fine-Tuning Data Access	13
Performance	14
Minimum IAM Requirements	15
Changelog	18
NoSQL Database	23
Automatic Schema Discovery	23
Vertical Flattening	25
JSON Functions	27
DynamoDB Queries	29
Querying Documents and Lists	31
Data Type Mapping	32
Custom Schema Definitions	33
Custom Schema Example	36
Advanced Features	38
Automatic Index Detection	39
User Defined Views	41
SSL Configuration	43
Firewall and Proxy	44
Query Processing	44
Logging	45
SQL Compliance	48
SELECT Statements	49

SELECT INTO Statements .....	53
INSERT Statements .....	53
UPDATE Statements .....	54
DELETE Statements .....	55
EXECUTE Statements .....	56
PIVOT and UNPIVOT .....	56
Data Model .....	58
Tables .....	58
Table Columns .....	58
Stored Procedures .....	60
Connection String Options .....	63
AWS Authentication .....	69
SSO .....	82
SSL .....	86
Firewall .....	88
Proxy .....	91
Logging .....	98
Schema .....	98
Miscellaneous .....	100
<b>TIBCO Product Documentation and Support Services .....</b>	<b>116</b>
How to Access TIBCO Documentation .....	116
How to Contact TIBCO Support .....	117
Release Version Support .....	117
How to Join TIBCO Community .....	118
<b>Legal and Third-Party Notices .....</b>	<b>119</b>

# Amazon DynamoDB Adapter

---

## Amazon DynamoDB Version Support

The adapter uses the current version of the Amazon DynamoDB REST API, version 2012-08-10, to enable read/write access to DynamoDB instances.

## SQL Compliance

The [SQL Compliance](#) section shows the SQL syntax supported by the adapter and points out any limitations.

# Getting Started

## Connecting to Amazon DynamoDB

[Basic Tab](#) shows how to authenticate to Amazon DynamoDB and configure any necessary connection properties. Additional adapter capabilities can be configured using the available [Connection](#) properties on the Advanced tab. The Advanced Settings section shows how to set up more advanced configurations and troubleshoot connection errors.

## Deploying the Amazon DynamoDB Adapter

To deploy the adapter, you can execute the `server_util` utility via the command line by

1. Unzip the `tdv.amazondynamodb.zip` file to the location of your choice.
2. Open a command prompt window.
3. Navigate to the `<TDV_install_dir>/bin`
4. Enter the `server_util` command with the `-deploy` option:

```
server_util -server <hostname> [-port <port>] -user <user> -  
password <password> -deploy -package <TDV_install_  
dir>/adapters/tdv.amazondynamodb/tdv.amazondynamodb.jar
```

Note: When deploying a build of an existing adapter, you will need to undeploy the existing adapter using the `server_util` command with the `-undeploy` option.

```
server_util -server <hostname> [-port <port>] -user <user> -password  
<password> -undeploy -version 1 -name DynamoDB
```

## Basic Tab

### Connecting to DynamoDB

Specify the following to connect to data:

- Domain: Set this if you want to use a domain name you have associated with AWS.
- AWSRegion: Set this to the region where your Amazon DynamoDB data is hosted.

### Authenticating to DynamoDB

#### Obtain AWS Keys

To obtain the credentials for an IAM user, follow the steps below:

1. Sign into the IAM console.
2. In the navigation pane, select **Users**.
3. To create or manage the access keys for a user, select the user and then go to the **Security Credentials** tab.

To obtain the credentials for your AWS root account, follow the steps below:

1. Sign into the AWS Management console with the credentials for your root account.
2. Select your account name or number and select **My Security Credentials** in the menu that is displayed.
3. Click **Continue to Security Credentials** and expand the "Access Keys" section to manage or create root account access keys.

#### Root Credentials

To authenticate using account root credentials, set the following:

- AuthScheme: Set this to **AwsRootKeys**.
- AWSAccessKey: The access key associated with the AWS root account.
- AWSSecretKey: The secret key associated with the AWS root account.

**Note:** Use of this authentication scheme is discouraged by Amazon for anything but simple tests. The account root credentials have the full permissions of the user, making this the least secure authentication method.

## Temporary Credentials

To authenticate using temporary credentials, specify the following:

- AuthScheme: Set this to **TemporaryCredentials**.
- AWSAccessKey: The access key of the IAM user to assume the role for.
- AWSSecretKey: The secret key of the IAM user to assume the role for.
- AWSSessionToken: Your AWS session token. This will have been provided alongside your temporary credentials. See [AWS Identity and Access Management User Guide](#) for more info.

The adapter can now request resources using the same permissions provided by long-term credentials (such as IAM user credentials) for the lifespan of the temporary credentials.

If you are also using an IAM role to authenticate, you must additionally specify the following:

- AWSRoleARN: Specify the Role ARN for the role you'd like to authenticate with. This will cause the adapter to attempt to retrieve credentials for the specified role.
- AWSExternalId: Only if required when you assume a role in another account.

## EC2 Instances

If you are using the adapter from an EC2 Instance and have an IAM Role assigned to the instance, you can use the IAM Role to authenticate. To do so, set the following properties to authenticate:

- AuthScheme: Set this to **AwsEC2Roles**.

Do not specify AWSAccessKey and AWSSecretKey because the adapter will automatically obtain your IAM Role credentials and authenticate with them.

If you are also using an IAM role to authenticate, you must additionally specify the following:

- AWSRoleARN: Specify the Role ARN for the role you'd like to authenticate with. This will cause the adapter to attempt to retrieve credentials for the specified role.
- AWSExternalId: Only if required when you assume a role in another account.

## IMDSv2 Support

The Amazon DynamoDB adapter now supports IMDSv2. Unlike IMDSv1, the new version requires an authentication token. Endpoints and response are the same in both versions. In IMDSv2, the Amazon DynamoDB adapter first attempts to retrieve the IMDSv2 metadata token and then uses it to call AWS metadata endpoints. If it is unable to retrieve the token, the adapter reverts to IMDSv1.

## AWS IAM Roles

In many situations it may be preferable to use an IAM role for authentication instead of the direct security credentials of an AWS root user.

To authenticate as an AWS role, set the following:

- AuthScheme: Set this to **AwsIAMRoles**.
- AWSRoleARN: Specify the Role ARN for the role you'd like to authenticate with. This will cause the adapter to attempt to retrieve credentials for the specified role.
- AWSExternalId: Only if required when you assume a role in another account.
- AWSAccessKey: The access key of the IAM user to assume the role for.
- AWSSecretKey: The secret key of the IAM user to assume the role for.

**Note:** Roles may not be used when specifying the AWSAccessKey and AWSSecretKey of an AWS root user.

## ADFS

Set the AuthScheme to **ADFS**. The following connection properties need to be set:

- User: Set this to your ADFS username.
- Password: Set this to your ADFS password.

- SSOLoginURL: Set this to the login URL used by the SSO provider.

Below is an example connection string:

```
AuthScheme=ADFS; AWSRegion=Ireland; User=user@cdata.com;
Password=CH8WerW121235647iCa6; SSOLoginURL='https://adfs.domain.com';
AWSRoleArn=arn:aws:iam::1234:role/ADFS_SSO;
AWSPrincipalArn=arn:aws:iam::1234:saml-provider/ADFSPROVIDER;
S3StagingDirectory=s3://athena/staging;
```

## ADFS Integrated

To use the ADFS Integrated flow, specify the SSOLoginURL and leave the username and password empty.

## Okta

Set the AuthScheme to **Okta**. The following connection properties are used to authenticate through Okta:

- User: Set to your Okta user.
- Password: Set to your Okta password.
- SSOLoginURL: Set to the login URL used by the SSO provider.

If you are:

- using a trusted application or proxy that overrides the Okta client request
- configuring MFA

then you need to use combinations of SSOProperties input parameters to authenticate using Okta. Otherwise, you do not need to set any of these values.

In SSOProperties when required, set these input parameters:

- APIToken: When authenticating a user via a trusted application or proxy that overrides the Okta client request context, set this to the API Token the customer created from the Okta organization.
- MFAType: Set this if you have configured the MFA flow. Currently we support the following types: OktaVerify, Email, and SMS.
- MFAPassCode: Set this only if you have configured the MFA flow. If you set this to empty or an invalid value, the adapter issues a one-time password challenge to your



device or email. After the passcode is received, reopen the connection where the retrieved one-time password value is set to the MFAPassCode connection property.

- **MFARememberDevice**: Okta supports remembering devices when MFA is required. If remembering devices is allowed according to the configured authentication policies, the adapter sends a device token to extend MFA authentication lifetime. This property is, by default, set to **True**. Set this to **False** only if you do not want MFA to be remembered.

Example connection string:

```
AuthScheme=Okta; AWSRegion=Ireland; User=user@cdata.com;
Password=CH8WerW121235647iCa6; SSOLoginURL='https://cdata-
us.okta.com/home/amazon_aws/0oa35m8arsAL5f5NrE6NdA356/272';
SSOProperties='ApiToken=01230GGG2ceAnm_tPAf4MhiMELXZ0L0N1pAYr01VR-
hGQSf;'; AWSRoleArn=arn:aws:iam::1234:role/Okta_SSO;
AWSPrincipalARN=arn:aws:iam::1234:saml-provider/OktaProvider;
S3StagingDirectory=s3://athena/staging;
```

## PingFederate

Set the AuthScheme to **PingFederate**. The following connection properties need to be set:

- User: Set this to the PingFederate user.
- Password: Set this to PingFederate password for the user.
- SSOLoginURL: Set this to the login url used by the SSO provider.
- SSOExchangeUrl: The **Partner Service Identifier** URI configured in your PingFederate server instance under: **SP Connections > SP Connection > WS-Trust > Protocol Settings**. This should uniquely identify a PingFederate SP Connection, so it is a good idea to set it to your **AWS SSO ACS URL**. You can find it under **AWS SSO > Settings > View Details** next to the **Authentication** field.

The following SSOProperties are needed to authenticate to PingFederate:

- **AuthScheme** (optional): The authorization scheme to be used for the IdP endpoint. The allowed values for this IdP are None or Basic.

Additionally, you can use the following SSOProperties to configure mutual SSL authentication for SSOLoginURL, the WS-Trust STS endpoint:

- SSLClientCert
- SSLClientCertType

- SSLClientCertSubject
- SSLClientCertPassword

Below is an example connection string:

```
authScheme=pingfederate;SSOLoginURL=https://mycustomserver.com:9033/idp/
sts.wst;SSOExchangeUrl=https://us-east-
1.signin.aws.amazon.com/platform/saml/acs/764ef411-
xxxxxx;user=admin;password=PassValue;AWSPrincipalARN=arn:aws:iam::215338
515180:saml-
provider/pingFederate;AWSRoleArn=arn:aws:iam::215338515180:role/SSOTest
2;
```

## MFA

For users and roles that require Multi-factor Authentication, specify the following to authenticate:

- AuthScheme: Set this to **AwsMFA**.
- CredentialsLocation: The location of the settings file where MFA credentials are saved. See the Credentials File Location page under Connection String Options for more information.
- MFASerialNumber: The serial number of the MFA device if one is being used.
- MFAToken: The temporary token available from your MFA device.

If you are connecting to AWS (instead of already being connected such as on an EC2 instance), you must additionally specify the following:

- AWSAccessKey: The access key of the IAM user for whom MFA will be issued.
- AWSSecretKey: The secret key of the IAM user whom MFA will be issued.

If you are also using an IAM role to authenticate, you must additionally specify the following:

- AWSRoleARN: Specify the Role ARN for the role you'd like to authenticate with. This will cause the adapter to attempt to retrieve credentials for the specified role using MFA.
- AWSExternalId: Only if required when you assume a role in another account.

This causes the adapter to submit the MFA credentials in a request to retrieve temporary authentication credentials.

Note that you can control the duration of the temporary credentials by setting the TemporaryTokenDuration property (default 3600 seconds).

## Credentials Files

You can use a credentials file to authenticate. Any configurations related to AccessKey/SecretKey authentication, temporary credentials, role authentication, or MFA can be used. To do so, set the following properties to authenticate:

- AuthScheme: Set this to **AwsCredentialsFile**.
- AWSCredentialsFile: Set this to the location of your credentials file.
- AWSCredentialsFileProfile (optional): Optionally set this to the name of the profile you would like to use from the specified credentials file. If not specified, the profile with the name *default* will be used.

See [AWS Command Line Interface User Guide](#) for more information.

## AWS Cognito Credentials

If you want to use the adapter with a user registered in a User Pool in AWS Cognito, set the following properties to authenticate:

- AuthScheme: Set this to **AwsCognitoSrp** (recommended). You can also use *AwsCognitoBasic*.
- AWSCognitoRegion: Set this to the region of the User Pool.
- AWSUserPoolId: Set this to the User Pool Id.
- AWSUserPoolClientId: Set this to the User Pool Client App Id.
- AWSUserPoolClientAppSecret: Set this to the User Pool Client Secret.
- AWSIdentityPoolId: Set this to the Identity Pool Id of the Identity Pool that is linked with the User Pool.
- User: Set this to the username of the user registered in the User Pool.
- Password: Set this to the password of the user registered in the User Pool.

## Logging

The adapter uses TDV Server's logging (log4j) to generate log files. The settings within the TDV Server's logging (log4j) configuration file are used by the adapter to determine the type of messages to log. The following categories can be specified:

- Error: Only error messages are logged.
- Info: Both Error and Info messages are logged.
- Debug: Error, Info, and Debug messages are logged.

The Other property of the adapter can be used to set Verbosity to specify the amount of detail to be included in the log file, that is:

```
Verbosity=4;
```

You can use Verbosity to specify the amount of detail to include in the log within a category. The following verbosity levels are mapped to the log4j categories:

- 0 = Error
- 1-2 = Info
- 3-5 = Debug

For example, if the log4j category is set to DEBUG, the Verbosity option can be set to 3 for the minimum amount of debug information or 5 for the maximum amount of debug information.

Note that the log4j settings override the Verbosity level specified. The adapter never logs at a Verbosity level greater than what is configured in the log4j properties. In addition, if Verbosity is set to a level less than the log4j category configured, Verbosity defaults to the minimum value for that particular category. For example, if Verbosity is set to a value less than 3 and the Debug category is specified, the Verbosity defaults to 3.

The following list is an explanation of the Verbosity levels and the information that they log.

- 1 - Will log the query, the number of rows returned by it, the start of execution and the time taken, and any errors.
- 2 - Will log everything included in Verbosity 1 and HTTP headers.
- 3 - Will additionally log the body of the HTTP requests.
- 4 - Will additionally log transport-level communication with the data source. This includes SSL negotiation.

- 5 - Will additionally log communication with the data source and additional details that may be helpful in troubleshooting problems. This includes interface commands.

## Configure Logging for the Amazon DynamoDB Adapter

By default, logging is turned on without debugging. If debugging information is desired, uncomment the following line in the TDV Server's log4j.properties file (default location of this file is: C:\Program Files\TIBCO\TDV Server <version>\conf\server):

```
log4j.logger.com.cdata=DEBUG
```

The TDV Server must be restarted after changing the log4j.properties file, which can be accomplished by running the composite.bat script located at: C:\Program Files\TIBCO\TDV Server <version>\bin. Note that reauthenticating to the TDV Studio is required after restarting the server.

Here is an example of the calls:

```
.\composite.bat monitor restart
```

All logs for the adapter are written to the "cs\_server\_dsrc.log" file as specified in the log4j properties.

**Note:** The "log4j.logger.com.cdata=DEBUG" option is not required if the **Debug Output Enabled** option is set to true within the TDV Studio. To set this option, navigate to **Administrator > Configuration**. Select **Server > Configuration > Debugging** and set the Debug Output Enabled option to **True**.

## Fine-Tuning Data Access

### Inferring the Data Type

You can use the following properties to configure automatic data type detection, which is enabled by default.

- TypeDetectionScheme: You can use this property to enable or disable automatic type detection based on the value specified in RowScanDepth.
- RowScanDepth: This property determines the number of rows that will be scanned to determine column data types.

- **IgnoreTypes**: The data types that should be ignored and resolve to varchar data types. By default, Date, Time, and Datetime types are ignored. This is because Amazon DynamoDB does not support them as types. Any filtering of these columns may be done only as their original varchar data type.

## Fine Tuning Data Access

You can use the following properties to gain greater control over Amazon DynamoDB API features and the strategies the adapter uses to surface them:

- **GenerateSchemaFiles**: This property enables you to persist table metadata in static schema files that are easy to customize, to change column data types, for example. You can set this property to "OnStart" to generate schema files for all tables in your database at connection. Or, you can generate schemas as you execute SELECT queries to tables. The resulting schemas are based on the connection properties you use to configure [Automatic Schema Discovery](#).

- **UseSimpleNames**: Amazon DynamoDB supports attribute names with special characters that many database-oriented tools do not support.

In addition, Amazon DynamoDB table names can include dots and dashes -- the adapter interprets dots within table names as hierarchy separators that enable you to drill down to nested fields, similar to XPath.

You can use this property to replace any nonalphanumeric character with an underscore.

- **SeparatorCharacter**: You can use this property to more easily access nested fields when [Querying Documents and Lists](#); specify the hierarchy separator with this property. By default, this character is the '.' (dot) character.

## Performance

### Setting a Retry Interval

You can set the following properties to retry queries instead of returning a temporary error such as "maximum throughput exceeded":

- **RetryWaitTime**: The minimum number of milliseconds the adapter will wait to retry a request.
- **MaximumRequestRetries**: The maximum number of times to retry a request.

The Amazon DynamoDB Adapter also has two separate APIs that may be used depending on the query, PartiQL and Scan. The API that is used depends on the query that is executed.

## PartiQL

PartiQL is used on any insert/update/delete request query, as well as any select that contains a filter. This is due to the PartiQL API containing more advanced filtering capabilities than the older Scan endpoint. In general, queries where a significant portion of the result is filtered out can be expected to execute faster than a query with very little filtered.

## Using Paging Effectively

You can use the Pagesize property to optimize use of your provisioned throughput, based on the size of your items and Amazon DynamoDB's 1MB page size. Set this property to the number of items to return.

Generally, a smaller page size reduces spikes in throughput that cause throttling. A smaller page size also inserts pauses between requests. This interval evens out the distribution of requests and allows more requests to be successful by avoiding throttling.

## Scans

A Scan will occur during a SELECT query that contains no filter. In this case, all results must be retrieved, so there is no advantage in using the PartiQL API. Executing a Scan will retrieve all results, but the API contains a key feature that gives it better performance than an unfiltered PartiQL query: multiple threads.

The ThreadCount connection property may be set to influence how many threads will be used when executing a Scan request. Using more threads will cause more memory to be taken up, but will result in faster results per thread. The default is 4. This works best on tables where a high or variable throughput is provisioned.

In cases where the maximum throughput for a table would be exceeded on a single thread, there is no benefit to using a Scan over the single threaded PartiQL API. The Amazon DynamoDB will simply throttle all threads until the maximum throughput is no longer exceeded.

## Minimum IAM Requirements

We recommend using predefined roles for services rather than creating custom IAM policies. Predefined roles for Amazon DynamoDB are

- `AmazonDynamoDBReadOnlyAccess`-grants read-only access to DynamoDB resources through the AWS Management Console.
- `AmazonDynamoDBFullAccess`-grants full access to DynamoDB resources through the AWS Management Console.

If you want to create custom policies, use the roles described in the table below. Note that the specific policies required by the Amazon DynamoDB driver are subject to change in future releases. Amazon DynamoDB requires at a minimum the following permissions:

IAM Role	Description
<code>dynamodb:ListTables</code>	<p>Required for getting a list of your DynamoDB tables. Used during metadata retrieval to dynamically determine the list of your tables. Note that this action does not support resource-level permissions and requires you to choose <i>All resources</i> (hence the * for "Resource"). In other words, the action <code>dynamodb:ListTables</code> needs a * Resource, and the other actions can be given permission to all the tables <code>arn:aws:dynamodb:us-east-1:987654321098:table/*</code> or to a list of specific tables:</p> <pre data-bbox="641 1167 1250 1367"> "Resource": [     "arn:aws:dynamodb:us-east-1:987654321098:table/Customers",     "arn:aws:dynamodb:us-east-1:987654321098:table/Orders" ]</pre>
<code>dynamodb:DescribeTable</code>	<p>Required for getting metadata about the selected table. Used during table metadata retrieval to dynamically determine the list of the columns. This action supports resource-level permissions, so you can specify the tables you want to get the metadata from. For example, for the table <code>Customers</code> and <code>Orders</code> in the region <code>Northern Virginia us-east-1</code>, for account <code>987654321098</code>:</p> <pre data-bbox="732 1766 748 1797">{</pre>



```

    "Effect": "Allow",
    "Action": [
        "dynamodb:DescribeTable"
    ],
    "Resource": [
        "arn:aws:dynamodb:us-east-1:987654321098:table/Customers",
        "arn:aws:dynamodb:us-east-1:987654321098:table/Orders"
    ]
}

```

To give permissions to all the tables in the region you specified in the connection property AWSRegion, use an `*` instead of the table name:

```

"Resource": "arn:aws:dynamodb:us-east-1:987654321098:table/*"

```

dynamodb:Scan	Required for getting one or more items by accessing every item in the table. Used for most of the SELECT queries, for example, <i>SELECT * FROM [Customers]</i> . This action supports resource-level permissions, so you can specify the tables you want to get data from, similar to dynamodb:DescribeTable.
dynamodb:PartiQLSelect	Required for getting specific items from a table when using SELECT queries and filtering by the primary key column, for example, <i>SELECT * FROM [Customers] WHERE id=1234</i> . This action supports resource-level permissions, so you can specify the tables you want to get data from, similar to dynamodb:DescribeTable.
dynamodb:PartiQLInsert	Required for inserting data to a table. This action supports resource-level permissions, so you can specify the tables you want to insert data to, similar to dynamodb:DescribeTable.
dynamodb:PartiQLUpdate	Required for modifying data in a table. This action supports resource-level permissions, so you can specify

	the tables you want to modify data on, similar to dynamodb:DescribeTable.
dynamodb:PartiQLDelete	Required for deleting data from a table. This action supports resource-level permissions, so you can specify the tables you want to delete data from, similar to dynamodb:DescribeTable.
dynamodb:CreateTable	Required for creating a table. This action supports resource-level permissions, so you can specify the table names you can create.

## Changelog

### General Changes

Date	Build Number	Change Type	Description
01/18/2023	8418	Amazon DynamoDB	<b>Added</b> <ul style="list-style-type: none"> <li>Added support for document() function.</li> </ul>
12/14/2022	8383	General	<b>Changed</b> <ul style="list-style-type: none"> <li>Added the Default column to the sys_procedureparameters table.</li> </ul>
09/30/2022	8308	General	<b>Changed</b> <ul style="list-style-type: none"> <li>Added the IsPath column to the sys_procedureparameters table.</li> </ul>
09/22/2022	8300	Amazon DynamoDB	<b>Added</b> <ul style="list-style-type: none"> <li>Added the FileStream output for the CreateSchema stored procedure. An</li> </ul>

			instance of an output stream where file data is written to. Only used if FileName is not set.
08/19/2022	8266	Amazon DynamoDB	<b>Added</b> <ul style="list-style-type: none"> <li>Added the FileData output for the CreateSchema stored procedure. It pushes the generated schema encoded in Base64 only if the FileName input is not set.</li> </ul>
08/17/2022	8264	General	<b>Changed</b> <ul style="list-style-type: none"> <li>We now support handling the keyword "COLLATE" as standard function name as well.</li> </ul>
03/16/2022	8110	Amazon DynamoDB	<b>Added</b> <ul style="list-style-type: none"> <li>Added support for Instance Metadata Service Version 2 (IMDSv2) when authenticating from an Amazon EC2 instance.</li> </ul>
03/02/2022	8096	Amazon DynamoDB	<b>Added</b> <ul style="list-style-type: none"> <li>Added support for the AutoDetectIndex connection property.</li> </ul> <b>Changed</b> <ul style="list-style-type: none"> <li>In DynamoDB, secondary indexes can be used to more quickly select data from a given table. By default, we now attempt to automatically detect an index to use based on the query criteria. To turn off index detecting logic, simply set the AutoDetectIndex connection property to false. Alternatively, you may use the SecondaryIndexName pseudo column to specify which index to use (if any).</li> </ul>

03/01/2022	8095	Amazon DynamoDB	<b>Added</b> <ul style="list-style-type: none"> <li>• Add support for the AwsCognitoBasic and AwsCognitoSrp authentication methods.</li> </ul>
09/08/2021	7923	Amazon DynamoDB	<b>Added</b> <ul style="list-style-type: none"> <li>• Added support for PartiQL statements. Driver now utilizes the ExecuteStatement API operation to execute PartiQL statements, which offers better support for executing filters on the query server side.</li> <li>• Added support for BULK UPDATE operations.</li> </ul> <b>Changed</b> <ul style="list-style-type: none"> <li>• The DynamoDB JSON document parser is re-written to fully support all the levels of structure nesting and indexing or un-nesting at all levels.</li> <li>• Data type auto discovery mechanism to properly detect datetime fields.</li> <li>• Fixed vertical flattening on primitive arrays.</li> </ul>
09/02/2021	7915	General	<b>Added</b> <ul style="list-style-type: none"> <li>• Added support for the STRING_SPLIT table-valued function in the CROSS APPLY clause.</li> </ul>
08/07/2021	7889	General	<b>Changed</b> <ul style="list-style-type: none"> <li>• Added the KeySeq column to the sys_foreignkeys table.</li> </ul>
08/06/2021	7888	General	<b>Changed</b>

			<ul style="list-style-type: none"> <li>Added the new sys_primarykeys system table.</li> </ul>
07/23/2021	7874	General	<p><b>Changed</b></p> <ul style="list-style-type: none"> <li>Updated the Literal Function Names for relative date/datetime functions. Previously relative date/datetime functions resolved to a different value when used in the projection vs te predicate. Ie: SELECT LAST_MONTH() AS lm, Col FROM Table WHERE Col &gt; LAST_MONTH(). Formerly the two LAST_MONTH() methods would resolve to different datetimes. Now they will match.</li> <li>As a replacement for the previous behavior, the relative date/datetime functions in the criteria may have an 'L' appended to them. Ie: WHERE col &gt; L_LAST_MONTH(). This will continue to resolve to the same values that previously were calculated in the criteria. Note that the "L_" prefix will only work in the predicate - it not available for the projection.</li> </ul>
07/08/2021	7859	General	<p><b>Added</b></p> <ul style="list-style-type: none"> <li>Added the TCP Logging Module for the logging information happening on the TCP wire protocol. The transport bytes that are incoming and ongoing will be logged at verbosity=5.</li> </ul>
05/17/2021	7807	Amazon DynamoDB	<p><b>Added</b></p> <ul style="list-style-type: none"> <li>Added support for Single Sign On (SSO) connection with PingFederate as the Identity Provider (IdP).</li> </ul>

04/23/2021	7785	General	<b>Added</b> <ul style="list-style-type: none"> <li>Added support for handling client side formulas during insert / update. For example: UPDATE Table SET Col1 = Concat(Col1, " - ", Col2) WHERE Col2 LIKE 'A%'</li> </ul>
04/23/2021	7783	General	<b>Changed</b> <ul style="list-style-type: none"> <li>Updated how display sizes are determined for varchar primary key and foreign key columns so they will match the reported length of the column.</li> </ul>
04/16/2021	7776	General	<b>Added</b> <ul style="list-style-type: none"> <li>Non-conditional updates between two columns is now available to all drivers. For example: UPDATE Table SET Col1=Col2</li> </ul> <b>Changed</b> <ul style="list-style-type: none"> <li>Reduced the length to 255 for varchar primary key and foreign key columns.</li> <li>Updated implicit and metadata caching to improve performance and support for multiple connections. Old metadata caches are not compatible - you would need to generate new metadata caches if you are currently using CacheMetadata.</li> <li>Updated index naming convention to avoid duplicates</li> <li>Updated and standardized Getting Started connection help.</li> <li>Added the Advanced Features section to the help of all drivers.</li> <li>Categorized connection property listings</li> </ul>

---

in the help for all editions.

---

04/15 /2021

7775

General

**Changed**

- Kerberos authentication is updated to use TCP by default, but will fall back to UDP if a TCP connection cannot be established
- 

## NoSQL Database

Amazon DynamoDB is a schemaless database that provides high performance, availability, and scalability. These features are not necessarily incompatible with a standards-compliant query language like SQL-92. In this section we will show various schemes that the adapter offers to bridge the gap with relational SQL and a document database.

The adapter models the schemaless Amazon DynamoDB tables into relational tables and translates SQL queries into Amazon DynamoDB queries to get the requested data. The adapter offers two ways, [Automatic Schema Discovery](#) and [Custom Schema Definitions](#), to model Amazon DynamoDB tables as relational tables.

The [Automatic Schema Discovery](#) scheme automatically finds the data types in a Amazon DynamoDB table by scanning a configured number of rows of the table. You can use [RowScanDepth](#), [FlattenArrays](#), and [FlattenObjects](#) to control the relational representation of the tables in Amazon DynamoDB.

Optionally, you can use [Custom Schema Definitions](#) to project your chosen relational structure on top of a Amazon DynamoDB table. This allows you to define your chosen column names, their data types, and the location of their values in the Amazon DynamoDB table.

## Automatic Schema Discovery

The adapter automatically infers a relational schema by inspecting a series of Amazon DynamoDB documents in a collection. You can use the [RowScanDepth](#) property to define the number of documents the adapter will scan to do so. The columns identified during the discovery process depend on the [FlattenArrays](#) and [FlattenObjects](#) properties.

## Flattening Objects

If `FlattenObjects` is set, all nested objects will be flattened into a series of columns. For example, consider the following document:

```
{
  id: 12,
  name: "Lohia Manufacturers Inc.",
  address: {street: "Main Street", city: "Chapel Hill", state: "NC"},
  offices: ["Chapel Hill", "London", "New York"],
  annual_revenue: 35,600,000
}
```

This document will be represented by the following columns:

Column Name	Data Type	Example Value
id	Integer	12
name	String	Lohia Manufacturers Inc.
address.street	String	Main Street
address.city	String	Chapel Hill
address.state	String	NC
offices	String	["Chapel Hill", "London", "New York"]
annual_revenue	Double	35,600,000

If `FlattenObjects` is not set, then the `address.street`, `address.city`, and `address.state` columns will not be broken apart. The `address` column of type string will instead represent the entire object. Its value would be `{street: "Main Street", city: "Chapel Hill", state: "NC"}`. See [JSON Functions](#) for more details on working with JSON aggregates.

You can change the separator character in the column name from a dot by setting `SeparatorCharacter`.



## Flattening Arrays

The `FlattenArrays` property can be used to flatten array values into columns of their own. This is only recommended for arrays that are expected to be short, for example the coordinates below:

```
"coord": [ -73.856077, 40.848447 ]
```

The `FlattenArrays` property can be set to 2 to represent the array above as follows:

Column Name	Data Type	Example Value
coord.0	Float	-73.856077
coord.1	Float	40.848447

It is best to leave other unbounded arrays as they are and piece out the data for them as needed using [JSON Functions](#).

## Vertical Flattening

It is possible to retrieve an array of objects as if it were a separate table. Take the following JSON structure from the restaurants table for example:

```
{
  "restaurantid" : "30075445",
  "address" : {
    "building" : "1007",
    "coord" : "-73.856077, 40.848447",
    "street" : "Morris Park Ave",
    "zipcode" : "10462"
  },
  "borough" : "Bronx",
  "cuisine" : "Bakery",
  "grades" : "{
    \"date\" : 1393804800000,
    \"grade\" : \"B\",
    \"score\" : 2
  }, {
    \"date\" : 1378857600000,
    \"grade\" : \"A\",
    \"score\" : 6
  }"
```

```

    }, {
      "date" : 1358985600000,
      "grade" : "A",
      "score" : 10
    }",
    "name" : "Morris Park Bake Shop"
  }

```

Vertical flattening will allow you to retrieve the grades array as a separate table by using the syntax below:

```
SELECT * FROM "restaurants.grades"
```

This query returns the following data set:

<b>date</b>	<b>grade</b>	<b>score</b>	<b>_index</b>
1393804800000	B	2	1
1378857600000	A	6	2
1358985600000	A	10	3

The grades array could also be nested some levels deeper. In that case, the same syntax should be used:

```
SELECT * FROM "restaurants.cuisine.bakery.grades"
```

There are also cases where the nested structure includes another array in a higher level. Take the following JSON as an example:

```

{
  "restaurantid" : "30075445",
  "reviews": "
    {
      "grades": "
        {
          "date": 1393804800000,
          "score": 2,
          "grade": "B"
        },
        {

```

```

    "date": 1378857600000,
    "score": 6,
    "grade": "A"
  },
  {
    "date": 1358985600000,
    "score": 10,
    "grade": "A"
  }
],
"name" : "Morris Park Bake Shop"
}

```

For this structure, the index of the reviews array will need to get wrapped in square brackets. If they are already being used as escape characters in the SQL query, the square brackets will need to be escaped themselves as shown in the query below:

```
SELECT * FROM "restaurants.reviews.\"0\".grades"
```

This query will return the same data set as the JSON structure at the top. Note that this syntax is case sensitive, so make sure to write the field names the same way that they're saved in DynamoDB.

## JSON Functions

The adapter can return JSON structures as column values. The adapter enables you to use standard SQL functions to work with these JSON structures. The examples in this section use the following array:

```

[
  { "grade": "A", "score": 2 },
  { "grade": "A", "score": 6 },
  { "grade": "A", "score": 10 },
  { "grade": "A", "score": 9 },
  { "grade": "B", "score": 14 }
]

```

### JSON\_EXTRACT

The `JSON_EXTRACT` function can extract individual values from a JSON object. The following query returns the values shown below based on the JSON path passed as the second argument to the function:

```
SELECT Name, JSON_EXTRACT(grades,'[0].grade') AS Grade, JSON_EXTRACT(grades,'[0].score') AS Score FROM Students;
```

Column Name	Example Value
Grade	A
Score	2

## JSON\_COUNT

The JSON\_COUNT function returns the number of elements in a JSON array within a JSON object. The following query returns the number of elements specified by the JSON path passed as the second argument to the function:

```
SELECT Name, JSON_COUNT(grades,'[x]') AS NumberOfGrades FROM Students;
```

Column Name	Example Value
NumberOfGrades	5

## JSON\_SUM

The JSON\_SUM function returns the sum of the numeric values of a JSON array within a JSON object. The following query returns the total of the values specified by the JSON path passed as the second argument to the function:

```
SELECT Name, JSON_SUM(score,'[x].score') AS TotalScore FROM Students;
```

Column Name	Example Value
TotalScore	41

## JSON\_MIN

The JSON\_MIN function returns the lowest numeric value of a JSON array within a JSON object. The following query returns the minimum value specified by the JSON path passed as the second argument to the function:

```
SELECT Name, JSON_MIN(score, '[x].score') AS LowestScore FROM Students;
```

Column Name	Example Value
LowestScore	2

## JSON\_MAX

The JSON\_MAX function returns the highest numeric value of a JSON array within a JSON object. The following query returns the maximum value specified by the JSON path passed as the second argument to the function:

```
SELECT Name, JSON_MAX(score, '[x].score') AS HighestScore FROM Students;
```

Column Name	Example Value
HighestScore	14

# DynamoDB Queries

Because Amazon DynamoDB is a NoSQL data source, queries need to be handled a bit differently than standard relational databases.

## Value-Sensitive Queries

The lack of a required data type for a given column means that you could store different types of data in a single column. For instance, one row could have a String called EmailAddresses and another could have a StringSet also called EmailAddresses. For these and other kinds of cases, the adapter largely determines what data type to use based on the values in the query.

For instance, say you have an Items table where the PartNumber could store either a String or a Number. To get back a part with the PartNumber of the number value 12345, you would issue the following query:

```
SELECT Name, Location, Quantity, PartNumber FROM Items WHERE PartNumber = 12345
```

Alternatively, the PartNumber could have been stored as the string "12345". To get back a part with the PartNumber of the literal string 12345, issue the following query:

```
SELECT Name, Location, Quantity, PartNumber FROM Items WHERE PartNumber = '12345'
```

If the data type of the specified value is not ambiguous, it is always used before the autodetected data type. In both of these cases if a parameter was used instead of a hardcoded value, then the data type of the parameter would be used to determine what type to submit to Amazon DynamoDB.

## Detected Column Data Type

If a value is not obvious based purely on the detected data type, the adapter compares it to the autodetected column. For instance, if you want to insert a column called Coordinates into the Location table, your insert would look like:

```
INSERT INTO Locations (Address, Coordinates) VALUES ('123 Fake Street', '[40.7127, 74.0059]')
```

Based on the input value alone, the detected data type is a string. However, because a Coordinates column was previously autodetected, the adapter inserts a NumberSet and not a simple String.

If a Coordinates column was not autodetected when scanning the Locations table, the data type of the inserted value is used.

In this case, we could still resolve that the insert is a NumberSet, but it will cost a bit more overhead to do this.

## Count

Amazon DynamoDB supports 2 different methods of using the COUNT aggregate function. To simply return the number of Items in your table, issue the following query:

```
SELECT COUNT(*) FROM MyTable
```

The Amazon DynamoDB Adapter will read the ItemCount from the DescribeTable Action. This avoids using too many read units to scan the full table. However, DynamoDB updates this value approximately every six hours and recent changes might not be reflected in this value.

Issuing the below example queries will instead scan the full table for count:

```
SELECT COUNT(*) FROM MyTable WHERE MyInt > 10
SELECT COUNT(MyInt) FROM MyTable
```

## Querying Documents and Lists

Amazon DynamoDB documents and lists are supported with the Amazon DynamoDB Adapter. You can access documents and lists directly at the root level or use the '.' character as a hierarchy divider to drill down to documents and lists.

### Reporting Values in Documents and Lists

When data types are autodetected, they are reported down to the lowest level that can be reliably detected. For instance, a document called Customer with a child called Address and a child on Address called Street would be represented by the column Customer.Address.Street.

However, this process does not apply to Lists since a list could have any number of entries. Once a List or a Set is detected, additional values are not reported as being available in the table schema.

### Getting Back Unreported Values

If there are attributes that frequently do not have a value and thus are not autodetected, these can still be retrieved by specifying the correct path to them. For instance, to get the Special attribute from the Customer document:

```
SELECT [Customer.Address.Street], [Customer.Special] FROM MyTable
```

Once a List has been detected, additional values are not reported. But individual values on the list can be referenced by specifying '.' and a number. For instance:

```
SELECT [MyList.0], [MyList.1.Email], [MyList.1.Age] FROM MyTable
```

This will retrieve the first value on the list and the second value's Email and Age attributes.

## Inserting Documents and Lists

Inserts in Amazon DynamoDB require that the full object is specified during insert. Insert a document or list at the root. Pass in the full JSON aggregate. For instance:

```
INSERT INTO MyTable (PrimaryKey, EmailAddresses, Address, MyList) VALUES
('uniquekey', '["user@email.com", "user2@email2.com"]', '{"Street":"123
Fake Street", "City":"Chapel Hill", "Zip":"27713"}', '[{"S":"somestr"},
{"NS":[1,2]},{ "N":4}]')
```

In this case, the EmailAddress is inserted as a StringSet, Address is inserted as a document, and MyList is inserted as a list.

## Updating Documents and Lists

Updates are supported using the same syntax that is available during selects. Documents and Lists can be specified using the '.' character to specify hierarchy. For instance:

```
UPDATE MyTable SET [EmailAddress.0]='user@email.com',
[EmailAddress.1]='user2@email2.com', [Address.Street]='123 Fake Street',
[Address.City]='Chapel Hill', [Address.Zip]='27713',
[MyList.0]='somestr', [MyList.1]='[1,2]', [MyList.2]=4 WHERE
PrimaryKey='uniquekey'
```

Note that EmailAddress and MyList must be autodetected to resolve how to handle EmailAddress differently from MyList. If you are in doubt about whether or not something will be automatically detected, specifying the full JSON to update will always work.

# Data Type Mapping

## Data Type Mappings

The adapter maps types from the data source to the corresponding data type available in the schema. Additionally, we will attempt to scan the available data coming back based on the [IgnoreTypes](#) connection property. The table below documents these mappings.



Amazon DynamoDB	CData Schema
String	string, date, datetime, time
Binary	string
Number	bigint, int, float (depending on data that is detected)
StringSet	string
NumberSet	string
BinarySet	string
Map	string
List	string
Boolean	bool
Null	string

Note that depending on the settings of [IgnoreTypes](#), some of these types may not be detected by default. Date, datetime, and time for example are ignored by default as they cannot be filtered server side, and may be inserted / updated in a different format than your existing entries if enabled. Please use caution when enabling them.

[FlattenArrays](#) and [FlattenObjects](#) may also be used to to flatten the StringSets, NumberSets, BinarySets, Maps, and Lists into individual columns.

## Custom Schema Definitions

In addition to [Automatic Schema Discovery](#) the adapter also allows you to statically define the schema for your Amazon DynamoDB table. Let's consider a schema for the restaurants data set.

Below is an example item from the table:

```
{
  "address":{
    "building":"461",
    "coord":[
      -74.138492,
      40.631136
    ],
    "street":"Port Richmond Ave",
    "zipcode":"10302"
  },
  "borough":"Staten Island",
  "cuisine":"Other",
  "grades":[
  ],
  "name":"Indian Oven",
  "restaurant_id":"50018994"
}
```

## Defining a Custom Schema

You can define a custom schema to extract out nested properties as their own columns. Set the Location property to the file directory that will contain the schema file.

The following schema uses the `other:path` property to define where the data for a particular column should be retrieved from. Using this model you can flatten arbitrary levels of hierarchy.

The `'other:tablename'` attribute specifies the table to parse. This attribute gives you the flexibility to use multiple schemas for the same table.

In [Custom Schema Example](#), you will find the complete schema that contains the example above.

```
<api:info title="StaticRestaurants" other:catalog="CData"
other:schema="AmazonDynamoDB" description="StaticRestaurants"
other:tablename="StaticRestaurants" other:version="20">
  <attr name="id" xs:type="decimal" key="true"
columnsize="17" precision="38" scale="6" readonly="false"
description="Dynamic Column." other:dynamodatatype="N"
other:relativepath="restaurant_id" other:filterable="true"
other:fullpath="restaurant_id" other:tablename="&quot;restaurant_
id&quot;" />
  <attr name="borough" xs:type="string"
columnsize="2000" readonly="false"
description="Dynamic Column." other:dynamodatatype="S"
```

```

other:relativepath="borough"          other:filterable="true"
other:fullpath="borough"
other:apiname="&quot;borough&quot;"
/>
<attr  name="address_zipcode"    xs:type="int"
columnsize="4"      precision="10"      readonly="false"
description="Dynamic Column."  other:dynamodatatype="S"
other:relativepath="zipcode"      other:filterable="true"
other:fullpath="address.zipcode"
other:apiname="&quot;address&quot;.&quot;zipcode&quot;"
/>
<attr  name="address_coord_0"    xs:type="double"
columnsize="8"      precision="15"      readonly="false"
description="Dynamic Column."  other:dynamodatatype="N"
other:relativepath="coord"      other:filterable="true"
other:fullpath="address.coord[0]"
other:apiname="&quot;address&quot;.&quot;coord&quot;[0]"
/>
<attr  name="address_coord_1"    xs:type="double"
columnsize="8"      precision="15"      readonly="false"
description="Dynamic Column."  other:dynamodatatype="N"
other:relativepath="coord[1]"    other:filterable="true"
other:fullpath="address.coord[1]"
other:apiname="&quot;address&quot;.&quot;coord&quot;[1]"
/>
<attr  name="address_building"   xs:type="int"
columnsize="4"      precision="10"      readonly="false"
description="Dynamic Column."  other:dynamodatatype="S"
other:relativepath="building"    other:filterable="true"
other:fullpath="address.building"
other:apiname="&quot;address&quot;.&quot;building&quot;"
/>
<attr  name="address_street"     xs:type="string"
columnsize="2000"      readonly="false"
description="Dynamic Column."  other:dynamodatatype="S"
other:relativepath="street"      other:filterable="true"
other:fullpath="address.street"
other:apiname="&quot;address&quot;.&quot;street&quot;"
/>
<attr  name="name"              xs:type="string"
columnsize="2000"      readonly="false"
description="Dynamic Column."  other:dynamodatatype="S"
other:relativepath="name"        other:filterable="true"
other:fullpath="name"
other:apiname="&quot;name&quot;"
/>
<attr  name="cuisine"           xs:type="string"

```

```

columnsize="2000"                                readonly="false"
description="Dynamic Column."    other:dynamodatatype="S"
other:relativepath="cuisine"    other:filterable="true"
other:fullpath="cuisine"
other:apiname="&quot;cuisine&quot;"
  />
</api:info>

```

## Custom Schema Example

This section contains a complete schema. The info section enables a relational view of a Amazon DynamoDB table. For more details, see [Custom Schema Definitions](#). The table below allows the SELECT, INSERT, UPDATE, and DELETE commands as implemented in the GET, POST, MERGE, and DELETE sections of the schema below. Set the Location property to the file directory that will contain the schema file.

Use the 'other:tableapiname' attribute to specify the name of the Amazon DynamoDB table you want to parse. You can use the 'other:tableapiname' attribute to define multiple schemas for the same table. *Note:* Amazon DynamoDB is case sensitive. Your table name and specified paths must match the case of how your fields appear in Amazon DynamoDB.

The operations, such as dynamodbadoProviderOperationCaller, are internal implementations and can also be copied as is.

```

<api:script xmlns:api="http://apiscript.com/ns?v1"
  xmlns:xs="http://www.cdata.com/ns/rsbscript/2"
  xmlns:other="http://apiscript.com/ns?v1">
  <api:info title="StaticRestaurants" other:catalog="CData"
    other:schema="AmazonDynamoDB" description="StaticRestaurants"
    other:tableapiname="StaticRestaurants" other:version="20">
    <attr name="id" xs:type="decimal" key="true"
      columnsize="17" precision="38" scale="6" readonly="false"
      description="Dynamic Column." other:dynamodatatype="N"
      other:relativepath="restaurant_id" other:filterable="true"
      other:fullpath="restaurant_id" other:apiname="&quot;restaurant_
        id&quot;" />
    <attr name="borough" xs:type="string"
      columnsize="2000" readonly="false"
      description="Dynamic Column." other:dynamodatatype="S"
      other:relativepath="borough" other:filterable="true"
      other:fullpath="borough"
      other:apiname="&quot;borough&quot;" />
  </api:info>
</api:script>

```

```

    <attr name="address_zipcode" xs:type="int"
columnsize="4" precision="10" readonly="false"
description="Dynamic Column." other:dynamodatatype="S"
other:relativepath="zipcode" other:filterable="true"
other:fullpath="address.zipcode"
other:apiname="&quot;address&quot;.&quot;zipcode&quot;"
/>
    <attr name="address_coord_0" xs:type="double"
columnsize="8" precision="15" readonly="false"
description="Dynamic Column." other:dynamodatatype="N"
other:relativepath="coord" other:filterable="true"
other:fullpath="address.coord[0]"
other:apiname="&quot;address&quot;.&quot;coord&quot;[0]"
/>
    <attr name="address_coord_1" xs:type="double"
columnsize="8" precision="15" readonly="false"
description="Dynamic Column." other:dynamodatatype="N"
other:relativepath="coord[1]" other:filterable="true"
other:fullpath="address.coord[1]"
other:apiname="&quot;address&quot;.&quot;coord&quot;[1]"
/>
    <attr name="address_building" xs:type="int"
columnsize="4" precision="10" readonly="false"
description="Dynamic Column." other:dynamodatatype="S"
other:relativepath="building" other:filterable="true"
other:fullpath="address.building"
other:apiname="&quot;address&quot;.&quot;building&quot;"
/>
    <attr name="address_street" xs:type="string"
columnsize="2000" readonly="false"
description="Dynamic Column." other:dynamodatatype="S"
other:relativepath="street" other:filterable="true"
other:fullpath="address.street"
other:apiname="&quot;address&quot;.&quot;street&quot;"
/>
    <attr name="name" xs:type="string"
columnsize="2000" readonly="false"
description="Dynamic Column." other:dynamodatatype="S"
other:relativepath="name" other:filterable="true"
other:fullpath="name"
other:apiname="&quot;name&quot;"
/>
    <attr name="cuisine" xs:type="string"
columnsize="2000" readonly="false"
description="Dynamic Column." other:dynamodatatype="S"
other:relativepath="cuisine" other:filterable="true"
other:fullpath="cuisine"

```

```

other:apiname="&quot;cuisine&quot;"
/>
</api:info>

<api:script method="GET">
  <api:call op="dynamodbProviderOperationCaller">
    <api:push/>
  </api:call>
</api:script>
<api:script method="POST">
  <api:call op="dynamodbProviderOperationCaller">
    <api:push/>
  </api:call>
</api:script>
<api:script method="MERGE">
  <api:call op="dynamodbProviderOperationCaller">
    <api:push/>
  </api:call>
</api:script>
<api:script method="DELETE">
  <api:call op="dynamodbProviderOperationCaller">
    <api:push/>
  </api:call>
</api:script>
</api:script>

```

## Advanced Features

This section details a selection of advanced features of the Amazon DynamoDB adapter.

### Automatic Index Detection

The `AutoDetectIndex` property provides fast access to items in a table by detecting an alternate index which can be queried in place of the table itself. This secondary index is a data structure that contains a subset of attributes from a table and an alternate key. The benefit of querying an index instead of the main table is skipping a full scan of the main table. This makes the operation much faster.

### User Defined Views

The adapter allows you to define virtual tables, called *user defined views*, whose contents are decided by a pre-configured query. These views are useful when you cannot directly

control queries being issued to the drivers. See [User Defined Views](#) for an overview of creating and configuring custom views.

## SSL Configuration

Use [SSL Configuration](#) to adjust how adapter handles TLS/SSL certificate negotiations. You can choose from various certificate formats; see the `SSLServerCert` property under "Connection String Options" for more information.

## Firewall and Proxy

Configure the adapter for compliance with [Firewall and Proxy](#), including Windows proxies and HTTP proxies. You can also set up tunnel connections.

## Query Processing

The adapter offloads as much of the SELECT statement processing as possible to Amazon DynamoDB and then processes the rest of the query in memory (client-side).

See [Query Processing](#) for more information.

## Logging

See [Logging](#) for an overview of configuration settings that can be used to refine CData logging. For basic logging, you only need to set two connection properties, but there are numerous features that support more refined logging, where you can select subsets of information to be logged using the `LogModules` connection property.

## Automatic Index Detection

Amazon DynamoDB provides fast access to items in a table by specifying primary key values. However, to allow efficient access to data with attributes other than the primary key, you can use a *secondary index*, which is a data structure that contains a subset of attributes from a table and an alternate key. The index alternate key is made from a partition key and a sort key.

Every secondary index is associated with exactly one table from which it obtains its data. This is called the *base table* for the index.

The benefit of querying an index instead of the main table (if the index key has been specified in the criteria), is skipping a full scan of the main table. This makes the operation much faster.

## Autodetection Algorithm

The auto-detection algorithm takes the following cases into consideration. If you have specified:

- an index key attribute in the criteria, the index is automatically detected, and it retrieves the items from the secondary index.
- more than one index key attribute in the criteria, an index is detected only if these attributes are part of the same index (i.e., are the partition key and the sort key of the index). Otherwise the data is selected from the main table.
- more than one key index attribute in the criteria, and if these key attributes are not keys of the same secondary index, the items are retrieved from the main table.
- the **SecondaryIndexName** pseudo column in the criteria, the specified value has priority and is the detected index.

## Example

Here is an example of how secondary indexes are built and used:

### [Using Global Secondary Indexes in DynamoDB - Amazon DynamoDB](#)

In the above example, the table **GameScores** has a secondary index, **GameTitleIndex**, which has as its partition key the **GameTitle** attribute. If you run this query:

```
SELECT * FROM GameScores WHERE GameTitle='Alient Adventure'
```

for *AutoDetectIndex=true*, it detects that the **GameTitle** is the partition key of the **GameTitleIndex** secondary index, and the following PartiQL (DynamoDB querying language) is executed, making the query faster by removing the need to scan the whole GameScores table:

```
SELECT * FROM "GameScores"."GameTitleIndex" WHERE GameTitle='Alient Adventure'
```

Turning off the *AutoDetectIndex* connection property executes:



```
SELECT * FROM "GameScores" WHERE GameTitle='Alient Adventure'
```

, which requires the full scanning of the data. You can use a the **SecondaryIndexName** pseudo column to specifically query a secondary index. This transforms this query:

```
SELECT * FROM Table WHERE SecondaryIndexName='index1'
```

to PartiQL:

```
SELECT * FROM "Table"."index1"
```

## User Defined Views

The Amazon DynamoDB Adapter allows you to define a virtual table whose contents are decided by a pre-configured query. These are called *User Defined Views*, which are useful in situations where you cannot directly control the query being issued to the driver, e.g. when using the driver from a tool. The User Defined Views can be used to define predicates that are always applied. If you specify additional predicates in the query to the view, they are combined with the query already defined as part of the view.

There are two ways to create user defined views:

- Create a JSON-formatted configuration file defining the views you want.
- DDL statements.

### Defining Views Using a Configuration File

User Defined Views are defined in a JSON-formatted configuration file called *UserDefinedViews.json*. The adapter automatically detects the views specified in this file.

You can also have multiple view definitions and control them using the UserDefinedViews connection property. When you use this property, only the specified views are seen by the adapter.

This User Defined View configuration file is formatted as follows:

- Each root element defines the name of a view.
- Each root element contains a child element, called **query**, which contains the custom SQL query for the view.

For example:

```
{
  "MyView": {
    "query": "SELECT * FROM Account WHERE MyColumn = 'value'"
  },
  "MyView2": {
    "query": "SELECT * FROM MyTable WHERE Id IN (1,2,3)"
  }
}
```

Use the [UserDefinedViews](#) connection property to specify the location of your JSON configuration file. For example:

```
"UserDefinedViews",
"C:\\Users\\yourusername\\Desktop\\tmp\\UserDefinedViews.json"
```

## Defining Views Using DDL Statements

The adapter is also capable of creating and altering the schema via DDL Statements such as CREATE LOCAL VIEW, ALTER LOCAL VIEW, and DROP LOCAL VIEW.

### Create a View

To create a new view using DDL statements, provide the view name and query as follows:

```
CREATE LOCAL VIEW [MyViewName] AS SELECT * FROM Customers LIMIT 20;
```

If no JSON file exists, the above code creates one. The view is then created in the JSON configuration file and is now discoverable. The JSON file location is specified by the [UserDefinedViews](#) connection property.

### Alter a View

To alter an existing view, provide the name of an existing view alongside the new query you would like to use instead:

```
ALTER LOCAL VIEW [MyViewName] AS SELECT * FROM Customers WHERE
TimeModified > '3/1/2020';
```

The view is then updated in the JSON configuration file.

## Drop a View

To drop an existing view, provide the name of an existing schema alongside the new query you would like to use instead.

```
DROP LOCAL VIEW [MyViewName]
```

This removes the view from the JSON configuration file. It can no longer be queried.

## Schema for User Defined Views

User Defined Views are exposed in the **UserViews** schema by default. This is done to avoid the view's name clashing with an actual entity in the data model. You can change the name of the schema used for UserViews by setting the UserViewsSchemaName property.

## Working with User Defined Views

For example, a SQL statement with a User Defined View called *UserViews.RCustomers* only lists customers in Raleigh:

```
SELECT * FROM Customers WHERE City = 'Raleigh';
```

An example of a query to the driver:

```
SELECT * FROM UserViews.RCustomers WHERE Status = 'Active';
```

Resulting in the effective query to the source:

```
SELECT * FROM Customers WHERE City = 'Raleigh' AND Status = 'Active';
```

That is a very simple example of a query to a User Defined View that is effectively a combination of the view query and the view definition. It is possible to compose these queries in much more complex patterns. All SQL operations are allowed in both queries and are combined when appropriate.

# SSL Configuration

## Customizing the SSL Configuration

By default, the adapter attempts to negotiate SSL/TLS by checking the server's certificate against the system's trusted certificate store.

To specify another certificate, see the [SSLServerCert](#) property for the available formats to do so.

## Firewall and Proxy

### Connecting Through a Firewall or Proxy

#### HTTP Proxies

To connect through the Windows system proxy, you do not need to set any additional connection properties. To connect to other proxies, set [ProxyAutoDetect](#) to false.

In addition, to authenticate to an HTTP proxy, set [ProxyAuthScheme](#), [ProxyUser](#), and [ProxyPassword](#), in addition to [ProxyServer](#) and [ProxyPort](#).

#### Other Proxies

Set the following properties:

- To use a proxy-based firewall, set [FirewallType](#), [FirewallServer](#), and [FirewallPort](#).
- To tunnel the connection, set [FirewallType](#) to TUNNEL.
- To authenticate, specify [FirewallUser](#) and [FirewallPassword](#).
- To authenticate to a SOCKS proxy, additionally set [FirewallType](#) to SOCKS5.

## Query Processing

### Query Processing

CData has a client-side SQL engine built into the adapter library. This enables support for the full capabilities that SQL-92 offers, including filters, aggregations, functions, etc.

For sources that do not support SQL-92, the adapter offloads as much of SQL statement processing as possible to Amazon DynamoDB and then processes the rest of the query in memory (client-side). This results in optimal performance.

For data sources with limited query capabilities, the adapter handles transformations of the SQL query to make it simpler for the adapter. The goal is to make smart decisions based on the query capabilities of the data source to push down as much of the computation as possible. The Amazon DynamoDB Query Evaluation component examines SQL queries and returns information indicating what parts of the query the adapter is not capable of executing natively.

The Amazon DynamoDB Query Slicer component is used in more specific cases to separate a single query into multiple independent queries. The client-side Query Engine makes decisions about simplifying queries, breaking queries into multiple queries, and pushing down or computing aggregations on the client-side while minimizing the size of the result set.

There's a significant trade-off in evaluating queries, even partially, client-side. There are always queries that are impossible to execute efficiently in this model, and some can be particularly expensive to compute in this manner. CData always pushes down as much of the query as is feasible for the data source to generate the most efficient query possible and provide the most flexible query capabilities.

## More Information

For a full discussion of how CData handles query processing, see [CData Architecture: Query Execution](#).

# Logging

Capturing adapter logging can be very helpful when diagnosing error messages or other unexpected behavior.

## Basic Logging

You will simply need to set two connection properties to begin capturing adapter logging.

- Logfile: A filepath which designates the name and location of the log file.
- Verbosity: This is a numerical value (1-5) that determines the amount of detail in the log. See the page in the Connection Properties section for an explanation of the five levels.
- MaxLogFileSize: When the limit is hit, a new log is created in the same folder with the date and time appended to the end. The default limit is 100 MB. Values lower than 100 kB will use 100 kB as the value instead.

- **MaxLogFileCount:** A string specifying the maximum file count of log files. When the limit is hit, a new log is created in the same folder with the date and time appended to the end and the oldest log file will be deleted. Minimum supported value is 2. A value of 0 or a negative value indicates no limit on the count.

Once this property is set, the adapter will populate the log file as it carries out various tasks, such as when authentication is performed or queries are executed. If the specified file doesn't already exist, it will be created.

## Log Verbosity

The verbosity level determines the amount of detail that the adapter reports to the [Logfile](#). [Verbosity](#) levels from 1 to 5 are supported. These are described in the following list:

1	Setting <a href="#">Verbosity</a> to 1 will log the query, the number of rows returned by it, the start of execution and the time taken, and any errors.
2	Setting <a href="#">Verbosity</a> to 2 will log everything included in <a href="#">Verbosity</a> 1 and additional information about the request.
3	Setting <a href="#">Verbosity</a> to 3 will additionally log HTTP headers, as well as the body of the request and the response.
4	Setting <a href="#">Verbosity</a> to 4 will additionally log transport-level communication with the data source. This includes SSL negotiation.
5	Setting <a href="#">Verbosity</a> to 5 will additionally log communication with the data source and additional details that may be helpful in troubleshooting problems. This includes interface commands.

The [Verbosity](#) should not be set to greater than 1 for normal operation. Substantial amounts of data can be logged at higher verbatimities, which can delay execution times.

To refine the logged content further by showing/hiding specific categories of information, see [LogModules](#).

## Sensitive Data

Verbosity levels 3 and higher may capture information that you do not want shared

outside of your organization. The following lists information of concern for each level:

- Verbosity 3: The full body of the request and the response, which includes all the data returned by the adapter
- Verbosity 4: SSL certificates
- Verbosity 5: Any extra transfer data not included at Verbosity 3, such as non human-readable binary transfer data

### Best Practices for Data Security

Although we mask sensitive values, such as passwords, in the connection string and any request in the log, it is always best practice to review the logs for any sensitive information before sharing outside your organization.

## Java Logging

When Java logging is enabled in Logfile, the Verbosity will instead map to the following logging levels.

- 0: Level.WARNING
- 1: Level.INFO
- 2: Level.CONFIG
- 3: Level.FINE
- 4: Level.FINER
- 5: Level.FINEST

## Advanced Logging

You may want to refine the exact information that is recorded to the log file. This can be accomplished using the LogModules property.

This property allows you to filter the logging using a semicolon-separated list of logging modules.

All modules are four characters long. **Please note that modules containing three letters have a required trailing blank space.** The available modules are:

- **EXEC:** Query Execution. Includes execution messages for original SQL queries, parsed SQL queries, and normalized SQL queries. Query and page success/failure messages appear here as well.
- **INFO:** General Information. Includes the connection string, driver version (build number), and initial connection messages.
- **HTTP:** HTTP Protocol messages. Includes HTTP requests/responses (including POST messages), as well as Kerberos related messages.
- **SSL :** SSL certificate messages.
- **OAUT:** OAuth related failure/success messages.
- **SQL :** Includes SQL transactions, SQL bulk transfer messages, and SQL result set messages.
- **META:** Metadata cache and schema messages.
- **TCP :** Incoming and Ongoing raw bytes on TCP transport layer messages.

An example value for this property would be.

```
LogModules=INFO;EXEC;SSL ;SQL ;META;
```

Note that these modules refine the information as it is pulled after taking the [Verbosity](#) into account.

## SQL Compliance

The Amazon DynamoDB Adapter supports several operations on data, including querying, deleting, modifying, and inserting.

### SELECT Statements

See [SELECT Statements](#) for a syntax reference and examples.

See [NoSQL Database](#) for information on the capabilities of the Amazon DynamoDB API.

### INSERT Statements

See [INSERT Statements](#) for a syntax reference and examples, as well as retrieving the new records' Ids.



## UPDATE Statements

The primary key Id is required to update a record. See [UPDATE Statements](#) for a syntax reference and examples.

## DELETE Statements

The primary key Id is required to delete a record. See [DELETE Statements](#) for a syntax reference and examples.

## EXECUTE Statements

Use EXECUTE or EXEC statements to execute stored procedures. See [EXECUTE Statements](#) for a syntax reference and examples.

## Names and Quoting

- Table and column names are considered identifier names; as such, they are restricted to the following characters: [A-Z, a-z, 0-9, \_:@].
- To use a table or column name with characters not listed above, the name must be quoted using double quotes ("name") in any SQL statement.
- Strings must be quoted using single quotes (e.g., 'John Doe').

## SELECT Statements

A SELECT statement can consist of the following basic clauses.

- SELECT
- INTO
- FROM
- JOIN
- WHERE
- GROUP BY
- HAVING

- UNION
- ORDER BY
- LIMIT

## SELECT Syntax

The following syntax diagram outlines the syntax supported by the Amazon DynamoDB adapter:

```

SELECT {
  [ TOP <numeric_literal> ]
  {
    *
    | {
        <expression> [ [ AS ] <column_reference> ]
        | { <table_name> | <correlation_name> } .*
      } [ , ... ]
    }
  [ INTO csv:// [ filename= ] <file_path> [ ;delimiter=tab ] ]
  {
    FROM <table_reference> [ [ AS ] <identifier> ]
  }
  [ WHERE <search_condition> ]
} | SCOPE_IDENTITY()
<expression> ::=
  | <column_reference>
  | @ <parameter>
  | ?
  | COUNT( * | { [ DISTINCT ] <expression> } )
  | { AVG | MAX | MIN | SUM | COUNT } ( <expression> )
  | NULLIF ( <expression> , <expression> )
  | COALESCE ( <expression> , ... )
  | CASE <expression>
      WHEN { <expression> | <search_condition> } THEN { <expression> |
NULL } [ ... ]
  | ELSE { <expression> | NULL } ]
  END
  | <literal>
  | <sql_function>
<search_condition> ::=
  {
    <expression> { = | != | > | < | >= | <= | IS NULL | IS NOT NULL |
CONTAINS | NOT CONTAINS | BETWEEN | IN | NOT IN | LIKE | NOT LIKE | AND
| OR } [ <expression> ]
  } [ { AND | OR } ... ]

```

## Examples

1. Return all columns:

```
SELECT * FROM Account
```

2. Rename a column:

```
SELECT "Name" AS MY_Name FROM Account
```

3. Cast a column's data as a different data type:

```
SELECT CAST(AnnualRevenue AS VARCHAR) AS Str_AnnualRevenue FROM Account
```

4. Search data:

```
SELECT * FROM Account WHERE FirstName <> 'Bob'
```

5. The Amazon DynamoDB APIs support the following operators in the WHERE clause: =, !=, >, <, >=, <=, IS NULL, IS NOT NULL, CONTAINS, NOT CONTAINS, BETWEEN, IN, NOT IN, LIKE, NOT LIKE, AND, OR.

```
SELECT * FROM Account WHERE FirstName <> 'Bob';
```

6. Return the number of items matching the query criteria:

```
SELECT COUNT(*) AS MyCount FROM Account
```

## Projection Functions

### JSON\_AVG(json, jsonpath)

Computes the average value of a JSON array within a JSON object.

- **json:** The column containing JSON data.

- **jsonpath:** The path to the json array.

## JSON\_COUNT(json, jsonpath)

Returns the number of elements in a JSON array within a JSON object.

- **json:** The column containing JSON data.
- **jsonpath:** The path to the json array.

## JSON\_MAX(json, jsonpath)

Gets the maximum value in a JSON array within a JSON object.

- **json:** The column containing JSON data.
- **jsonpath:** The path to the json array.

## JSON\_MIN(json, jsonpath)

Gets the minimum value in a JSON array within a JSON object.

- **json:** The column containing JSON data.
- **jsonpath:** The path to the json array.

## JSON\_SUM(json, jsonpath)

Computes the sum of the elements in a JSON within a JSON object.

- **json:** The column containing JSON data.
- **jsonpath:** The path to the json array.

## JSON\_EXTRACT(json, jsonpath)

Selects any value in a JSON array or object. The path to the array is specified in the jsonpath argument. Return value is numeric or null.

- **json:** The JSON document to extract.

- **jsonpath:** The XPath used to select the nodes. The JSONPath must be a string constant. The values of the nodes selected will be returned in a token-separated list.

## XML\_EXTRACT(xml, xpath [, separator])

Extracts an XML document using the specified XPath to flatten the XML. A comma is used to separate the outputs by default, but this can be changed by specifying the third parameter.

- **xml:** The XML document to extract.
- **xpath:** The XPath used to select the nodes. The nodes selected will be returned in a token-separated list.
- **separator:** The optional token used to separate the items in the flattened response. If this is not specified, the separator will be a comma.

## SELECT INTO Statements

You can use the SELECT INTO statement to export formatted data to a file.

### Data Export with an SQL Query

The following query exports data into a file formatted in comma-separated values (CSV):

```
boolean ret = stat.execute("SELECT Id, Name INTO 'csv://c:/Account.txt'
FROM 'Account' WHERE FirstName <> 'Bob'");
System.out.println(stat.getUpdateCount()+" rows affected");
```

You can specify other file formats in the URI. The following example exports tab-separated values:

```
Statement stat = conn.createStatement();
boolean ret = stat.execute("SELECT * INTO 'Account' IN
'csv://filename=c:/Account.csv;delimiter=tab' FROM 'Account' WHERE
FirstName <> 'Bob'");
System.out.println(stat.getUpdateCount()+" rows affected");
```

## INSERT Statements

To create new records, use INSERT statements.

## INSERT Syntax

The INSERT statement specifies the columns to be inserted and the new column values. You can specify the column values in a comma-separated list in the VALUES clause, as shown in the following example:

```
INSERT INTO <table_name>
( <column_reference> [ , ... ] )
VALUES
( { <expression> | NULL } [ , ... ] )

<expression> ::=
| @ <parameter>
| ?
| <literal>
```

You can use the `executeUpdate` method of the `Statement` and `PreparedStatement` classes to execute data manipulation commands and retrieve the rows affected. To retrieve the Id of the last inserted record use `getGeneratedKeys`. Additionally, set the **RETURN\_GENERATED\_KEYS** flag of the `Statement` class when you call `prepareStatement`.

```
String cmd = "INSERT INTO Account (Name) VALUES (?)";
PreparedStatement pstmt = connection.prepareStatement
(cmd,Statement.RETURN_GENERATED_KEYS);
pstmt.setString(1, "John");
int count = pstmt.executeUpdate();
System.out.println(count+" rows were affected");
ResultSet rs = pstmt.getGeneratedKeys();
while(rs.next()){
    System.out.println(rs.getString("Id"));
}
connection.close();
```

## UPDATE Statements

To modify existing records, use UPDATE statements.

### Update Syntax

The UPDATE statement takes as input a comma-separated list of columns and new column values as name-value pairs in the SET clause, as shown in the following example:

```
UPDATE <table_name> SET { <column_reference> = <expression> } [ , ... ]
WHERE { Id = <expression> } [ { AND | OR } ... ]
<expression> ::=
    | @ <parameter>
    | ?
    | <literal>
```

You can use the `executeUpdate` method of the `Statement` or `PreparedStatement` classes to execute data manipulation commands and retrieve the rows affected, as shown in the following example:

```
String cmd = "UPDATE Account SET Name='John' WHERE Id = ?";
PreparedStatement pstmt = connection.prepareStatement(cmd);
pstmt.setString(1, "1");
int count = pstmt.executeUpdate();
System.out.println(count + " rows were affected");
connection.close();
```

## DELETE Statements

To delete information from a table, use DELETE statements.

### DELETE Syntax

The DELETE statement requires the table name in the FROM clause and the row's primary key in the WHERE clause, as shown in the following example:

```
<delete_statement> ::= DELETE FROM <table_name> WHERE { Id =
<expression> } [ { AND | OR } ... ]
<expression> ::=
    | @ <parameter>
    | ?
    | <literal>
```

You can use the `executeUpdate` method of the `Statement` or `PreparedStatement` classes to execute data manipulation commands and retrieve the number of affected rows, as shown in the following example:

```
Connection connection = DriverManager.getConnection
("jdbc:amazondynamodb:AWS Access Key=xxx;AWS Secret
Key=xxx;Domain=amazonaws.com;AWS Region=OREGON;"),);
String cmd = "DELETE FROM Account WHERE Id = ?";
```

```
PreparedStatement pstmt = connection.prepareStatement(cmd);
pstmt.setString(1, "1");
int count=pstmt.executeUpdate();
connection.close();
```

## EXECUTE Statements

To execute stored procedures, you can use EXECUTE or EXEC statements.

EXEC and EXECUTE assign stored procedure inputs, referenced by name, to values or parameter names.

### Stored Procedure Syntax

To execute a stored procedure as an SQL statement, use the following syntax:

```
{ EXECUTE | EXEC } <stored_proc_name>
{
  [ @ ] <input_name> = <expression>
} [ , ... ]
<expression> ::=
  | @ <parameter>
  | ?
  | <literal>
```

### Example Statements

Reference stored procedure inputs by name:

```
EXECUTE my_proc @second = 2, @first = 1, @third = 3;
```

Execute a parameterized stored procedure statement:

```
EXECUTE my_proc second = @p1, first = @p2, third = @p3;
```

## PIVOT and UNPIVOT

**PIVOT** and **UNPIVOT** can be used to change a table-valued expression into another table.



## PIVOT

PIVOT rotates a table-value expression by turning unique values from one column into multiple columns in the output. PIVOT can run aggregations where required on any column value.

## PIVOT Syntax

```
"SELECT 'AverageCost' AS Cost_Sorted_By_Production_Days, [0], [1], [2],
[3], [4]
FROM
(
SELECT DaysToManufacture, StandardCost
FROM Production.Product
) AS SourceTable
PIVOT
(
AVG(StandardCost)
FOR DaysToManufacture IN ([0], [1], [2], [3], [4])
) AS PivotTable;"
```

## UNPIVOT

UNPIVOT carries out nearly the opposite to PIVOT by rotating columns of a table-valued expressions into column values.

## UNPIVOT Syntax

```
"SELECT VendorID, Employee, Orders
FROM
(SELECT VendorID, Emp1, Emp2, Emp3, Emp4, Emp5
FROM pvt) p
UNPIVOT
(Orders FOR Employee IN
(Emp1, Emp2, Emp3, Emp4, Emp5)
)AS unpvt;"
```

For further information on PIVOT and UNPIVOT, see [FROM clause plus JOIN, APPLY, PIVOT \(Transact-SQL\)](#)

# Data Model

The adapter allows you to access data in Amazon DynamoDB using a standard database-like interface. Amazon DynamoDB is a highly scalable NoSQL cloud database that is very different from a regular database. In this section we describe how we model schemaless Amazon DynamoDB tables as regular [Tables](#) and [Stored Procedures](#).

The adapter can dynamically detect schemas at connection time. See [Automatic Schema Discovery](#) for more information on defining schemas implicitly at connection time. This method is useful if the structure of your data is volatile.

You can also persist schemas in static schema definitions. The adapter's schema files have a simple format. See [Custom Schema Definitions](#) for more information on defining and extending static schemas.

## Tables

The list of tables is dynamically retrieved from your Amazon DynamoDB account. You can use the stored procedure to create a new table, or you can create a table using the Amazon Web Services Admin Console.

Because DynamoDB tables are partitioned based on their key, you should take care in selecting a proper key based on the query requirements of your table. Refer to the documentation for DynamoDB for more information about using best practices to model data in DynamoDB tables. DynamoDB supports two types of primary keys:

- Hash Primary Key: This is a single-column key.
- Hash and Range Primary Key: This is a two-column key that includes a hash column and a range column.

The adapter will model all key attributes in DynamoDB as key columns.

## Table Columns

Since Amazon DynamoDB tables are schemaless, the adapter offers the following two mechanisms to uncover the schema.

### Dynamic Schemas

The columns of a table are dynamically determined by scanning data in the first few rows. You can adjust the number of rows that are used by modifying the [RowScanDepth](#)

---

property. In addition to the name of the column, the row scan also determines the data type. The following table shows how the different data types supported by Amazon DynamoDB are modeled in the adapter.

Amazon DynamoDB Type	Modeled Type	Encoding	Sample Value
Boolean	Boolean	Not Required	True
String	String	Not Required	USA
Blob	String	Not Required	
Number	Double	Not Required	24.0
String Array	String	JSON Array	["USA","Canada","UK"]
Number Array	String	JSON Array	[20,200.5,500]
Blob Array	JSON Array	JSON Array	["ABCD","EFGH"]
Document	JSON Object	JSON Object	{"Address":"123 Fake Street","City":"Chapel Hill","Zip":"27516"}
List	JSON Array	JSON Array	[{"S":"mystring"},{"NS":[1,2]},{"N":4}]

## Static Schemas

Instead of using dynamically discovered schemas, you can define your own schemas. This will give you more control over the projected columns and also enable you to use other data types such as boolean, datetime, etc. Refer to the [CreateSchema](#) Stored Procedure in order to create your own schema. You can simply specify the FileName (fullpath) and

TableName of the new schema file, which should match with the name of the Amazon DynamoDB table, and edit the column listing to use it for your own table.

## Schemaless Operations

While the schema of the table is necessary to report metadata, data may be selected, inserted, updated, or deleted from columns that do not exist in the schema. Columns that do not already exist in the table schema will have their data types dynamically determined based on the data that is specified. See [DynamoDB Queries](#) for more information.

## Stored Procedures

Stored procedures are function-like interfaces that extend the functionality of the adapter beyond simple SELECT/INSERT/UPDATE/DELETE operations with Amazon DynamoDB.

Stored procedures accept a list of parameters, perform their intended function, and then return, if applicable, any relevant response data from Amazon DynamoDB, along with an indication of whether the procedure succeeded or failed.

### Amazon DynamoDB Adapter Stored Procedures

Name	Description
<a href="#">CreateSchema</a>	Creates a schema file for the specified table or view.
<a href="#">CreateTable</a>	Creates a table in DynamoDB.

## CreateSchema

Creates a schema file for the specified table or view.

### CreateSchema

Creates a local schema file (.rsd) from an existing table or view in the data model.

The schema file is created in the directory set in the Location connection property when this procedure is executed. You can edit the file to include or exclude columns, rename columns, or adjust column datatypes.

The adapter checks the Location to determine if the names of any .rsd files match a table or view in the data model. If there is a duplicate, the schema file will take precedence over the default instance of this table in the data model. If a schema file is present in Location that does not match an existing table or view, a new table or view entry is added to the data model of the adapter.

## Input

Name	Type	Required	Accepts Output Streams	Description
TableName	<i>String</i>	<i>True</i>	<i>False</i>	The name of the table or view.
FileName	<i>String</i>	<i>False</i>	<i>False</i>	The full file path and name of the schema to generate. If not set, the FileData output is used instead. Ex : 'C:\\Users\\User\\Desktop\\table.rsd'
FileStream	<i>String</i>	<i>False</i>	<i>True</i>	An instance of an output stream where file data is written to. Only used if FileName is not set.

## Result Set Columns

Name	Type	Description
Result	<i>String</i>	Returns Success or Failure.

FileData	<i>String</i>	The generated schema encoded in Base64. Only returned if FileName is not set.
----------	---------------	---

## CreateTable

Creates a table in DynamoDB.

### Input

Name	Type	Required	Description
TableName	<i>String</i>	<i>True</i>	The name of the table to create. A minimum of 3 characters and maximum of 255 characters are allowed.
PartitionKeyName	<i>String</i>	<i>True</i>	The name of the partition key for the table.
PartitionKeyType	<i>String</i>	<i>True</i>	The type of the partition key for the table. The allowed values are <i>S</i> , <i>N</i> , <i>B</i> .
SortKeyName	<i>String</i>	<i>False</i>	The name of the sort key for the table.
SortKeyType	<i>String</i>	<i>False</i>	The type of the sort key for the table. The allowed values are <i>S</i> , <i>N</i> , <i>B</i> .
BillingMode	<i>String</i>	<i>False</i>	Controls how you are charged for read and write throughput and how you manage capacity.  The allowed values are <i>PROVISIONED</i> , <i>PAY_PER_REQUEST</i> .  The default value is <i>PROVISIONED</i> .
ReadCapacityUnits	<i>String</i>	<i>False</i>	The maximum number of strongly

			consistent reads consumed per second before DynamoDB returns a <code>ThrottlingException</code> .  The default value is 5.
<code>WriteCapacityUnits</code>	<code>String</code>	<code>False</code>	The maximum number of writes consumed per second before DynamoDB returns a <code>ThrottlingException</code> .  The default value is 5.

## Result Set Columns

Name	Type	Description
Success	<code>String</code>	This value shows whether the operation was successful or not.

## Connection String Options

The connection string properties are the various options that can be used to establish a connection. This section provides a complete list of the options you can configure in the connection string for this provider. Click the links for further details.

For more information on establishing a connection, see [Basic Tab](#).

## AWS Authentication

Property	Description
<a href="#">AuthScheme</a>	The scheme used for authentication. Accepted entries are: <code>Auto</code> , <code>AwsRootKeys</code> , <code>AwsIAMRoles</code> , <code>AwsEC2Roles</code> , <code>AwsMFA</code> , <code>ADFS</code> , <code>Okta</code> , <code>PingFederate</code> , <code>AwsCredentialsFile</code> , <code>AwsCognitoBasic</code> , <code>AwsCognitoSrp</code> .

<a href="#">Domain</a>	Your AWS domain name. You can optionally choose to associate your domain name with AWS.
<a href="#">AWSAccessKey</a>	Your AWS account access key. This value is accessible from your AWS security credentials page.
<a href="#">AWSSecretKey</a>	Your AWS account secret key. This value is accessible from your AWS security credentials page.
<a href="#">AWSRoleARN</a>	The Amazon Resource Name of the role to use when authenticating.
<a href="#">AWSRegion</a>	The hosting region for your Amazon Web Services.
<a href="#">AWSCredentialsFile</a>	The path to the AWS Credentials File to be used for authentication.
<a href="#">AWSCredentialsFileProfile</a>	The name of the profile to be used from the supplied AWSCredentialsFile.
<a href="#">AWSSessionToken</a>	Your AWS session token.
<a href="#">AWSExternalId</a>	A unique identifier that might be required when you assume a role in another account.
<a href="#">MFASerialNumber</a>	The serial number of the MFA device if one is being used.
<a href="#">MFAToken</a>	The temporary token available from your MFA device.
<a href="#">CredentialsLocation</a>	The location of the settings file where MFA credentials are saved.
<a href="#">TemporaryTokenDuration</a>	The amount of time (in seconds) a temporary token will last.
<a href="#">AWSCognitoRegion</a>	The hosting region for AWS Cognito.
<a href="#">AWSUserPoolId</a>	The User Pool Id.
<a href="#">AWSUserPoolClientId</a>	The User Pool Client App Id.



---

<a href="#">AWSUserPoolClientAppSecret</a>	Optional. The User Pool Client App Secret.
--	--

---

<a href="#">AWSIdentityPoolId</a>	The Identity Pool Id.
-----------------------------------	-----------------------

---

## SSO

---

Property	Description
<a href="#">User</a>	The IDP user used to authenticate the IDP via SSO.
<a href="#">Password</a>	The password used to authenticate the IDP user via SSO.
<a href="#">SSOLoginURL</a>	The identity provider's login URL.
<a href="#">SSOProperties</a>	Additional properties required to connect to the identity provider in a semicolon-separated list.
<a href="#">SSOExchangeUrl</a>	The url used for consuming the SAML response and exchanging it with Amazon DynamoDB specific credentials.

---

## SSL

---

Property	Description
<a href="#">SSLServerCert</a>	The certificate to be accepted from the server when connecting using TLS/SSL.

---

## Firewall

---

Property	Description
<a href="#">FirewallType</a>	The protocol used by a proxy-based firewall.

---

<a href="#">FirewallServer</a>	The name or IP address of a proxy-based firewall.
<a href="#">FirewallPort</a>	The TCP port for a proxy-based firewall.
<a href="#">FirewallUser</a>	The user name to use to authenticate with a proxy-based firewall.
<a href="#">FirewallPassword</a>	A password used to authenticate to a proxy-based firewall.

## Proxy

Property	Description
<a href="#">ProxyAutoDetect</a>	This indicates whether to use the system proxy settings or not. This takes precedence over other proxy settings, so you'll need to set ProxyAutoDetect to FALSE in order use custom proxy settings.
<a href="#">ProxyServer</a>	The hostname or IP address of a proxy to route HTTP traffic through.
<a href="#">ProxyPort</a>	The TCP port the ProxyServer proxy is running on.
<a href="#">ProxyAuthScheme</a>	The authentication type to use to authenticate to the ProxyServer proxy.
<a href="#">ProxyUser</a>	A user name to be used to authenticate to the ProxyServer proxy.
<a href="#">ProxyPassword</a>	A password to be used to authenticate to the ProxyServer proxy.
<a href="#">ProxySSLType</a>	The SSL type to use when connecting to the ProxyServer proxy.
<a href="#">ProxyExceptions</a>	A semicolon separated list of destination hostnames or IPs that are exempt from connecting through the ProxyServer .

## Logging

Property	Description
<a href="#">LogModules</a>	Core modules to be included in the log file.

## Schema

Property	Description
<a href="#">Location</a>	A path to the directory that contains the schema files defining tables, views, and stored procedures.

## Miscellaneous

Property	Description
<a href="#">AutoDetectIndex</a>	A boolean indicating if secondary indexes should be automatically detected based on the query used.
<a href="#">FlattenArrays</a>	By default, nested arrays are returned as strings of JSON. The FlattenArrays property can be used to flatten the elements of nested arrays into columns of their own. Set FlattenArrays to the number of elements you want to return from nested arrays.
<a href="#">FlattenObjects</a>	Set FlattenObjects to true to flatten object properties into columns of their own. Otherwise, objects nested in arrays are returned as strings of JSON.
<a href="#">FlexibleSchema</a>	Set FlexibleSchema to true to scan for additional metadata on the query result set. Otherwise, the metadata will remain the same.
<a href="#">GenerateSchemaFiles</a>	Indicates the user preference as to when schemas should be generated and saved.

<a href="#">IgnoreTypes</a>	Removes support for the specified types. For example, Time. These types will then be reported as strings instead.
<a href="#">MaximumRequestRetries</a>	The maximum number of times to retry a request.
<a href="#">MaxRows</a>	Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.
<a href="#">Other</a>	These hidden properties are used only in specific use cases.
<a href="#">Pagesize</a>	The maximum number of results to return per page from Amazon DynamoDB.
<a href="#">Readonly</a>	You can use this property to enforce read-only access to Amazon DynamoDB from the provider.
<a href="#">RetryWaitTime</a>	The minimum number of milliseconds the provider will wait to retry a request.
<a href="#">RowScanDepth</a>	The maximum number of rows to scan to look for the columns available in a table.
<a href="#">SeparatorCharacter</a>	The character or characters used to denote hierarchy.
<a href="#">ThreadCount</a>	The number of threads to use when selecting data via a parallel scan. Setting ThreadCount to 1 will disable parallel scans.
<a href="#">Timeout</a>	The value in seconds until the timeout error is thrown, canceling the operation.
<a href="#">TypeDetectionScheme</a>	Determines how to determine the data type of columns.
<a href="#">UseBatchWriteItemOperation</a>	When enabled the provider will use BatchWriteItem operation for handling updates and inserts. By default, the provider uses ExecuteStatement/BatchExecuteStatement operation. You need to enable BatchWriteItem only when inserting/updating binary/binary-set data. ExecuteStatement/BatchExecuteStatement doesn't support

	manipulating binary fields.
<a href="#">UseConsistentReads</a>	Whether to always use Consistent Reads or not when querying DynamoDb.
<a href="#">UserDefinedViews</a>	A filepath pointing to the JSON configuration file containing your custom views.
<a href="#">UseSimpleNames</a>	Boolean determining if simple names should be used for tables and columns.

## AWS Authentication

This section provides a complete list of the AWS Authentication properties you can configure in the connection string for this provider.

Property	Description
<a href="#">AuthScheme</a>	The scheme used for authentication. Accepted entries are: Auto, , AwsRootKeys , AwsIAMRoles , AwsEC2Roles , AwsMFA , ADFS, Okta, PingFederate , AwsCredentialsFile , AwsCognitoBasic , AwsCognitoSrp.
<a href="#">Domain</a>	Your AWS domain name. You can optionally choose to associate your domain name with AWS.
<a href="#">AWSAccessKey</a>	Your AWS account access key. This value is accessible from your AWS security credentials page.
<a href="#">AWSSecretKey</a>	Your AWS account secret key. This value is accessible from your AWS security credentials page.
<a href="#">AWSRoleARN</a>	The Amazon Resource Name of the role to use when authenticating.
<a href="#">AWSRegion</a>	The hosting region for your Amazon Web Services.

<a href="#">AWSCredentialsFile</a>	The path to the AWS Credentials File to be used for authentication.
<a href="#">AWSCredentialsFileProfile</a>	The name of the profile to be used from the supplied AWSCredentialsFile.
<a href="#">AWSSessionToken</a>	Your AWS session token.
<a href="#">AWSExternalId</a>	A unique identifier that might be required when you assume a role in another account.
<a href="#">MFASerialNumber</a>	The serial number of the MFA device if one is being used.
<a href="#">MFAToken</a>	The temporary token available from your MFA device.
<a href="#">CredentialsLocation</a>	The location of the settings file where MFA credentials are saved.
<a href="#">TemporaryTokenDuration</a>	The amount of time (in seconds) a temporary token will last.
<a href="#">AWSCognitoRegion</a>	The hosting region for AWS Cognito.
<a href="#">AWSUserPoolId</a>	The User Pool Id.
<a href="#">AWSUserPoolClientId</a>	The User Pool Client App Id.
<a href="#">AWSUserPoolClientAppSecret</a>	Optional. The User Pool Client App Secret.
<a href="#">AWSIdentityPoolId</a>	The Identity Pool Id.

## AuthScheme

The scheme used for authentication. Accepted entries are: Auto, , AwsRootKeys , AwsIAMRoles , AwsEC2Roles , AwsMFA , ADFS, Okta, PingFederate , AwsCredentialsFile , AwsCognitoBasic , AwsCognitoSrp.

## Possible Values

Auto, ADFS, AwsRootKeys, AwsIAMRoles, AwsEC2Roles, AwsMFA, AwsCredentialsFile, Okta,

TemporaryCredentials, PingFederate, AwsCognitoBasic, AwsCognitoSrp,

## Data Type

string

## Default Value

""

## Remarks

Use the following options to select your authentication scheme:

- **Auto:** Set this to have the adapter attempt to automatically resolve the proper authentication scheme to use based on the other connection properties specified.
- **TemporaryCredentials:** Set this to leverage temporary security credentials alongside a session token to connect.
- **AwsRootKeys:** Set this to use the root user access key and secret. Useful for quickly testing, but production use cases are encouraged to use something with narrowed permissions.
- **AwsIAMRoles:** Set to use IAM Roles for the connection.
- **AwsEC2Roles:** Set this to automatically use IAM Roles assigned to the EC2 machine the Amazon DynamoDB Adapter is currently running on.
- **AwsMFA:** Set to use multi factor authentication.
- **Okta:** Set to use a single sign on connection with OKTA as the identity provider.
- **ADFS:** Set to use a single sign on connection with ADFS as the identity provider.
- **PingFederate:** Set to use a single sign on connection with PingFederate as the identity provider.
- **AwsCredentialsFile:** Set to use a credential file for authentication.
- **AwsCognitoSrp:** Set to use Cognito based authentication. This is recommended over AwsCognitoBasic because this option does NOT send the password to the server for authentication, instead it uses the SRP protocol.
- **AwsCognitoBasic:** Set to use Cognito based authentication.

## Domain

Your AWS domain name. You can optionally choose to associate your domain name with AWS.

### Data Type

string

### Default Value

"amazonaws.com"

### Remarks

If you do not have a unique AWS domain name, leave this value specified as amazonaws.com.

## AWSAccessKey

Your AWS account access key. This value is accessible from your AWS security credentials page.

### Data Type

string

### Default Value

""

### Remarks

Your AWS account access key. This value is accessible from your AWS security credentials page:

1. Sign into the AWS Management console with the credentials for your root account.
2. Select your account name or number and select My Security Credentials in the menu



that is displayed.

3. Click Continue to Security Credentials and expand the Access Keys section to manage or create root account access keys.

## AWSecretKey

Your AWS account secret key. This value is accessible from your AWS security credentials page.

### Data Type

string

### Default Value

""

### Remarks

Your AWS account secret key. This value is accessible from your AWS security credentials page:

1. Sign into the AWS Management console with the credentials for your root account.
2. Select your account name or number and select My Security Credentials in the menu that is displayed.
3. Click Continue to Security Credentials and expand the Access Keys section to manage or create root account access keys.

## AWSRoleARN

The Amazon Resource Name of the role to use when authenticating.

### Data Type

string

## Default Value

""

## Remarks

When authenticating outside of AWS, it is common to use a Role for authentication instead of your direct AWS account credentials. Entering the [AWSRoleARN](#) will cause the Amazon DynamoDB Adapter to perform a role based authentication instead of using the [AWSAccessKey](#) and [AWSSecretKey](#) directly. The [AWSAccessKey](#) and [AWSSecretKey](#) must still be specified to perform this authentication. You cannot use the credentials of an AWS root user when setting RoleARN. The [AWSAccessKey](#) and [AWSSecretKey](#) must be those of an IAM user.

## AWSRegion

The hosting region for your Amazon Web Services.

## Possible Values

OHIO, NORTHERNVIRGINIA, NORTHERNCALIFORNIA, OREGON, CAPETOWN, HONGKONG, JAKARTA, MUMBAI, OSAKA, SEOUL, SINGAPORE, SYDNEY, TOKYO, CENTRAL, BEIJING, NINGXIA, FRANKFURT, IRELAND, LONDON, MILAN, PARIS, STOCKHOLM, ZURICH, BAHRAIN, UAE, SAOPAULO, GOVCLOUDEAST, GOVCLLOUDWEST

## Data Type

string

## Default Value

"NORTHERNVIRGINIA"

## Remarks

The hosting region for your Amazon Web Services. Available values are OHIO, NORTHERNVIRGINIA, NORTHERNCALIFORNIA, OREGON, CAPETOWN, HONGKONG, JAKARTA, MUMBAI, OSAKA, SEOUL, SINGAPORE, SYDNEY, TOKYO, CENTRAL, BEIJING, NINGXIA, FRANKFURT, IRELAND, LONDON, MILAN, PARIS, STOCKHOLM, ZURICH, BAHRAIN, UAE, SAOPAULO, GOVCLOUDEAST, and GOVCLLOUDWEST.

## AWSCredentialsFile

The path to the AWS Credentials File to be used for authentication.

### Data Type

string

### Default Value

""

### Remarks

The path to the AWS Credentials File to be used for authentication. See <https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html> for more information.

## AWSCredentialsFileProfile

The name of the profile to be used from the supplied AWSCredentialsFile.

### Data Type

string

### Default Value

"default"

### Remarks

The name of the profile to be used from the supplied AWSCredentialsFile. See <https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html> for more information.

## AWSSessionToken

Your AWS session token.

## Data Type

string

## Default Value

""

## Remarks

Your AWS session token. This value can be retrieved in different ways. See [this link](#) for more info.

## AWSExternalId

A unique identifier that might be required when you assume a role in another account.

## Data Type

string

## Default Value

""

## Remarks

A unique identifier that might be required when you assume a role in another account.

## MFASerialNumber

The serial number of the MFA device if one is being used.

## Data Type

string

## Default Value

""

## Remarks

You can find the device for an IAM user by going to the AWS Management Console and viewing the user's security credentials. For virtual devices, this is actually an Amazon Resource Name (such as `arn:aws:iam::123456789012:mfa/user`).

## MFAToken

The temporary token available from your MFA device.

## Data Type

string

## Default Value

""

## Remarks

If MFA is required, this value will be used along with the [MFASerialNumber](#) to retrieve temporary credentials to login. The temporary credentials available from AWS will only last up to 1 hour by default (see [TemporaryTokenDuration](#)). Once the time is up, the connection must be updated to specify a new MFA token so that new credentials may be obtained.

## CredentialsLocation

The location of the settings file where MFA credentials are saved.

## Data Type

string

## Default Value

"%APPDATA%\\CData\\AmazonDynamoDB Data Provider\\CredentialsFile.txt"

## Remarks

MFA credentials are short lived and typically expire after an hour. At that point, a different [MFAToken](#) must be specified to continue connecting. MFA tokens only work once, so in the case of multi-threaded or multi-process applications, the credentials must be centrally located and shared to avoid problems. When [MFAToken](#) is not specified, this property does nothing.

If left unspecified, the default location is "%APPDATA%\\CData\\AmazonDynamoDB Data Provider\\CredentialsFile.txt" with **%APPDATA%** being set to the user's configuration directory:

Platform	%APPDATA%
Windows	The value of the APPDATA environment variable
Mac	~/Library/Application Support
Linux	~/.config

MFA credentials are short lived and typically expire after an hour. At that point, a different [MFAToken](#) must be specified to continue connecting. MFA tokens only work once, so in the case of multi-threaded or multi-process applications, the credentials must be centrally located and shared to avoid problems. When [MFAToken](#) is not specified, this property does nothing.

If left unspecified, the default location is "%APPDATA%\\CData\\AmazonDynamoDB Data Provider\\CredentialsFile.txt" with **%APPDATA%** being set to the user's configuration directory:

Platform	%APPDATA%
Windows	The value of the APPDATA environment variable

Mac	~/Library/Application Support
Linux	~/.config

## TemporaryTokenDuration

The amount of time (in seconds) a temporary token will last.

### Data Type

string

### Default Value

"3600"

### Remarks

Temporary tokens are used with both MFA and Role based authentication. Temporary tokens will eventually time out, at which time a new temporary token must be obtained. For situations where MFA is not used, this is not a big deal. The Amazon DynamoDB Adapter will internally request a new temporary token once the temporary token has expired.

However, for MFA required connection, a new [MFAToken](#) must be specified in the connection to retrieve a new temporary token. This is a more intrusive issue since it requires an update to the connection by the user. The maximum and minimum that can be specified will depend largely on the connection being used.

For Role based authentication, the minimum duration is 900 seconds (15 minutes) while the maximum is 3600 (1 hour). Even if MFA is used with role based authentication, 3600 is still the maximum.

For MFA authentication by itself (using an IAM User or root user), the minimum is 900 seconds (15 minutes), the maximum is 129600 (36 hours).

## AWSCognitoRegion

The hosting region for AWS Cognito.

## Possible Values

OHIO, NORTHERNVIRGINIA, NORTHERNCALIFORNIA, OREGON, CAPETOWN, HONGKONG, MUMBAI, OSAKA, SEOUL, SINGAPORE, SYDNEY, TOKYO, CENTRAL, BEIJING, NINGXIA, FRANKFURT, IRELAND, LONDON, MILAN, PARIS, STOCKHOLM, BAHRAIN, SAOPAULO, GOVCLOUDEAST, GOVCLOUDWEST

## Data Type

string

## Default Value

"NORTHERNVIRGINIA"

## Remarks

The hosting region for AWS Cognito. Available values are OHIO, NORTHERNVIRGINIA, NORTHERNCALIFORNIA, OREGON, CAPETOWN, HONGKONG, MUMBAI, OSAKA, SEOUL, SINGAPORE, SYDNEY, TOKYO, CENTRAL, BEIJING, NINGXIA, FRANKFURT, IRELAND, LONDON, MILAN, PARIS, STOCKHOLM, BAHRAIN, SAOPAULO, GOVCLOUDEAST, and GOVCLOUDWEST.

## AWSUserPoolId

The User Pool Id.

## Data Type

string

## Default Value

""

## Remarks

You can find this in AWS Cognito -> Manage User Pools -> select your user pool -> General settings -> Pool Id.



## AWSUserPoolClientId

The User Pool Client App Id.

### Data Type

string

### Default Value

""

### Remarks

You can find this in AWS Cognito -> Manage Identity Pools -> select your user pool -> General settings -> App clients -> App client Id.

## AWSUserPoolClientAppSecret

Optional. The User Pool Client App Secret.

### Data Type

string

### Default Value

""

### Remarks

You can find this in AWS Cognito -> Manage Identity Pools -> select your user pool -> General settings -> App clients -> App client secret.

## AWSIdentityPoolId

The Identity Pool Id.

## Data Type

string

## Default Value

""

## Remarks

You can find this in AWS Cognito -> Manage Identity Pools -> select your identity pool -> Edit identity pool -> Identity Pool Id

## SSO

This section provides a complete list of the SSO properties you can configure in the connection string for this provider.

Property	Description
<a href="#">User</a>	The IDP user used to authenticate the IDP via SSO.
<a href="#">Password</a>	The password used to authenticate the IDP user via SSO.
<a href="#">SSOLoginURL</a>	The identity provider's login URL.
<a href="#">SSOProperties</a>	Additional properties required to connect to the identity provider in a semicolon-separated list.
<a href="#">SSOExchangeUrl</a>	The url used for consuming the SAML response and exchanging it with Amazon DynamoDB specific credentials.

## User

The IDP user used to authenticate the IDP via SSO.

## Data Type

string

## Default Value

""

## Remarks

Together with [Password](#), this field is used to authenticate in SSO connections against the Amazon DynamoDB server.

## Password

The password used to authenticate the IDP user via SSO.

## Data Type

string

## Default Value

""

## Remarks

The [User](#) and [Password](#) are together used in SSO connections to authenticate with the server.

## SSOLoginURL

The identity provider's login URL.

## Data Type

string

## Default Value

""

## Remarks

The identity provider's login URL.

## SSOProperties

Additional properties required to connect to the identity provider in a semicolon-separated list.

## Data Type

string

## Default Value

""

## Remarks

Additional properties required to connect to the identity provider in a semicolon-separated list. SSOProperties is used in conjunction with the the [AWSRoleARN](#) and [AWSPrincipalARN](#). The following section provides an example using the OKTA identity provider.

## ADFS

Set the [AuthScheme](#) to **ADFS**. The following connection properties need to be set:

- [User](#): Set this to your ADFS username.
- [Password](#): Set this to your ADFS password.
- [SSOLoginURL](#): Set this to the login URL used by the SSO provider.

Below is an example connection string:

```
AuthScheme=ADFS; AWSRegion=Ireland; User=user@cdata.com;
Password=CH8WerW121235647iCa6; SSOLoginURL='https://adfs.domain.com';
AWSRoleArn=arn:aws:iam::1234:role/ADFS_SSO;
```

```
AWSPrincipalArn=arn:aws:iam::1234:saml-provider/ADFSProvider;
S3StagingDirectory=s3://athena/staging;
```

## ADFS Integrated

To use the ADFS Integrated flow, specify the [SSOLoginURL](#) and leave the username and password empty.

## Okta

Set the [AuthScheme](#) to **Okta**. The following connection properties are used to authenticate through Okta:

- [User](#): Set to your Okta user.
- [Password](#): Set to your Okta password.
- [SSOLoginURL](#): Set to the login URL used by the SSO provider.

If you are:

- using a trusted application or proxy that overrides the Okta client request
- configuring MFA

then you need to use combinations of [SSOProperties](#) input parameters to authenticate using Okta. Otherwise, you do not need to set any of these values.

In [SSOProperties](#) when required, set these input parameters:

- [APIToken](#): When authenticating a user via a trusted application or proxy that overrides the Okta client request context, set this to the API Token the customer created from the Okta organization.
- [MFAType](#): Set this if you have configured the MFA flow. Currently we support the following types: OktaVerify, Email, and SMS.
- [MFAPassCode](#): Set this only if you have configured the MFA flow. If you set this to empty or an invalid value, the adapter issues a one-time password challenge to your device or email. After the passcode is received, reopen the connection where the retrieved one-time password value is set to the MFAPassCode connection property.
- [MFARememberDevice](#): Okta supports remembering devices when MFA is required. If remembering devices is allowed according to the configured authentication policies, the adapter sends a device token to extend MFA authentication lifetime. This

property is, by default, set to **True**. Set this to **False** only if you do not want MFA to be remembered.

Example connection string:

```
AuthScheme=Okta; AWSRegion=Ireland; User=user@cdata.com;
Password=CH8WerW121235647iCa6; SSOLoginURL='https://cdata-
us.okta.com/home/amazon_aws/0oa35m8arsAL5f5NrE6NdA356/272';
SSOProperties='ApiToken=01230GGG2ceAnm_tPAf4MhiMELXZ0L0N1pAYr01VR-
hGQSf;'; AWSRoleArn=arn:aws:iam::1234:role/Okta_SSO;
AWSPrincipalARN=arn:aws:iam::1234:saml-provider/OktaProvider;
S3StagingDirectory=s3://athena/staging;
```

## SSOExchangeUrl

The url used for consuming the SAML response and exchanging it with Amazon DynamoDB specific credentials.

### Data Type

string

### Default Value

""

### Remarks

The Amazon DynamoDB Adapter will use the url specified here to consume a SAML response and retrieve Amazon DynamoDB specific credentials. The retrieved credentials are the final piece during the SSO connection that are used to communicate with Amazon DynamoDB.

## SSL

This section provides a complete list of the SSL properties you can configure in the connection string for this provider.



---

colon separated)

---

The SHA1 Thumbprint (hex values can also be either space or colon separated)

34a929226ae0819f2ec14b4a3d904f801cbb150d

---

If not specified, any certificate trusted by the machine is accepted.

Certificates are validated as trusted by the machine based on the System's trust store. The trust store used is the 'javax.net.ssl.trustStore' value specified for the system. If no value is specified for this property, Java's default trust store is used (for example, JAVA\_HOME\lib\security\cacerts).

Use '\*' to signify to accept all certificates. Note that this is not recommended due to security concerns.

## Firewall

This section provides a complete list of the Firewall properties you can configure in the connection string for this provider.

---

Property	Description
<a href="#">FirewallType</a>	The protocol used by a proxy-based firewall.
<a href="#">FirewallServer</a>	The name or IP address of a proxy-based firewall.
<a href="#">FirewallPort</a>	The TCP port for a proxy-based firewall.
<a href="#">FirewallUser</a>	The user name to use to authenticate with a proxy-based firewall.
<a href="#">FirewallPassword</a>	A password used to authenticate to a proxy-based firewall.

---

## FirewallType

The protocol used by a proxy-based firewall.

---



## Possible Values

NONE, TUNNEL, SOCKS4, SOCKS5

## Data Type

string

## Default Value

"NONE"

## Remarks

This property specifies the protocol that the adapter will use to tunnel traffic through the [FirewallServer](#) proxy. Note that by default, the adapter connects to the system proxy; to disable this behavior and connect to one of the following proxy types, set [ProxyAutoDetect](#) to false.

Type	Default Port	Description
TUNNEL	80	When this is set, the adapter opens a connection to Amazon DynamoDB and traffic flows back and forth through the proxy.
SOCKS4	1080	When this is set, the adapter sends data through the SOCKS 4 proxy specified by <a href="#">FirewallServer</a> and <a href="#">FirewallPort</a> and passes the <a href="#">FirewallUser</a> value to the proxy, which determines if the connection request should be granted.
SOCKS5	1080	When this is set, the adapter sends data through the SOCKS 5 proxy specified by <a href="#">FirewallServer</a> and <a href="#">FirewallPort</a> . If your proxy requires authentication, set <a href="#">FirewallUser</a> and <a href="#">FirewallPassword</a> to credentials the proxy recognizes.

To connect to HTTP proxies, use [ProxyServer](#) and [ProxyPort](#). To authenticate to HTTP proxies, use [ProxyAuthScheme](#), [ProxyUser](#), and [ProxyPassword](#).

## FirewallServer

The name or IP address of a proxy-based firewall.

## Data Type

string

## Default Value

""

## Remarks

This property specifies the IP address, DNS name, or host name of a proxy allowing traversal of a firewall. The protocol is specified by [FirewallType](#): Use [FirewallServer](#) with this property to connect through SOCKS or do tunneling. Use [ProxyServer](#) to connect to an HTTP proxy.

Note that the adapter uses the system proxy by default. To use a different proxy, set [ProxyAutoDetect](#) to false.

## FirewallPort

The TCP port for a proxy-based firewall.

## Data Type

int

## Default Value

0

## Remarks

This specifies the TCP port for a proxy allowing traversal of a firewall. Use [FirewallServer](#) to specify the name or IP address. Specify the protocol with [FirewallType](#).

## FirewallUser

The user name to use to authenticate with a proxy-based firewall.

## Data Type

string

## Default Value

""

## Remarks

The [FirewallUser](#) and [FirewallPassword](#) properties are used to authenticate against the proxy specified in [FirewallServer](#) and [FirewallPort](#), following the authentication method specified in [FirewallType](#).

## FirewallPassword

A password used to authenticate to a proxy-based firewall.

## Data Type

string

## Default Value

""

## Remarks

This property is passed to the proxy specified by [FirewallServer](#) and [FirewallPort](#), following the authentication method specified by [FirewallType](#).

## Proxy

This section provides a complete list of the Proxy properties you can configure in the connection string for this provider.

---

Property	Description
<a href="#">ProxyAutoDetect</a>	This indicates whether to use the system proxy settings or not. This takes precedence over other proxy settings, so you'll need to set ProxyAutoDetect to FALSE in order use custom proxy settings.
<a href="#">ProxyServer</a>	The hostname or IP address of a proxy to route HTTP traffic through.
<a href="#">ProxyPort</a>	The TCP port the ProxyServer proxy is running on.
<a href="#">ProxyAuthScheme</a>	The authentication type to use to authenticate to the ProxyServer proxy.
<a href="#">ProxyUser</a>	A user name to be used to authenticate to the ProxyServer proxy.
<a href="#">ProxyPassword</a>	A password to be used to authenticate to the ProxyServer proxy.
<a href="#">ProxySSLType</a>	The SSL type to use when connecting to the ProxyServer proxy.
<a href="#">ProxyExceptions</a>	A semicolon separated list of destination hostnames or IPs that are exempt from connecting through the ProxyServer .

## ProxyAutoDetect

This indicates whether to use the system proxy settings or not. This takes precedence over other proxy settings, so you'll need to set ProxyAutoDetect to FALSE in order use custom proxy settings.

### Data Type

bool

### Default Value

true

## Remarks

This takes precedence over other proxy settings, so you'll need to set `ProxyAutoDetect` to `FALSE` in order use custom proxy settings.

NOTE: When this property is set to `True`, the proxy used is determined as follows:

- A search from the JVM properties (**`http.proxy`**, **`https.proxy`**, **`socksProxy`**, etc.) is performed.
- In the case that the JVM properties don't exist, a search from **`java.home/lib/net.properties`** is performed.
- In the case that `java.net.useSystemProxies` is set to `True`, a search from **the `SystemProxy`** is performed.
- In Windows only, an attempt is made to retrieve these properties from the **Internet Options** in the **registry**.

To connect to an HTTP proxy, see [ProxyServer](#). For other proxies, such as SOCKS or tunneling, see [FirewallType](#).

## ProxyServer

The hostname or IP address of a proxy to route HTTP traffic through.

### Data Type

string

### Default Value

""

## Remarks

The hostname or IP address of a proxy to route HTTP traffic through. The adapter can use the HTTP, Windows (NTLM), or Kerberos authentication types to authenticate to an HTTP proxy.

If you need to connect through a SOCKS proxy or tunnel the connection, see [FirewallType](#).

By default, the adapter uses the system proxy. If you need to use another proxy, set [ProxyAutoDetect](#) to `false`.

## ProxyPort

The TCP port the ProxyServer proxy is running on.

### Data Type

int

### Default Value

80

### Remarks

The port the HTTP proxy is running on that you want to redirect HTTP traffic through. Specify the HTTP proxy in [ProxyServer](#). For other proxy types, see [FirewallType](#).

## ProxyAuthScheme

The authentication type to use to authenticate to the ProxyServer proxy.

### Possible Values

BASIC, DIGEST, NONE, NEGOTIATE, NTLM, PROPRIETARY

### Data Type

string

### Default Value

"BASIC"

### Remarks

This value specifies the authentication type to use to authenticate to the HTTP proxy specified by [ProxyServer](#) and [ProxyPort](#).

Note that the adapter will use the system proxy settings by default, without further configuration needed; if you want to connect to another proxy, you will need to set

[ProxyAutoDetect](#) to false, in addition to [ProxyServer](#) and [ProxyPort](#). To authenticate, set [ProxyAuthScheme](#) and set [ProxyUser](#) and [ProxyPassword](#), if needed.

The authentication type can be one of the following:

- **BASIC:** The adapter performs HTTP BASIC authentication.
- **DIGEST:** The adapter performs HTTP DIGEST authentication.
- **NEGOTIATE:** The adapter retrieves an NTLM or Kerberos token based on the applicable protocol for authentication.
- **PROPRIETARY:** The adapter does not generate an NTLM or Kerberos token. You must supply this token in the Authorization header of the HTTP request.

If you need to use another authentication type, such as SOCKS 5 authentication, see [FirewallType](#).

## ProxyUser

A user name to be used to authenticate to the ProxyServer proxy.

### Data Type

string

### Default Value

""

### Remarks

The [ProxyUser](#) and [ProxyPassword](#) options are used to connect and authenticate against the HTTP proxy specified in [ProxyServer](#).

You can select one of the available authentication types in [ProxyAuthScheme](#). If you are using HTTP authentication, set this to the user name of a user recognized by the HTTP proxy. If you are using Windows or Kerberos authentication, set this property to a user name in one of the following formats:

```
user@domain
domain\user
```

## ProxyPassword

A password to be used to authenticate to the ProxyServer proxy.

### Data Type

string

### Default Value

""

### Remarks

This property is used to authenticate to an HTTP proxy server that supports NTLM (Windows), Kerberos, or HTTP authentication. To specify the HTTP proxy, you can set [ProxyServer](#) and [ProxyPort](#). To specify the authentication type, set [ProxyAuthScheme](#).

If you are using HTTP authentication, additionally set [ProxyUser](#) and [ProxyPassword](#) to HTTP proxy.

If you are using NTLM authentication, set [ProxyUser](#) and [ProxyPassword](#) to your Windows password. You may also need these to complete Kerberos authentication.

For SOCKS 5 authentication or tunneling, see [FirewallType](#).

By default, the adapter uses the system proxy. If you want to connect to another proxy, set [ProxyAutoDetect](#) to false.

## ProxySSLType

The SSL type to use when connecting to the ProxyServer proxy.

### Possible Values

AUTO, ALWAYS, NEVER, TUNNEL

### Data Type

string



## Default Value

"AUTO"

## Remarks

This property determines when to use SSL for the connection to an HTTP proxy specified by [ProxyServer](#). This value can be AUTO, ALWAYS, NEVER, or TUNNEL. The applicable values are the following:

<b>AUTO</b>	Default setting. If the URL is an HTTPS URL, the adapter will use the TUNNEL option. If the URL is an HTTP URL, the component will use the NEVER option.
<b>ALWAYS</b>	The connection is always SSL enabled.
<b>NEVER</b>	The connection is not SSL enabled.
<b>TUNNEL</b>	The connection is through a tunneling proxy. The proxy server opens a connection to the remote host and traffic flows back and forth through the proxy.

## ProxyExceptions

A semicolon separated list of destination hostnames or IPs that are exempt from connecting through the ProxyServer .

## Data Type

string

## Default Value

""

## Remarks

The [ProxyServer](#) is used for all addresses, except for addresses defined in this property. Use semicolons to separate entries.

Note that the adapter uses the system proxy settings by default, without further configuration needed; if you want to explicitly configure proxy exceptions for this connection, you need to set [ProxyAutoDetect](#) = false, and configure [ProxyServer](#) and [ProxyPort](#). To authenticate, set [ProxyAuthScheme](#) and set [ProxyUser](#) and [ProxyPassword](#), if needed.

## Logging

This section provides a complete list of the Logging properties you can configure in the connection string for this provider.

---

Property	Description
<a href="#">LogModules</a>	Core modules to be included in the log file.

---

## LogModules

Core modules to be included in the log file.

### Data Type

string

### Default Value

""

### Remarks

Only the modules specified (separated by ';') will be included in the log file. By default all modules are included.

See the [Logging](#) page for an overview.

## Schema

This section provides a complete list of the Schema properties you can configure in the connection string for this provider.

Property	Description
<a href="#">Location</a>	A path to the directory that contains the schema files defining tables, views, and stored procedures.

## Location

A path to the directory that contains the schema files defining tables, views, and stored procedures.

## Data Type

string

## Default Value

"%APPDATA%\\CData\\AmazonDynamoDB Data Provider\\Schema"

## Remarks

The path to a directory which contains the schema files for the adapter (.rsd files for tables and views, .rsb files for stored procedures). The folder location can be a relative path from the location of the executable. The [Location](#) property is only needed if you want to customize definitions (for example, change a column name, ignore a column, and so on) or extend the data model with new tables, views, or stored procedures.

If left unspecified, the default location is "%APPDATA%\\CData\\AmazonDynamoDB Data Provider\\Schema" with **%APPDATA%** being set to the user's configuration directory:

Platform	%APPDATA%
Windows	The value of the APPDATA environment variable

Mac	~/Library/Application Support
Linux	~/.config

## Miscellaneous

This section provides a complete list of the Miscellaneous properties you can configure in the connection string for this provider.

Property	Description
<a href="#">AutoDetectIndex</a>	A boolean indicating if secondary indexes should be automatically detected based on the query used.
<a href="#">FlattenArrays</a>	By default, nested arrays are returned as strings of JSON. The FlattenArrays property can be used to flatten the elements of nested arrays into columns of their own. Set FlattenArrays to the number of elements you want to return from nested arrays.
<a href="#">FlattenObjects</a>	Set FlattenObjects to true to flatten object properties into columns of their own. Otherwise, objects nested in arrays are returned as strings of JSON.
<a href="#">FlexibleSchema</a>	Set FlexibleSchema to true to scan for additional metadata on the query result set. Otherwise, the metadata will remain the same.
<a href="#">GenerateSchemaFiles</a>	Indicates the user preference as to when schemas should be generated and saved.
<a href="#">IgnoreTypes</a>	Removes support for the specified types. For example, Time. These types will then be reported as strings instead.
<a href="#">MaximumRequestRetries</a>	The maximum number of times to retry a request.
<a href="#">MaxRows</a>	Limits the number of rows returned rows when no

	aggregation or group by is used in the query. This helps avoid performance issues at design time.
Other	These hidden properties are used only in specific use cases.
Pagesize	The maximum number of results to return per page from Amazon DynamoDB.
Readonly	You can use this property to enforce read-only access to Amazon DynamoDB from the provider.
RetryWaitTime	The minimum number of milliseconds the provider will wait to retry a request.
RowScanDepth	The maximum number of rows to scan to look for the columns available in a table.
SeparatorCharacter	The character or characters used to denote hierarchy.
ThreadCount	The number of threads to use when selecting data via a parallel scan. Setting ThreadCount to 1 will disable parallel scans.
Timeout	The value in seconds until the timeout error is thrown, canceling the operation.
TypeDetectionScheme	Determines how to determine the data type of columns.
UseBatchWriteItemOperation	When enabled the provider will use BatchWriteItem operation for handling updates and inserts. By default, the provider uses ExecuteStatement/BatchExecuteStatement operation. You need to enable BatchWriteItem only when inserting/updating binary/binary-set data. ExecuteStatement/BatchExecuteStatement doesn't support manipulating binary fields.
UseConsistentReads	Whether to always use Consistent Reads or not when querying DynamoDb.
UserDefinedViews	A filepath pointing to the JSON configuration file containing

---

your custom views.

---

### `UseSimpleNames`

Boolean determining if simple names should be used for tables and columns.

---

## AutoDetectIndex

A boolean indicating if secondary indexes should be automatically detected based on the query used.

### Data Type

bool

### Default Value

true

### Remarks

In DynamoDB, you can use secondary indexes to more quickly select data from a given table. By default, we attempt to automatically detect an index to use based on the query criteria. However, this may not always be desirable. To turn off index detecting logic, set the property to false, or if you have control over the query and would prefer to specify the index yourself, use the `SecondaryIndexName` pseudo column to specify which index to use (if any).

## FlattenArrays

By default, nested arrays are returned as strings of JSON. The `FlattenArrays` property can be used to flatten the elements of nested arrays into columns of their own. Set `FlattenArrays` to the number of elements you want to return from nested arrays.

### Data Type

string

---

## Default Value

""

## Remarks

By default, nested arrays are returned as strings of JSON. The `FlattenArrays` property can be used to flatten the elements of nested arrays into columns of their own. This is only recommended for arrays that are expected to be short.

Set `FlattenArrays` to the number of elements you want to return from nested arrays. The specified elements are returned as columns. The zero-based index is concatenated to the column name. Other elements are ignored.

For example, you can return an arbitrary number of elements from an array of strings:

```
["FLOW-MATIC", "LISP", "COBOL"]
```

When `FlattenArrays` is set to 1, the preceding array is flattened into the following table:

Column Name	Column Value
languages_0	FLOW-MATIC

## FlattenObjects

Set `FlattenObjects` to true to flatten object properties into columns of their own. Otherwise, objects nested in arrays are returned as strings of JSON.

## Data Type

bool

## Default Value

true

## Remarks

Set `FlattenObjects` to true to flatten object properties into columns of their own. Otherwise, objects nested in arrays are returned as strings of JSON. The property name is concatenated onto the object name with an underscore to generate the column name.

For example, you can flatten the nested objects below at connection time:

```
[
  { "grade": "A", "score": 2 },
  { "grade": "A", "score": 6 },
  { "grade": "A", "score": 10 },
  { "grade": "A", "score": 9 },
  { "grade": "B", "score": 14 }
]
```

When `FlattenObjects` is set to true and `FlattenArrays` is set to 1, the preceding array is flattened into the following table:

Column Name	Column Value
grades_0_grade	A
grades_0_score	2

## FlexibleSchema

Set `FlexibleSchema` to true to scan for additional metadata on the query result set. Otherwise, the metadata will remain the same.

### Data Type

bool

### Default Value

true



## Remarks

Set FlexibleSchema to true to scan for additional metadata on the query result set. Otherwise, the metadata will remain the same.

## GenerateSchemaFiles

Indicates the user preference as to when schemas should be generated and saved.

### Possible Values

Never, OnUse, OnStart, OnCreate

### Data Type

string

### Default Value

"Never"

## Remarks

This property outputs schemas to .rsd files in the path specified by [Location](#).

Available settings are the following:

- Never: A schema file will never be generated.
- OnUse: A schema file will be generated the first time a table is referenced, provided the schema file for the table does not already exist.
- OnStart: A schema file will be generated at connection time for any tables that do not currently have a schema file.
- OnCreate: A schema file will be generated by when running a CREATE TABLE SQL query.

Note that if you want to regenerate a file, you will first need to delete it.

## Generate Schemas with SQL

When you set GenerateSchemaFiles to **OnUse**, the adapter generates schemas as you execute SELECT queries. Schemas are generated for each table referenced in the query.

When you set GenerateSchemaFiles to **OnCreate**, schemas are only generated when a CREATE TABLE query is executed.

## Generate Schemas on Connection

Another way to use this property is to obtain schemas for every table in your database when you connect. To do so, set GenerateSchemaFiles to **OnStart** and connect.

## IgnoreTypes

Removes support for the specified types. For example, Time. These types will then be reported as strings instead.

### Data Type

string

### Default Value

"Datetime,Date,Time"

### Remarks

Removes support for the specified types. For example, Time. These types will then be reported as strings instead.

## MaximumRequestRetries

The maximum number of times to retry a request.

### Data Type

string

### Default Value

"4"

## Remarks

MaximumRequestRetries is the maximum number of times the adapter will retry a request when the problem has been detected as temporary (errors like "unknown error", network issues, and exceeding the maximum threshold per table). In this case on the first retry the adapter will back off and wait for the amount of time designated by [RetryWaitTime](#). If that request fails, the adapter will double the time and then double again until the adapter has exhausted the available retries.

For example, if [RetryWaitTime](#) is set to 2 seconds and MaximumRequestRetries is set to 5, the wait times will be as follows: 0 -> 2 -> 4 -> 8 -> 16 -> 32.

## MaxRows

Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.

### Data Type

int

### Default Value

-1

## Remarks

Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.

## Other

These hidden properties are used only in specific use cases.

### Data Type

string

## Default Value

""

## Remarks

The properties listed below are available for specific use cases. Normal driver use cases and functionality should not require these properties.

Specify multiple properties in a semicolon-separated list.

## Integration and Formatting

DefaultColumnSize	Sets the default length of string fields when the data source does not provide column length in the metadata. The default value is 2000.
ConvertDateTimeToGMT	Determines whether to convert date-time values to GMT, instead of the local time of the machine.
RecordToFile=filename	Records the underlying socket data transfer to the specified file.

## Pagesize

The maximum number of results to return per page from Amazon DynamoDB.

## Data Type

int

## Default Value

2000

## Remarks

The Pagesize property affects the maximum number of results to return per page from Amazon DynamoDB. By default this property has the value '-1' which indicates that the

provider will try to get all the records without per page limit.

## Readonly

You can use this property to enforce read-only access to Amazon DynamoDB from the provider.

### Data Type

bool

### Default Value

false

### Remarks

If this property is set to true, the adapter will allow only SELECT queries. INSERT, UPDATE, DELETE, and stored procedure queries will cause an error to be thrown.

## RetryWaitTime

The minimum number of milliseconds the provider will wait to retry a request.

### Data Type

string

### Default Value

"2000"

### Remarks

The value of this property is doubled on every retry to determine how long to wait until the next retry. Specify the maximum number of retries with [MaximumRequestRetries](#).

## RowScanDepth

The maximum number of rows to scan to look for the columns available in a table.

### Data Type

int

### Default Value

50

### Remarks

The columns in a table must be determined by scanning table rows. This value determines the maximum number of rows that will be scanned.

Setting a high value may decrease performance. Setting a low value may prevent the data type from being determined properly, especially when there is null data.

## SeparatorCharacter

The character or characters used to denote hierarchy.

### Data Type

string

### Default Value

","

### Remarks

In order to flatten out structures such as Maps and List attributes in DynamoDB, we need some specifier that states what the separation is between those columns and other columns. If this value is "," and a column comes back with the name address.city, this indicates that there is a mapped attribute with a child called city. If your data has columns that already use a single period within the attribute name, set the SeparatorCharacter to a different character or characters.

## ThreadCount

The number of threads to use when selecting data via a parallel scan. Setting ThreadCount to 1 will disable parallel scans.

### Data Type

string

### Default Value

"4"

### Remarks

Parallel scans allow data to be retrieved faster by splitting up the retrieval process across multiple threads. This can greatly improve performance when scanning data in Amazon DynamoDB. However, this will also consume your read units for a table much faster than a single thread. Consider your available cores, bandwidth, and read units for your tables before increasing the ThreadCount.

## Timeout

The value in seconds until the timeout error is thrown, canceling the operation.

### Data Type

int

### Default Value

60

### Remarks

If Timeout = 0, operations do not time out. The operations run until they complete successfully or until they encounter an error condition.

If Timeout expires and the operation is not yet complete, the adapter throws an exception.

## TypeDetectionScheme

Determines how to determine the data type of columns.

### Possible Values

None, RowScan

### Data Type

string

### Default Value

"RowScan"

### Remarks

None	Setting <a href="#">TypeDetectionScheme</a> to None will return all columns as string type. Note: Even when set to None, the column names will still be scanned when Header=True.
RowScan	Setting <a href="#">TypeDetectionScheme</a> to RowScan will scan rows to heuristically determine the data type. The <a href="#">RowScanDepth</a> determines the number of rows to be scanned. If no value is specified, RowScan will be used by default.

## UseBatchWriteItemOperation

When enabled the provider will use BatchWriteItem operation for handling updates and inserts. By default, the provider uses ExecuteStatement/BatchExecuteStatement operation. You need to enable BatchWriteItem only when inserting/updating binary/binary-set data. ExecuteStatement/BatchExecuteStatement doesn't support manipulating binary fields.

### Data Type

bool



## Default Value

false

## Remarks

When enabled the provider will use BatchWriteItem operation for handling updates and inserts. By default, the provider uses ExecuteStatement/BatchExecuteStatement operation. You need to enable BatchWriteItem only when inserting/updating binary/binary-set data. ExecuteStatement/BatchExecuteStatement doesn't support manipulating binary fields.

## UseConsistentReads

Whether to always use Consistent Reads or not when querying DynamoDb.

## Data Type

bool

## Default Value

false

## Remarks

This parameter is not supported on global secondary indexes. If you scan or query using a secondary index, Consistent Reads will not be used even if the property is set to true.

## UserDefinedViews

A filepath pointing to the JSON configuration file containing your custom views.

## Data Type

string

## Default Value

""

## Remarks

User Defined Views are defined in a JSON-formatted configuration file called *UserDefinedViews.json*. The adapter automatically detects the views specified in this file.

You can also have multiple view definitions and control them using the UserDefinedViews connection property. When you use this property, only the specified views are seen by the adapter.

This User Defined View configuration file is formatted as follows:

- Each root element defines the name of a view.
- Each root element contains a child element, called **query**, which contains the custom SQL query for the view.

For example:

```
{
  "MyView": {
    "query": "SELECT * FROM Account WHERE MyColumn = 'value'"
  },
  "MyView2": {
    "query": "SELECT * FROM MyTable WHERE Id IN (1,2,3)"
  }
}
```

Use the UserDefinedViews connection property to specify the location of your JSON configuration file. For example:

```
"UserDefinedViews",
"C:\\Users\\yourusername\\Desktop\\tmp\\UserDefinedViews.json"
```

## UseSimpleNames

Boolean determining if simple names should be used for tables and columns.

### Data Type

bool

### Default Value

false

## Remarks

Amazon DynamoDB tables and columns can use special characters in names that are normally not allowed in standard databases. UseSimpleNames makes the adapter easier to use with traditional database tools.

Setting UseSimpleNames to true will simplify the names of tables and columns returned. It will enforce a naming scheme such that only alphanumeric characters and the underscore are valid for the displayed table and column names. Any nonalphanumeric characters will be converted to an underscore.

# TIBCO Product Documentation and Support Services

---

For information about this product, you can read the documentation, contact TIBCO Support, and join the TIBCO Community.

## How to Access TIBCO Documentation

Documentation for TIBCO products is available on the [TIBCO Product Documentation](#) website, mainly in HTML and PDF formats.

The [TIBCO Product Documentation](#) website is updated frequently and is more current than any other documentation included with the product.

## Product-Specific Documentation

The following documentation for this product is available on the [TIBCO® Data Virtualization](#) page.

- **Users**
  - TDV Getting Started Guide
  - TDV User Guide
  - TDV Web UI User Guide
  - TDV Client Interfaces Guide
  - TDV Tutorial Guide
  - TDV Northbay Example
- **Administration**
  - TDV Installation and Upgrade Guide
  - TDV Administration Guide
  - TDV Active Cluster Guide
  - TDV Security Features Guide
- **Data Sources**

TDV Adapter Guides

TDV Data Source Toolkit Guide (Formerly Extensibility Guide)

- **References**

TDV Reference Guide

TDV Application Programming Interface Guide

- **Other**

TDV Business Directory Guide

TDV Discovery Guide

- *TIBCO TDV and Business Directory Release Notes* Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

## How to Contact TIBCO Support

Get an overview of [TIBCO Support](#). You can contact TIBCO Support in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the [TIBCO Support](#) website.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to [TIBCO Support](#) website. If you do not have a user name, you can request one by clicking **Register** on the website.

## Release Version Support

TDV 8.5 is designated as a Long Term Support (LTS) version. Some release versions of TIBCO® Data Virtualization products are selected to be long-term support (LTS) versions. Defect corrections will typically be delivered in a new release version and as hotfixes or service packs to one or more LTS versions. See also

[https://docs.tibco.com/pub/tdv/general/LTS/tdv\\_LTS\\_releases.htm](https://docs.tibco.com/pub/tdv/general/LTS/tdv_LTS_releases.htm).

## How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the [TIBCO Ideas Portal](#). For a free registration, visit [TIBCO Community](#).

# Legal and Third-Party Notices

---

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, TIBCO logo, TIBCO O logo, ActiveSpaces, Enterprise Messaging Service, Spotfire, TERR, S-PLUS, and S+ are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle Corporation and/or its affiliates.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the

readme file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (<https://www.tibco.com/patents>) for details.

Copyright © 2002-2023 Cloud Software Group, Inc All Rights Reserved.