TIBCO® Data Virtualization

Elasticsearch Adapter Guide

Version 8.8.0 | October 2023

Contents

Elasticsearch Adapter

Elasticsearch Version Support

The adapter models Elasticsearch data as a read/write, relational database. The adapter can connect to Elasticsearch v2.2.0 and above via the REST API.

SQL Compliance

The SQL Compliance section shows the SQL syntax supported by the adapter and points out any limitations.

Getting Started

Connecting to Elasticsearch

Basic Tab shows how to authenticate to Elasticsearch 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 Elasticsearch Adapter

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

1. Unzip the tdv.elasticsearch.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.elasticsearch/tdv.elasticsearch.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 Elasticsearch
```

Basic Tab

Connecting to Elasticsearch Service

Set the following to connect to data:

- Server should be set to the IP Address or domain of the Elasticsearch instance. The Server could also be set to a comma-delimited list of node addresses or hostnames from a single cluster.

  ```
  Server=https://01.02.03.04
    OR
  Server=https://01.01.01.01:1234,https://02.02.02.02:5678
  ```

- Port should be set to the configured port for the Elasticsearch instance. If you include a port in a node specification for the Server property, that included port will take precedence over the specification for Port for that node only.

The adapter uses X-Pack Security for authentication and TLS/SSL encryption. You can prefix the server value with "https://" to connect using TLS/SSL.

Connecting to Amazon OpenSearch Service

Set the following to connect to data:

- Server should be set to the Endpoint URL for the Amazon ES instance.
- Port should be set to 443.
- AWSRegion should be set to the Amazon AWS region where the Elasticsearch instance is being hosted (the adapter will attempt to automatically identify the region based on the Server value).

The adapter uses X-Pack Security for authentication and TLS/SSL encryption.

Note: Requests are signed using AWS Signature Version 4.

Authenticating to Elasticsearch

In addition to standard connection properties, select one of the below authentication methods to authenticate.

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.

Standard Authentication

Set the AuthScheme to Basic, and set User and Password properties and/or use PKI (public key infrastructure) to authenticate. Once the adapter is connected, X-Pack performs user authentication and grants role permissions based on the realms you have configured.

To use PKI, set the SSLClientCert, SSLClientCertType, SSLClientCertSubject, and SSLClientCertPassword properties.

Note: TLS/SSL and client authentication must be enabled on X-Pack to use PKI.

Securing Elasticsearch Connections

To enable TLS/SSL in the adapter, prefix the Server value with 'https://'.

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 (optional): Only if required when you assume a role in another account.

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**.
- 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.
- 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 (optional): Only if required when you assume a role in another account.

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

Kerberos

Please see Using Kerberos for details on how to authenticate with Kerberos.

API Key

To authenticate using APIKey set the following:

- AuthScheme: Set this to APIKey.
- APIKey: Set this to APIKey returned from Elasticsearch.
- APIKeyId: Set this to the Id returned alongside APIKey.

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 Elasticsearch 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**.

Using Kerberos

This section shows how to use the adapter to authenticate using Kerberos.

Kerberos

To authenticate to Elasticsearch using Kerberos, set the following properties:

- AuthScheme: Set this to **NEGOTIATE**.
- KerberosKDC: Set this to the host name or IP Address of your Kerberos KDC machine.
- KerberosRealm: Set this to **the realm of the Elasticsearch Kerberos principal**. This will be the value after the '@' symbol (for instance, EXAMPLE.COM) of the **principal value** (for instance, HTTP/MyHost@EXAMPLE.COM).
- KerberosSPN: Set this to the service and host of the Elasticsearch Kerberos Principal.

This is the value prior to the '@' symbol (for instance, HTTP/MyHost) of the principal value (for instance, HTTP/MyHost@EXAMPLE.COM).

Retrieve the Kerberos Ticket

You can use one of the following options to retrieve the required Kerberos ticket.

MIT Kerberos Credential Cache File

This option enables you to use the MIT Kerberos Ticket Manager or kinit command to get tickets. Note that you do not need to set the User or Password connection properties with this option.

1. Ensure that you have an environment variable created called **KRB5CCNAME**.
2. Set the **KRB5CCNAME** environment variable to a path pointing to your credential cache file (for instance, *C:\krb_cache\krb5cc_0* or */tmp/krb5cc_0*). This file is created when generating your ticket with MIT Kerberos Ticket Manager.
3. To obtain a ticket, open the MIT Kerberos Ticket Manager application, click **Get Ticket**, enter your principal name and password, then click **OK**. If successful, ticket information appears in Kerberos Ticket Manager and is stored in the credential cache file.
4. Now that you have created the credential cache file, the adapter uses the cache file to obtain the Kerberos ticket to connect to Elasticsearch.

As an alternative to setting the **KRB5CCNAME** environment variable, you can directly set the file path using the KerberosTicketCache property. When set, the adapter uses the specified cache file to obtain the Kerberos ticket to connect to Elasticsearch.

Keytab File

If the **KRB5CCNAME environment variable has not been set**, you can retrieve a Kerberos ticket using a Keytab File. To do so, set the User property to the desired username and set the KerberosKeytabFile property to a file path pointing to the keytab file associated with the user.

User and Password

If both the **KRB5CCNAME** environment variable and the KerberosKeytabFile property have not been set, you can retrieve a ticket using a user and password combination. To do this, set the User and Password properties to the user/password combination that you use to authenticate with Elasticsearch.

Cross-Realm

More complex Kerberos environments may require cross-realm authentication where multiple realms and KDC servers are used (e.g., where one realm/KDC is used for user authentication and another realm/KDC is used for obtaining the service ticket).

In such an environment, set the KerberosRealm and KerberosKDC properties to the values required for user authentication. Also set the KerberosServiceRealm and KerberosServiceKDC properties to the values required to obtain the service ticket.

Fine-Tuning Data Access

Fine Tuning Data Access

You can use the following properties to gain greater control over Elasticsearch 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 persist your changes to column data types, for example. You can set this property to "OnStart" to generate schema files for all tables in your database at connection. The resulting schemas are based on the connection properties you use to configure Automatic Schema Discovery.
  Or, you can set this property to "OnUse" to generate schemas based on a query.
  To use the resulting schema files, set the Location property to the folder containing the schemas.
- QueryPassthrough: This property enables you to use Elasticsearch's Search DSL language instead of SQL.
- RowScanDepth: This property determines the number of rows that will be scanned to detect column data types when generating table metadata. This property applies if you are working with the dynamic schemas generated from Automatic Schema Discovery or if you are using QueryPassthrough.

Custom URLs

If a custom URL is required, using the form [Server]:[Port]/[URLPathPrefix], the 'URLPathPrefix' value can be specified via the Other property. For example: URLPathPrefix=myprefix

The adapter will use the specified path prefix to build the URL required for connecting to the Elasticsearch API endpoints.

Querying Multiple Indices

Querying Multiple Indices

Multiple indices can be queried by executing a query using one of the following formats:

- **Query all indices via the _all view**: SELECT * FROM [_all]
- **Query a list of indices**: SELECT * FROM [index1,index2,index3]
- **Query indices matching a wildcard pattern**: SELECT * FROM [index*]

Note, index lists can contain wildcards and indices can be excluded by prefixing an index with '-'. For example: SELECT * FROM [index*,-index3]

Performance

Fine Tuning Performance

- PageSize: This property enables you to optimize performance based on your resource provisioning.
  Paging has an impact on sorting performance in a distributed system, as each shard must first sort results before submitting them to the coordinating server.
  By default, the adapter requests a page size of 10,000. This is the default *index.max_result_window* setting in Elasticsearch.
- MaxResults: This property sets a limit on the results for queries at connection time, without requiring that you specify a LIMIT clause.
  By default, this is the same value as the *index.max_result_window* setting in

Elasticsearch.

If you are using the Scroll API, set ScrollDuration instead.

- ScrollDuration: This property specifies how long the server should keep the search context alive. Setting this property to a nonzero value and time unit enables the Scroll API.

Changelog

General Changes

Date	Build Number	Change Type	Description
04/25/2023	8515	General	**Removed** • Removed support for the SELECT INTO CSV statement. The core code doesn't support it anymore.
3/15/2023	8474	Elasticsearch	**Added** • Added the CreateIndex stored procedure. Can be used to create indices in the target Elasticsearch cluster.
1/06/2023	8406	Elasticsearch	**Removed** • Added the UseFullyQualifiedNestedTableName connection property. When using the Relational mode of the Datamodel connection property, UseFullyQualifiedNestedTableName controls whether or not the relational tables modeled for nested documents are named with a full representation of

Date	Build Number	Change Type	Description
			their path in the parent, indexed document.
12/14/2022	8383	General	**Changed** • Added the Default column to the sys_ procedureparameters table.
12/09/2022	8378	Elasticsearch	**Removed** • Removed the FileLocation parameter from CreateSchema. The Location property must be used to set the output directory for created schemas.
09/30/2022	8308	General	**Changed** • Added the IsPath column to the sys_ procedureparameters table.
08/17/2022	8264	General	**Changed** • We now support handling the keyword "COLLATE" as standard function name as well.
06/17/2022	8203	Elasticsearch	**Added** • Added support for specification of multiple nodes from the same cluster in the Server connection property. Driver will cycle through these nodes as the destinations for its requests to Elasticsearch.
06/02/2022	8188	Elasticsearch	**Added** • Added support for the _delete_by_ query API endpoint.

Date	Build Number	Change Type	Description
05/06/2022	8161	Elasticsearch	**Added** • Added support for Elasticsearch 8.0+.
09/02/2021	7915	General	**Added** • Added support for the STRING_SPLIT table-valued function in the CROSS APPLY clause.
08/07/2021	7889	General	**Changed** • Added the KeySeq column to the sys_foreignkeys table.
08/06/2021	7888	General	**Changed** • Added the new sys_primarykeys system table.
04/23/2021	7785	General	**Added** • Added support for handling client side formulas during insert / update. For example: UPDATE Table SET Col1 = CONCAT(Col1, " - ", Col2) WHERE Col2 LIKE 'A%'
04/23/2021	7783	General	**Changed** • Updated how display sizes are determined for varchar primary key and foreign key columns so they will match the reported length of the column.
04/16/2021	7776	General	**Added** • Non-conditional updates between two columns is now available to all drivers.

Date	Build Number	Change Type	Description
			For example: UPDATE Table SET Col1=Col2 **Changed** • Reduced the length to 255 for varchar primary key and foreign key columns. **Changed** • Updated implicit and metadata caching to improve performance and support for multiple connections. Old metadata caches are not compatible - you need to generate new metadata caches if you are currently using CacheMetadata. **Changed** • Updated index naming convention to avoid duplicates.
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.

Searching with SQL

Elasticsearch is a document-oriented database that provides high performance searching, flexibility, 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 an Elasticsearch database.

The adapter models Elasticsearch objects into relational tables and translates SQL queries into Elasticsearch queries to get the requested data. See Schema Mapping for more details on how Elasticsearch objects are mapped to tables to generate schemas. See Query Mapping for more details on how various Elasticsearch operations are represented as SQL.

The Automatic Schema Discovery scheme automatically finds the data types by retrieving the mapping for the Elasticsearch type. You can use RowScanDepth, FlattenArrays, and FlattenObjects to control the relational representation of the collections in Elasticsearch.

Optionally, you can use Custom Schema Definitions to project your chosen relational structure on top of a Elasticsearch object. This allows you choose your own column names, their data types, and the location of their values in the collection.

When GenerateSchemaFiles is set, you can persist schemas for all collections in the database or for the results of SELECT queries.

Schema Mapping

The Elasticsearch Adapter models the Elasticsearch REST APIs as relational tables and stored procedures that can be accessed with standard SQL. This enables access from standards-based tools.

The table definitions are dynamically retrieved. When you connect, the adapter connects to Elasticsearch and retrieves the schemas, list of tables, and the metadata for the tables by querying the Elasticsearch REST server. Any changes to the remote data are immediately reflected in your queries.

The following table maps Elasticsearch concepts to relational ones:

Elasticsearch Versions 6 and Above:

Elasticsearch Concept	SQL Concept
Index	Table
Alias	View
Document	Row (each document is a row and the document's JSON structure is represented as columns)
Field	Column

Note: Starting in Elasticsearch 6, indices are limited to a single type. Therefore the type is no longer treated as a table, since an index and type have a one-to-one relation. Types are hidden and used internally where necessary to issue the proper request to Elasticsearch.

Elasticsearch Versions Prior to Version 6:

Elasticsearch Concept	SQL Concept
Index	Schema
Type	Table
Alias	View
Document	Row (each document is a row and the document's JSON structure is represented as columns)
Field	Column

Parent-Child Relationships

Elasticsearch contains the ability to establish parent-child relationships. This relationship maps closely to SQL JOIN functionality. The adapter models these parent-child relationships in a way to enable the ability to perform JOIN queries.

Elasticsearch Versions 6 and Above:

In version 6 and above of Elasticsearch, relationships are established by using the join datatype. Included in this functionality is the ability to define multiple children for a single parent and to create multiple levels of relations.

The adapter supports all of these relationships and will generate a separate table for each relation in Elasticsearch. The table name will be in the form: [index]_[relation].

All child tables will have an additional column containing the parent table id. The column name will be in the form: _[parent_table]_id. This column is a foreign key to the _id column of the parent table and can be used to perform SQL JOIN queries.

When querying these tables individually, filtering logic is pushed to the server to improve performance by only returning the data relevant to the table selected.

Elasticsearch Versions Prior to Version 6:

In versions prior to 6, a relationship is established between two types via a _parent field. This creates a single parent-child relationship.

The tables identified in this parent-child relationship do not change (they are still based on the Elasticsearch type). However the child table will have an additional column containing the parent id. The column name will be in the form: _[parent_table]_id. This column is a foreign key to the _id column of the parent table and can be used to perform SQL JOIN queries.

Raw Data

Below is the raw data used throughout this chapter. Following is the mapping for the "insured" table (index):

```
{
  "insured": {
    "mappings": {
      "properties": {
        "name": { "type":"string" },
        "address": {
          "street": { "type":"string" },
          "city": { "type":"string" },
          "state": { "type":"string" }
        },
        "insured_ages": { "type": "integer" },
        "vehicles": {
          "type": "nested",
          "properties": {
            "year": { "type":"integer" },
            "make": { "type":"string" },
            "model": { "type":"string" },
            "body_style" { "type": "string" }
          }
        }
      }
    }
  }
}
```

The following is the sample data set for the "insured" table (index):

```
{
  "hits": {
    "total": 2,
    "max_score": 1,
    "hits": [
      {
        "_index": "insured",
        "_type": "_doc",
        "_id": "1",
        "_score": 1,
        "_source": {
          "name": "John Smith",
          "address": {
            "street": "Main Street",
            "city": "Chapel Hill",
            "state": "NC"
          },
          "insured_ages": [ 17, 43, 45 ],
          "vehicles": [
            {
              "year": 2015,
              "make": "Dodge",
              "model": "RAM 1500",
              "body_style": "TK"
            },
            {
              "year": 2015,
              "make": "Suzuki",
              "model": "V-Strom 650 XT",
              "body_style": "MC"
            },
            {
              "year": 1992,
              "make": "Harley Davidson",
              "model": "FXR",
              "body_style": "MC"
            }
          ]
        }
      },
      {
        "_index": "insured",
        "_type": "_doc",
        "_id": "2",
        "_score": 1,
        "_source": {
          "name": "Joseph Newman",
          "address": {
            "street": "Oak Street",
            "city": "Raleigh",
            "state": "NC"
          },
          "insured_ages": [ 23, 25 ],
          "vehicles": [
            {
              "year": 2010,
              "make": "Honda",
              "model": "Accord",
              "body_style": "SD"
            },
            {
              "year": 2008,
              "make": "Honda",
              "model": "Civic",
              "body_style": "CP"
            }
          ]
        }
      }
    ]
  }
}
```

Automatic Schema Discovery

The adapter automatically infers a relational schema by retrieving the mapping of the Elasticsearch type. The columns and data types are generated from the retrieved mapping.

Detecting Arrays

Any field within Elasticsearch can be an array of values, but this is not explicitly defined within the mapping. To account for this, the adapter will query the data to detect if any fields contain arrays. The number of Elasticsearch documents retrieved during this array scanning is based on the RowScanDepth property.

Elasticsearch nested types are special types that denote an array of objects and thus will always be treated as such when generating the metadata.

Detecting Columns

The columns identified during the discovery process depend on the FlattenArrays and FlattenObjects properties.

Example Data Set

To provide an example of how these options work, consider the following mapping (where 'insured' is the name of the table):

```
{
  "insured": {
    "properties": {
      "name": { "type":"string" },
      "address": {
        "street": { "type":"string" },
        "city": { "type":"string" },
        "state": { "type":"string" }
      },
      "insured_ages": { "type": "integer" },
      "vehicles": {
        "type": "nested",
        "properties": {
          "year": { "type":"integer" },
          "make": { "type":"string" },
          "model": { "type":"string" },
          "body_style" { "type": "string" }
        }
      }
    }
  }
}
```

Also consider the following example data for the above mapping:

```
{
  "_source": {
    "name": "John Smith",
    "address": {
      "street": "Main Street",
      "city": "Chapel Hill",
      "state": "NC"
    },
    "insured_ages": [ 17, 43, 45 ],
    "vehicles": [
      {
        "year": 2015,
        "make": "Dodge",
        "model": "RAM 1500",
        "body_style": "TK"
      },
      {
        "year": 2015,
        "make": "Suzuki",
        "model": "V-Strom 650 XT",
        "body_style": "MC"
      },
      {
        "year": 2012,
        "make": "Honda",
        "model": "Accord",
        "body_style": "4D"
      }
    ]
  }
}
```

Using FlattenObjects

If FlattenObjects is set, all nested objects will be flattened into a series of columns. The above example will be represented by the following columns:

Column Name	Data Type	Example Value
name	String	John Smith
address.street	String	Main Street
address.city	String	Chapel Hill
address.state	String	NC
insured_ages	String	[17, 43, 45]
vehicles	String	[{ "year": "2015", "make": "Dodge", ... }, { "year": "2015", "make": "Suzuki", ... }, { "year": "2012", "make": "Honda", ... }]

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 the following:

```
{street: "Main Street", city: "Chapel Hill", state: "NC"}
```

See JSON Functions for more details on working with JSON aggregates.

Using FlattenArrays

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. It is best to leave unbounded arrays as they are and piece out the data for them as needed using JSON Functions.

Note: Only the top-most array will be flattened. Any subarrays will be represented as the entire array.

The FlattenArrays property can be set to 3 to represent the arrays in the example above as follows (this example is with FlattenObjects not set):

Column Name	Data Type	Example Value
insured_ages	String	[17, 43, 45]
insured_ages.0	Integer	17
insured_ages.1	Integer	43
insured_ages.2	Integer	45
vehicles	String	[{ "year": "2015", "make": "Dodge", ... }, { "year": "2015", "make": "Suzuki", ... }, { "year": "2012", "make": "Honda", ... }]
vehicles.0	String	{ "year": "2015", "make": "Dodge", "model": "RAM 1500", "body_style": "TK" }
vehicles.1	String	{ "year": "2015", "make": "Suzuki", "model": "V-Strom 650 XT", "body_style": "MC" }
vehicles.2	String	{ "year": "2012", "make": "Honda", "model": "Accord", "body_style": "4D" }

Using Both FlattenObjects and FlattenArrays

If FlattenObjects is set along with FlattenArrays (set to 1 for brevity), the vehicles field will be represented as follows:

Column Name	Data Type	Example Value
vehicles	String	[{ "year": "2015", "make": "Dodge", ... }, { "year": "2015", "make": "Suzuki", ... }, { "year": "2012", "make": "Honda", ... }]
vehicles.0.year	String	2015
vehicles.0.make	String	Dodge
vehicles.0.model	String	RAM 1500
vehicles.0.body_style	String	TK

Parsing Hierarchical Data

The adapter offers three basic configurations to model documents as tables, described in the following sections. The adapter will parse the Elasticsearch document and identify the nested documents.

- Flattened Documents Model: Implicitly join nested documents into a single table.
- Relational Model: Model nested documents as individual tables containing a primary key and a foreign key that links to the parent document.
- Top-Level Document Model: Model a top-level view of an Elasticsearch document. Nested documents are returned as JSON strings.

See Searching with SQL to configure column discovery or customize the detected schemas.

Flattened Documents Model

For users who need access to the entirety of their nested Elasticsearch data, flattening the data into a single table is the best option. The adapter will use streaming and only parses the Elasticsearch data once per query in this mode.

Joining Object Arrays into a Single Table

With DataModel set to "FlattenedDocuments", nested documents will behave as separate tables and act in the same manner as a SQL JOIN. Any nested documents, at the same height (e.g. sibling documents), will be treated as a SQL CROSS JOIN.

Example

Below is a sample query and the results, based on the sample document in Raw Data. This implicitly JOINs the insured document with the nested vehicles document.

Query

The following query drills into the nested documents in each insured document.

```
SELECT
  "_id",
  "name",
  "address.street" AS address_street,
  "address.city.first" AS address_city,
  "address.state.last" AS address_state,
  "insured_ages",
  "year",
  "make",
  "model",
  "body_style",
  "_insured_id",
  "_vehicles_c_id"
FROM
  "insured"
```

Results

_id	name	address_street	address_city	address_state	insured_ages	year	make	model	body_style	_insured_id	_vehicles_c_id
1	John Smith	Main Street	Chapel Hill	NC	[17, 43, 45]	2015	Dodge	RAM 1500	TK	1	1
1	John Smith	Main Street	Chapel Hill	NC	[17, 43, 45]	2015	Suzuki	V-Strom 650 XT	MC	1	2
1	John Smith	Main Street	Chapel Hill	NC	[17, 43, 45]	1992	Harley Davidson	FXR	MC	1	3
2	Joseph Newman	Oak Street	Raleigh	NC	[23, 25]	2010	Honda	Accord	SD	2	4
2	Joseph Newman	Oak Street	Raleigh	NC	[23, 25]	2008	Honda	Civic	CP	2	5

See Also

- Automatic Schema Discovery: Configure the columns reported in the table schemas.
- FreeForm;: Use dot notation to select nested data.
- VerticalFlattening;: Access nested object arrays as separate tables.
- JSON Functions: Manipulate the data returned to perform client-side aggregation and transformations.

Top-Level Document Model

Using a top-level document view of the Elasticsearch data provides ready access to top-level elements. The adapter returns nested elements in aggregate, as single columns.

One aspect to consider is performance. You forego the time and resources to process and parse nested elements -- the adapter parses the returned data once, using streaming to read the JSON data. Another consideration is your need to access any data stored in nested parent elements, and the ability of your tool or application to process JSON.

Modeling a Top-Level Document View

With DataModel set to "Document" (the default), the adapter scans only the top-level object by default. The top-level object elements are available as columns due to the default object flattening. Nested objects are returned as aggregated JSON.

Example

Below is a sample query and the results, based on the sample document in Raw Data. The query results in a single "insured" table.

Query

The following query pulls the top-level object elements and the vehicles array into the results.

```
SELECT
  "_id",
  "name",
  "address.street" AS address_street,
  "address.city" AS address_city,
  "address.state" AS address_state,
  "insured_ages",
  "vehicles"
FROM
  "insured"
```

Results

With a document view of the data, the address object is flattened into 3 columns (when FlattenObjects set to true) and the _id, name, insured_ages, and vehicles elements are returned as individual columns, resulting in a table with 7 columns.

_id	name	address_street	address_city	address_state	insured_ages	vehicles
1	John Smith	Main Street	Chapel Hill	NC	[17, 43, 45]	`[{"year":2015,"make":"Dodge","model":"RAM 1500","body_style":"TK"},{"year":2015,"make":"Suzuki","model":"V-Strom 650 XT","body_style":"MC"},{"year":1992,"make":"Harley Davidson","model":"FXR","body_style":"MC"}]`
2	Joseph Newman	Oak Street	Raleigh	NC	[23, 25]	`[{"year":2010,"make":"Honda","model":"Accord","body_style":"SD"},{"year":2008,"make":"Honda","model":"Civic","body_style":"CP"}]`

See Also

- Automatic Schema Discovery: Configure column discovery with horizontal flattening.
- FreeForm;: Use dot notation to select nested data.
- VerticalFlattening;: Access nested object arrays as separate tables.
- JSON Functions: Manipulate the data returned to perform client-side aggregation and

transformations.

Relational Model

The Elasticsearch Adapter can be configured to create a relational model of the data, treating nested documents as individual tables containing a primary key and a foreign key that links to the parent document. This is particularly useful if you need to work with your Elasticsearch data in existing BI, reporting, and ETL tools that expect a relational data model.

Joining Nested Arrays as Tables

With DataModel set to "Relational", any JOINs are controlled by the query. Any time you perform a JOIN query, the Elasticsearch index will be queried once for each table (nested document) included in the query.

Example

Below is a sample query against the sample document in Raw Data, using a relational model.

Query

The following query explicitly JOINs the insured and vehiclestables.

```
SELECT
  "insured"."_id",
  "insured"."name",
  "insured"."address.street" AS address_street,
  "insured"."address.city.first" AS address_city,
  "insured"."address.state.last" AS address_state,
  "insured"."insured_ages",
  "vehicles"."year",
  "vehicles"."make",
  "vehicles"."model",
  "vehicles"."body_style",
  "vehicles"."_insured_id",
  "vehicles"."_c_id"
FROM
  "insured"
JOIN
```

```
  "vehicles"
ON
  "insured"."_id" = "vehicles"."_insured_id"
```

Results

In the example query, each vehicle document is JOINed to its parent insured object to produce a table with 5 rows.

_id	name	address_street	address_city	address_state	insured_ages	year	make	model	body_style	_insured_id	_vehicles_c_id
1	John Smith	Main Street	Chapel Hill	NC	[17, 43, 45]	2015	Dodge	RAM 1500	TK	1	1
1	John Smith	Main Street	Chapel Hill	NC	[17, 43, 45]	2015	Suzuki	V-Strom 650 XT	MC	1	2
1	John Smith	Main Street	Chapel Hill	NC	[17, 43, 45]	1992	Harley Davidson	FXR	MC	1	3
2	Joseph Newman	Oak Street	Raleigh	NC	[23, 25]	2010	Honda	Accord	SD	2	4
2	Joseph Newman	Oak Street	Raleigh	NC	[23, 25]	2008	Honda	Civic	CP	2	5

See Also

- Automatic Schema Discovery: Configure the columns reported in the table schemas.
- FreeForm;: Use dot notation to select nested data.
- VerticalFlattening;: Access nested object arrays as separate tables.
- JSON Functions: Manipulate the data returned to perform client-side aggregation and transformations.

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

DOCUMENT

The DOCUMENT function can be used to retrieve the entire document as a JSON string. See the following query and its result as an example:

```
SELECT DOCUMENT(*) FROM Employee;
```

The query above will return the entire document as shown.

```
{
  "_index": "megacorp",
  "_type": "employee",
  "_id": "2",
  "_score": 1,
  "_source": {
    "first_name": "Jane",
    "last_name": "Smith",
    "age": 32,
    "about": "I like to collect rock albums",
    "interests": [
      "music"
    ]
  }
}
```

Query Mapping

This section describes how SQL statements are interpreted and translated into Elasticsearch queries. Examples are also provided to explain the behavior of various queries.

Query/Filter Context and Scoring

When the *_score* column is selected, scoring will be requested by issuing a query context request, which scores the quality of the search results. By default, results are returned in descending order based on the calculated *_score*. An ORDER BY clause can be specified to change the order of the returned results.

When the *_score* column is not selected, a filter context will be sent, in which case Elasticsearch will not compute scores. The results for these queries will be returned in arbitrary order unless an ORDER BY clause is explicitly specified.

Text Matching and Search

Analyzed fields in Elasticsearch are stored in an inverted index after they are run through an analyzer. Analyzers are customizable and thus can perform a variety of different filters on the data prior to storing them in the inverted index. For example, the default Elasticsearch analyzer will lowercase all the terms.

To demonstrate this point, an analyzed field in Elasticsearch was created with a value of 'Bike'. After being analyzed, the value will be stored in the inverted index (using the default analyzer) as 'bike'. A non-analyzed field, on the other hand, would not analyze the search value and thus would be stored as 'Bike'.

When performing searches, some Elasticsearch query types run the search value through an analyzer (which will make the search case insensitive) and some do not (making the search case sensitive). Additionally, the default analyzer breaks up fields containing multiple words into separate terms. When performing searches on these fields, Elasticsearch may return records that contain the same words but in a different order. For example, a search is performed using a value of 'blue sky' but a record with 'sky blue' is returned.

To work around these case-sensitivity and ordering issues, the Elasticsearch Adapter will identify the column as analyzed or non-analyzed and will issue the appropriate Elasticsearch query based on the specified operator (such as =) and the search value.

Equals and Not Equals

Where clauses that contain an equals (=) or not equals (!= or <>) filter issue different Elasticsearch queries depending upon the column and data used. Analyzed and non-analyzed columns behave differently and thus different Elasticsearch queries are generated to provide the best search functionality. Additionally, string values generate different query types depending upon whether they contain empty space or not. Below is an explanation of the rules and behavior for the varying cases.

Analyzed Columns

Analyzed columns are stored after being run through an analyzer. As a result of that, the search values specified will be run through an analyzer on the Elasticsearch server prior to the search. This makes the searches case-insensitive (provided the analyzer used handles casing).

WHERE Clause Examples	Elasticsearch Query Type
WHERE analyzed_column='value'	Query String Query
WHERE analyzed_column='value with spaces'	Match Phrase Query

Non-Analyzed Columns

Non-analyzed columns are stored without being run through an analyzer. Thus, non-analyzed columns are case sensitive and thus search values specified for these columns are case sensitive. If the search value is a single word, the adapter will check the filter with the original casing specified along with three common forms: uppercase, lowercase, and capitalized. If the search value contains multiple words, the search value will be sent as-is and thus is case sensitive.

WHERE Clause Examples	Elasticsearch Query Type
WHERE nonanalyzed_column='myValue'	Query String Query: Four cases are checked - myValue OR MYVALUE OR myvalue OR Myvalue
WHERE nonanalyzed_column='value with spaces'	Wildcard Query

IN and NOT IN

The IN and NOT IN operators function very similarly to the equals and not equals operators.

WHERE Clause Examples	Behavior
WHERE column IN ('value')	Treated as: column='value'
WHERE column NOT IN ('value')	Treated as: column!='value'
WHERE column IN ('value1', 'value2')	Treated as: column='value1' OR column='value2'
WHERE column NOT IN ('value1', 'value2')	Treated as: column!='value1' AND column!='value2'

LIKE and NOT LIKE

The LIKE and NOT LIKE operators allow the use of wildcard characters. The percent sign (%) represents zero, one, or multiple characters. The underscore (_) represents a single character (in which the character must be present).

WHERE Clause Examples	Behavior
WHERE column LIKE 'value'	Treated as: column='value'
WHERE column NOT LIKE 'value'	Treated as: column!='value'
WHERE analyzed_column LIKE 'v_lu%'	Query String Query with wildcards
WHERE nonanalyzed_column LIKE 'v_lu%'	Wildcard Query with wildcards

Aggregate Filtering

Aggregate data may consist of JSON objects or arrays (both primitive and object arrays).

JSON objects and arrays of objects will be treated as raw strings and all filtering will be performed by the adapter. Therefore an equals operation must match the entire JSON aggregate to return a result, unless a CONTAINS or LIKE operation is used.

If JSON objects are flattened into individual columns (via FlattenObjects and FlattenArrays), the column for the specific JSON field will be treated as individual columns. Thus the data type will be that as contained in the Elasticsearch mapping and all filters will be pushed to the server (where applicable).

JSON primitive array aggregates will also be treated as raw strings by default and filters will be performed by the adapter. To filter data based on whether a primitive array contains a single value, the INARRAY function can be used (e.g. INARRAY(column) = 'value'). When performing a search on array fields, Elasticsearch looks at each value individually within an array. Thus when the INARRAY function is specified in a WHERE clause, the filter will be pushed to the server which performs a search within an array.

Primitive arrays may consist of different data types, such as strings or ints. Therefore the INARRAY function supports comparison operators applicable to the data type within the Elasticsearch mapping for the field. For example, INARRAY(int_array) > 5, will return all rows of data in which the int_array contains a value greater than 5. Supported comparison operators include the use of the LIKE operator for string arrays.

Custom Schema Definitions

View schemas persist the relational structure the adapter infers for Elasticsearch types and queries. To provide an example of how custom schemas work, we will use the below mapping (where 'insured' is the name of the table).

```
{
  "insured": {
    "properties": {
      "name": { "type":"string" },
      "address": {
        "street": { "type":"string" },
        "city": { "type":"string" },
        "state": { "type":"string" }
      },
      "insured_ages": { "type": "integer" },
      "vehicles": {
        "type": "nested",
        "properties": {
          "year": { "type":"integer" },
          "make": { "type":"string" },
          "model": { "type":"string" },
          "body_style" { "type": "string" }
        }
      }
    }
  }
}
```

Also, consider the following example data for the above mapping:

```
{
  "_source": {
    "name": "John Smith",
    "address": {
      "street": "Main Street",
      "city": "Chapel Hill",
      "state": "NC"
    },
    "insured_ages": [ 17, 43, 45 ],
    "vehicles": [
      {
        "year": 2015,
        "make": "Dodge",
        "model": "RAM 1500",
        "body_style": "TK"
      },
      {
        "year": 2015,
        "make": "Suzuki",
        "model": "V-Strom 650 XT",
        "body_style": "MC"
      },
      {
        "year": 2012,
        "make": "Honda",
        "model": "Accord",
        "body_style": "4D"
      }
    ]
  }
}
```

Defining a Custom Schema

Schemas persisted when GenerateSchemaFiles is set are placed into the folder specified by the Location property. For example, set GenerateSchemaFiles to "OnUse" and execute a

SELECT query:

```
SELECT * FROM insured
```

You can then change column behavior in the resulting schema. The following schema uses the *other:xPath* 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 *es_index* and *es_type* attributes specify the Elasticsearch index and type to retrieve. The *es_index* and *es_type* attributes give you the flexibility to use multiple schemas for the same type. If *es_type* is not specified, the filename determines the collection that is parsed.

Below is an example is an example of the column behavior markup. You can find a complete schema in Custom Schema Example.

```
  <rsb:script xmlns:rsb="http://www.rssbus.com/ns/rsbscript/2">

    <rsb:info title="StaticInsured" description="Custom Schema for the
Elasticsearch insured data set.">
      <!-- Column definitions -->
      <attr name="_id"                          xs:type="string"
other:xPath="_id"
 other:sourceField="_id"                  other:analyzed="true"  />
      <attr name="_score"                       xs:type="double"
other:xPath="_score"
 other:sourceField="_score"               other:analyzed="true"  />
      <attr name="name"                         xs:type="string"
other:xPath="_source/name"
 other:sourceField="name"                 other:analyzed="true"  />
      <attr name="address.street"               xs:type="string"
other:xPath="_source/address/street"
 other:sourceField="address.street"      other:analyzed="true"  />
      <attr name="address.city"                 xs:type="string"
other:xPath="_source/address/city"
 other:sourceField="address.city"        other:analyzed="true"  />
      <attr name="address.state"                xs:type="string"
other:xPath="_source/address/state"
 other:sourceField="address.state"       other:analyzed="true"  />
      <attr name="insured_ages"                 xs:type="string"
other:xPath="_source/insured_ages"
other:valueFormat="aggregate" other:sourceField="insured_ages"
other:analyzed="false" />
      <attr name="insured_ages.0"               xs:type="integer"
other:xPath="_source/insured_ages[0]"
 other:sourceField="insured_ages"        other:analyzed="false" />
      <attr name="vehicles"                     xs:type="string"
```

```
other:xPath="_source/vehicles"
other:valueFormat="aggregate" other:sourceField="vehicles"
other:analyzed="true"  />
      <attr name="vehicles.0.year"            xs:type="integer"
other:xPath="_source/vehicles[0]/year"
 other:sourceField="vehicles.year"      other:analyzed="true"  />
      <attr name="vehicles.0.make"            xs:type="string"
other:xPath="_source/vehicles[0]/make"
 other:sourceField="vehicles.make"      other:analyzed="true"  />
      <attr name="vehicles.0.model"           xs:type="string"
other:xPath="_source/vehicles[0]/model"
 other:sourceField="vehicles.model"     other:analyzed="true"  />
      <attr name="vehicles.0.body_style"      xs:type="string"
other:xPath="_source/vehicles[0]/body_style"
 other:sourceField="vehicles.body_style" other:analyzed="true"  />

      <input name="rows@next" desc="Internal attribute used for paging
through data."  />
    </rsb:info>


    <rsb:set attr="es_index" value="auto"/>
    <rsb:set attr="es_type"  value="insured"/>

  </rsb:script>
```

Custom Schema Example

In this section is a complete schema. The info section enables a relational view of an Elasticsearch object. For more details, see Custom Schema Definitions. The table below only supports SELECT commands. INSERT, UPDATE, and DELETE commands are not currently supported.

Use the *es_index* and *es_type* attributes to specify the name of the Elasticsearch type and index you want to retrieve and parse. You can use the *es_index* and *es_type* attributes to define multiple schemas for the same Elasticsearch type.

If *es_type* is not specified, the filename determines the Elasticsearch type that is parsed.

Copy the *rows@next* input as-is into your schema. The operations, such as *elasticsearchadoSelect*, are internal implementations and can also be copied as is.

```
<rsb:script xmlns:rsb="http://www.rssbus.com/ns/rsbscript/2">
  <rsb:info title="StaticInsured" description="Custom Schema for the
Elasticsearch insured data set.">
    <!-- Column definitions -->
    <attr name="_id"                         xs:type="string"
other:xPath="_id"
 other:sourceField="_id"                other:analyzed="true"  />
    <attr name="_score"                      xs:type="double"
other:xPath="_score"
 other:sourceField="_score"             other:analyzed="true"  />
    <attr name="name"                        xs:type="string"
other:xPath="_source/name"
 other:sourceField="name"               other:analyzed="true"  />
    <attr name="address.street"              xs:type="string"
other:xPath="_source/address/street"
 other:sourceField="address.street"     other:analyzed="true"  />
    <attr name="address.city"                xs:type="string"
other:xPath="_source/address/city"
 other:sourceField="address.city"       other:analyzed="true"  />
    <attr name="address.state"               xs:type="string"
other:xPath="_source/address/state"
 other:sourceField="address.state"      other:analyzed="true"  />
    <attr name="insured_ages"                xs:type="string"
other:xPath="_source/insured_ages"
other:valueFormat="aggregate" other:sourceField="insured_ages"
other:analyzed="false" />
    <attr name="insured_ages.0"              xs:type="integer"
other:xPath="_source/insured_ages[0]"
 other:sourceField="insured_ages"       other:analyzed="false" />
    <attr name="vehicles"                    xs:type="string"
other:xPath="_source/vehicles"
other:valueFormat="aggregate" other:sourceField="vehicles"
other:analyzed="true"  />
    <attr name="vehicles.0.year"             xs:type="integer"
other:xPath="_source/vehicles[0]/year"
 other:sourceField="vehicles.year"      other:analyzed="true"  />
    <attr name="vehicles.0.make"             xs:type="string"
other:xPath="_source/vehicles[0]/make"
 other:sourceField="vehicles.make"      other:analyzed="true"  />
    <attr name="vehicles.0.model"            xs:type="string"
other:xPath="_source/vehicles[0]/model"
 other:sourceField="vehicles.model"     other:analyzed="true"  />
    <attr name="vehicles.0.body_style"       xs:type="string"
other:xPath="_source/vehicles[0]/body_style"
 other:sourceField="vehicles.body_style" other:analyzed="true"  />
    <input name="rows@next" desc="Internal attribute used for paging
through data."  />
```

```
    </rsb:info>
    <rsb:set attr="es_index" value="auto"/>
    <rsb:set attr="es_type"  value="insured"/>
    <rsb:script method="GET">
      <rsb:call op="elasticsearchadoSelect">
        <rsb:push/>
      </rsb:call>
    </rsb:script>
    <rsb:script method="POST">
      <rsb:call op="elasticsearchadoModify">
        <rsb:push/>
      </rsb:call>
    </rsb:script>
    <rsb:script method="MERGE">
      <rsb:call op="elasticsearchadoModify">
        <rsb:push/>
      </rsb:call>
    </rsb:script>
    <rsb:script method="DELETE">
      <rsb:call op="elasticsearchadoModify">
        <rsb:push/>
      </rsb:call>
    </rsb:script>
  </rsb:script>
```

Advanced Features

This section details a selection of advanced features of the Elasticsearch adapter.

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 Elasticsearch 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.

User Defined Views

The Elasticsearch 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 [CData].[Elasticsearch].Employee 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.

Client SSL Certificates

The Elasticsearch adapter also supports setting client certificates. Set the following to connect using a client certificate.

- SSLClientCert: The name of the certificate store for the client certificate.
- SSLClientCertType: The type of key store containing the TLS/SSL client certificate.
- SSLClientCertPassword: The password for the TLS/SSL client certificate.
- SSLClientCertSubject: The subject of the TLS/SSL client certificate.

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 Elasticsearch 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 Elasticsearch Query Evaluation component examines SQL queries and returns information indicating what parts of the query the adapter is not capable of executing natively.

The Elasticsearch 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 Verbosity 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 Verbosity to 2 will log everything included in Verbosity 1 and additional information about the request.
3	Setting Verbosity to 3 will additionally log HTTP headers, as well as the body of the request and the response.
4	Setting Verbosity to 4 will additionally log transport-level communication with the data source. This includes SSL negotiation.
5	Setting Verbosity 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 verbosities, 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 Elasticsearch Adapter supports several operations on data, including querying, deleting, modifying, and inserting.

SELECT Statements

See SELECT Statements for a syntax reference and examples.

See Data Model for information on the capabilities of the Elasticsearch 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.

UPSERT Statements

An UPSERT updates a record if it exists and inserts the record if it does not. See UPSERT 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 Elasticsearch adapter:

```
SELECT {
  [ TOP <numeric_literal> | DISTINCT ]
  {
    *
    | {
        <expression> [ [ AS ] <column_reference> ]
        | { <table_name> | <correlation_name> } .*
      } [ , ... ]
  }
  [ INTO csv:// [ filename= ] <file_path> [ ;delimiter=tab ] ]
  {
    FROM <table_reference> [ [ AS ] <identifier> ]
  }
  [ WHERE <search_condition> ]
  [ GROUP BY <column_reference> [ , ... ]
  [
    ORDER BY
    <column_reference> [ ASC | DESC ] [ NULLS FIRST | NULLS LAST ]
  ]
  [
    LIMIT <expression>
    [
      { OFFSET | , }
      <expression>
    ]
  ]
} | 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> { = | > | < | >= | <= | <> | != | LIKE | NOT LIKE |
IS NULL | IS NOT NULL | IN | NOT IN | AND | OR | BETWEEN | CONTAINS |
NOT CONTAINS } [ <expression> ]
    } [ { AND | OR } ... ]
```

Examples

1. Return all columns:

   ```
   SELECT * FROM [CData].[Elasticsearch].Employee
   ```

2. Rename a column:

   ```
   SELECT "Name" AS MY_Name FROM [CData].[Elasticsearch].Employee
   ```

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

   ```
   SELECT CAST(AnnualRevenue AS VARCHAR) AS Str_AnnualRevenue FROM
   [CData].[Elasticsearch].Employee
   ```

4. Search data:

   ```
   SELECT * FROM [CData].[Elasticsearch].Employee WHERE Industry =
   'Floppy Disks'
   ```

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

   ```
   SELECT * FROM [CData].[Elasticsearch].Employee WHERE Industry =
   'Floppy Disks';
   ```

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

```
SELECT COUNT(*) AS MyCount FROM [CData].[Elasticsearch].Employee
```

7. Return the number of unique items matching the query criteria:

```
SELECT COUNT(DISTINCT Name) FROM [CData].[Elasticsearch].Employee
```

8. Return the unique items matching the query criteria:

```
SELECT DISTINCT Name FROM [CData].[Elasticsearch].Employee
```

9. Summarize data:

```
SELECT Name, MAX(AnnualRevenue) FROM [CData].
[Elasticsearch].Employee  GROUP BY Name
```

 See Aggregate Functions for details.

10. Sort a result set in ascending order:

```
SELECT Id, Name FROM [CData].[Elasticsearch].Employee  ORDER BY
Name ASC
```

Aggregate Functions

Examples of Aggregate Functions

Below are several examples of SQL aggregate functions. You can use these with a GROUP BY clause to aggregate rows based on the specified GROUP BY criterion. This can be a reporting tool.

COUNT

Returns the number of rows matching the query criteria.

```
SELECT COUNT(*) FROM [CData].[Elasticsearch].Employee WHERE Industry =
'Floppy Disks'
```

COUNT(DISTINCT)

Returns the number of distinct, non-null field values matching the query criteria.

```
SELECT COUNT(DISTINCT Id) AS DistinctValues FROM [CData].
[Elasticsearch].Employee WHERE Industry = 'Floppy Disks'
```

AVG

Returns the average of the column values.

```
SELECT Name, AVG(AnnualRevenue) FROM [CData].[Elasticsearch].Employee
WHERE Industry = 'Floppy Disks' GROUP BY Name
```

MIN

Returns the minimum column value.

```
SELECT MIN(AnnualRevenue), Name FROM [CData].[Elasticsearch].Employee
WHERE Industry = 'Floppy Disks' GROUP BY Name
```

MAX

Returns the maximum column value.

```
SELECT Name, MAX(AnnualRevenue) FROM [CData].[Elasticsearch].Employee
WHERE Industry = 'Floppy Disks' GROUP BY Name
```

SUM

Returns the total sum of the column values.

```
SELECT SUM(AnnualRevenue) FROM [CData].[Elasticsearch].Employee WHERE
Industry = 'Floppy Disks'
```

Predicate Functions

COMMON(expression, cutoff_frequency)

Used to explicitly specify the query type to send and thus will send 'expression' in a common terms query.

Example SQL Query:

```
SELECT * FROM employee WHERE COMMON(about) = 'like to build'
```

Elasticsearch Query:

```
{"common":{"about":{"query":"like to build"}}}
```

- **expression**: The expression to search for.
- **cutoff_frequency**: The cutoff frequency value used to allocate terms to the high or low frequency group. Can be an absolute frequency (>=1) or a relative frequency (0.0 .. 1.0).

FILTER(expression)

Used to explicitly specify the filter context and thus will send 'expression' in a filter context, rather than a query context. A filter context does not affect the calculated scores. This is useful when performing queries where you want part of the filter to be used to calculate scores but filter the results returned (without affecting the score) using additional criteria.

Example SQL Query:

```
SELECT * FROM employee WHERE FILTER(TERM(first_name)) = 'john'
```

Elasticsearch Query:

```
{"filter":{"bool":{"must":{"term":{"first_name":"john"}}}}}
```

- **expression**: Either a column or another function.

GEO_BOUNDING_BOX(column, top_left, bottom_right)

Used to specify a query to filter hits based on a point location using a bounding box.

Example SQL Query:

```
SELECT * FROM cities WHERE GEO_BOUNDING_BOX(location, '[-74.1,40.73]', '
[-71.12,40.01]')
```

Elasticsearch Query:

```
{"bool":{"filter":{"geo_bounding_box":{"location":{"top_left":[-
74.1,40.73],"bottom_right":[-71.12,40.01]}}},"must":[{"match_all":{}}]}}
```

- **column**: A Geo-point column to perform the GEO_BOUNDING_BOX filter on.
- **top_left**: The top-left coordinates of the bounding box. This value can be an array [shown in example], object of lat and lon values, comma-separated list, or a geohash of a latitude and longitude value.
- **bottom_right**: The bottom-right coordinates of the bounding box. This value can be an array [shown in example], object of lat and lon values, comma-separated list, or a geohash of a latitude and longitude value.

GEO_BOUNDING_BOX(column, top, left, bottom, right)

Used to specify a query to filter hits based on a point location using a bounding box.

Example SQL Query:

```
SELECT * FROM cities WHERE GEO_BOUNDING_BOX(location, -74.1, 40.73, -
71.12, 40.01)
```

Elasticsearch Query:

```
{"bool":{"filter":{"geo_bounding_box":{"location":{"top":-
74.1,"left":40.73,"bottom":-71.12,"right":40.01}}},"must":[{"match_all":
{}}]}}
```

- **column**: A Geo-point column to perform the GEO_BOUNDING_BOX filter on.
- **top**: The top coordinate of the bounding box.
- **left**: The left coordinate of the bounding box.
- **bottom**: The bottom coordinate of the bounding box.
- **right**: The right coordinate of the bounding box.

GEO_DISTANCE(column, point_lat_lon, distance)

Used to specify a query to filter documents that include only the hits that exist within a specific distance from a geo point.

Example SQL Query:

```
SELECT * FROM cities WHERE GEO_DISTANCE(location, '40,-70', '12mi')
```

Elasticsearch Query:

```
{"bool":{"filter":{"geo_distance":{"location":"40,-
70","distance":"12mi"}},"must":[{"match_all":{}}]}}
```

- **column**: A Geo-point column to perform the GEO_DISTANCE filter on.
- **point_lat_lon**: The coordinates of a geo point that will be used to measure the distance from. This value can be an array, object of lat and lon values, comma-separated list [shown in example], or a geohash of a latitude and longitude value.
- **distance**: The distance to search within from the specified geo point. This value takes an numeric value along with a distance unit. Common distance units are: mi (miles), yd (yards), ft (feet), in (inch), km (kilometers), m (meters). Please see Elastic documentation for complete list of distance units.

GEO_DISTANCE_RANGE(column, point_lat_lon, from_distance, to_distance)

Used to specify a query to filter documents that include only the hits that exist within a range from a specific geo point.

Example SQL Query:

```
SELECT * FROM cities WHERE GEO_DISTANCE_RANGE(location, 'drn5x1g8cu2y',
'10mi', '20mi')
```

Elasticsearch Query:

```
{"bool":{"filter":{"geo_distance_range":
{"location":"drn5x1g8cu2y","from":"10mi","to":"20mi"}},"must":[{"match_
all":{}}]}}
```

- **column**: A Geo-point column to perform the GEO_DISTANCE_RANGE filter on.
- **point_lat_lon**: The coordinates of a geo point that will be used to measure the range from. This value can be an array, object of lat and lon values, comma-separated list, or a geohash [shown in example] of a latitude and longitude value.
- **from_distance**: The starting distance to calculate the range from the specified geo point. This value takes an numeric value along with a distance unit. Common distance units are: mi (miles), yd (yards), ft (feet), in (inch), km (kilometers), m (meters). Please see Elastic documentation for complete list of distance units.
- **to_distance**: The end distance to calculate the range from the specified geo point. This value takes an numeric value along with a distance unit. Common distance units are: mi (miles), yd (yards), ft (feet), in (inch), km (kilometers), m (meters). Please see Elastic documentation for complete list of distance units.

GEO_POLYGON(column, points)

Used to specify a query to filter hits that only fall within a polygon of points.

Example SQL Query:

```
SELECT * FROM cities WHERE GEO_POLYGON(location, '[{"lat":40,"lon":-70},
{"lat":30,"lon":-80},{"lat":20,"lon":-90}]')
```

Elasticsearch Query:

```
{"bool":{"filter":{"geo_polygon":{"location":{"points":
[{"lat":40,"lon":-70},{"lat":30,"lon":-80},{"lat":20,"lon":-
90}]}}}},"must":[{"match_all":{}}]}}
```

- **column**: A Geo-point column to perform the GEO_POLYGON filter on.
- **points**: A JSON array of points that make up a polygon. This value can be an array of arrays, object of lat and lon values [shown in example], comma-separated lists, or geohashes of a latitude and longitude value.

GEO_SHAPE(column, type, points [, relation])

Used to specify an inline shape query to filter documents using the geo_shape type to find documents that have a shape that intersects with the query shape.

Example SQL Query:

```
SELECT * FROM shapes WHERE GEO_SHAPE(my_shape, 'envelope', '[[13.0,
53.0], [14.0, 52.0]]
```

Elasticsearch Query:

```
{"bool":{"filter":{"geo_shape":{"my_shape":{"shape":
{"type":"envelope","coordinates":[[13.0, 53.0], [14.0,
52.0]]}}}}},"must":[{"match_all":{}}]}}
```

- **column**: A Geo-shape column to perform the GEO_SHAPE filter on.
- **type**: The type of shape to search for. Valid values: point, linestring, polygon, multipoint, multilinestring, multipolygon, geometrycollection, envelope, and circle. Please see Elastic documentation for further information regarding these shapes.
- **points**: The coordinates for the shape type specified. These coordinates and their structure will vary depending upon the shape type desired. Please see Elastic search documentation for further details.
- **relation**: The name of the spatial relation operator to use at search time. Valid values: intersects (default), disjoint, within, and contains. Please see Elastic documentation for further information regarding spatial relations.

INARRAY(column)

Used to search for values contained within a primitive array. Supports comparison operators based on the data type contained within the array, including the LIKE operator.

Example SQL Query:

```
SELECT * FROM employee WHERE INARRAY(skills) = 'coding'
```

- **column**: A primitive array column to filter on.

MATCH(column)

Used to explicitly specify the query type to send and thus will send 'column' in a match query.

Example SQL Query:

```
SELECT * FROM employee WHERE MATCH(last_name) = 'SMITH'
```

Elasticsearch Query:

```
{"match":{"last_name":"SMITH"}}
```

- **column**: A column to perform the match query on.

MATCH_PHRASE(column)

Used to explicitly specify the query type to send and thus will send 'column' in a match phrase query.

Example SQL Query:

```
SELECT * FROM employee WHERE MATCH_PHRASE(about) = 'rides motorbikes'
```

Elasticsearch Query:

```
{"match_phrase":{"about":"rides motorbikes"}}
```

- **column**: A column to perform the match phrase query on.

MATCH_PHRASE_PREFIX(column)

Used to explicitly specify the query type to send and thus will send 'column' in a match phrase prefix query. The match phrase prefix query is the same as a match query except that it allows for prefix matches on the last term in the text.

Example SQL Query:

```
SELECT * FROM employee WHERE MATCH_PHRASE_PREFIX(about) = 'quick brown
f'
```

Elasticsearch Query:

```
{"match_phrase_prefix":{"about":"quick brown f"}}
```

- **expression**: A column to perform the match phrase prefix query on.

TERM(column)

Used to explicitly specify the query type to send and thus will send 'column' in a term query.

Example SQL Query:

```
SELECT * FROM employee WHERE TERM(last_name) = 'jacobs'
```

Elasticsearch Query:

```
{"term":{"last_name":"jacobs"}}
```

- **column**: A column to perform the term query on.

DSLQuery([table,] dsl_json)

Used to explicitly specify the Elasticsearch DSL query to send in the request. Can be used along with other filters and the AND and OR operators.

DSL query JSON can contain a full 'bool' query object, a 'must', 'should', 'must_not', or 'filter' occurrence type, or just a clause object (which will append to a 'must' (default) or 'should' occurrence type depending on whether an AND or OR operator is used).

Example SQL Query (These examples generate the same query using a 'bool' object, 'must' occurrence type, and query object):

```
SELECT * FROM employee WHERE DSLQuery('{"bool":{"must":[{"query_string":
{"default_field":"last_name","query":"\\"Smith\\""}}]}}')
SELECT * FROM employee WHERE DSLQuery('{"must":[{"query_string":
{"default_field":"last_name","query":"\\"Smith\\""}}]}')
SELECT * FROM employee WHERE DSLQuery('{"query_string":{"default_
field":"last_name","query":"\\"Smith\\""}}')
```

Elasticsearch Query:

```
{"bool":{"must":[{"query_string":{"default_field":"last_
name","query":"\\"Smith\\""}}]}}
```

Example SQL Query (with OR operator):

```
SELECT * FROM employee WHERE Age < 10 OR DSLQuery('{"should":[{"query_
string":{"default_field":"last_name","query":"\"Smith\""}}]}')
```

Elasticsearch Query:

```
{"bool":{"should":[{"range":{"age":{"lt":10}}},{"query_string":
{"default_field":"last_name","query":"\"Smith\""}}]}}
```

Additionally you can specify the table that you want the DSLQuery to be issued on, this is useful when executing queries against multiple tables such as JOIN queries.

Example SQL Query:

```
SELECT * FROM employee JOIN job ON employee.jobid = job.id WHERE
DSLQuery(employee, '{"bool":{"must":[{"query_string":{"default_
field":"last_name","query":"\\"Smith\\""}}]}}')
```

- **column**: A column to perform the term query on.

ORDER BY Functions

MAPFIELD(column, data_type)

Used to explicitly specify a mapping (by sending the 'unmapped_type' sort option) for a column that does not have a mapping associated with it, which will enable sorting on the column. By default, if a column does not have a mapping, an exception will be thrown containing an error message similar to: "No mapping found for [column] in order to sort on".

Example SQL Query:

```
SELECT * FROM employee ORDER BY MAPFIELD(start_date, 'long') DESC
```

Elasticsearch Sort:

```
{"start_date":{"order":"desc", "unmapped_type": "long"}}
```

- **column**: The column to perform the order by on.

- **data_type**: The Elasticsearch data type to map the column to.

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 [CData].[Elasticsearch].Employee (Name) VALUES
(?)";
PreparedStatement pstmt = connection.prepareStatement
(cmd,Statement.RETURN_GENERATED_KEYS);
pstmt.setString(1, "Floppy Disks");
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 [CData].[Elasticsearch].Employee SET Name='Floppy
Disks' WHERE Id = ?";
PreparedStatement pstmt = connection.prepareStatement(cmd);
pstmt.setString(1, "1");
int count = pstmt.executeUpdate();
System.out.println(count + " rows were affected");
connection.close();
```

UPSERT Statements

An UPSERT statement updates an existing record or creates a new record if an existing record is not identified.

UPSERT Syntax

The UPSERT syntax is the same as for INSERT. Elasticsearch uses the input provided in the VALUES clause to determine whether the record already exists. If the record does not exist, all columns required to insert the record must be specified. See Data Model for any table-specific information.

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

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

You can use the executeUpdate method of the Statement and PreparedStatement classes to issue data manipulation commands and retrieve the rows affected, as shown in the following example: 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 = "UPSERT INTO [CData].[Elasticsearch].Employee (Name) VALUES
(?)";
PreparedStatement pstmt = connection.prepareStatement
(cmd,Statement.RETURN_GENERATED_KEYS);
pstmt.setString(1, "Floppy Disks");
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();
```

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:elasticsearch:Server=127.0.0.1;Port=9200;",);
String cmd = "DELETE FROM [CData].[Elasticsearch].Employee 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 Synax

```
"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 Sytax

```
"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 Elasticsearch Adapter models Elasticsearch entities in relational Tables, Views, and Stored Procedures.

Tables

The table definitions are dynamically retrieved. When you connect, the adapter connects to Elasticsearch and retrieves the schemas, list of tables, and the metadata for the tables by querying the Elasticsearch REST server.

Searching with SQL describes in further detail how the tables are dynamically retrieved.

Views

Views are created from Elasticsearch aliases and the definitions are dynamically retrieved. When you connect, the adapter connects to Elasticsearch and retrieves the list of views and the metadata for the views by querying the Elasticsearch REST server.

Views are treated in a similar manner to Tables and thus exhibit similar behavior. There are some differences in the background though which are a direct result of how aliases work within Elasticsearch. (Note: In the following description, 'alias', 'index', 'type', and 'field' are referring to the Elasticsearch objects and not directly to anything within the adapter).

Views (aliases) are tied to an index and thus span all the types within an index. Additionally aliases can span multiple indices. Therefore you may see an alias (view) listed multiple times under different schemas (index). When querying the view, regardless of the schema specified, data will be retrieved and returned for all indices and types associated with the corresponding alias. Thus the generated metadata will contain a column for each field within each type of each index associated with the alias.

Searching with SQL describes in further detail how the views are dynamically retrieved.

The ModifyIndexAliases stored procedure can be used to create index aliases within Elasticsearch.

In addition to the Elasticsearch aliases, an '_all' view is returned which enables querying the _all endpoint to retrieve data for all indices in a single query. Given how many indices and documents the _all view could cover, certain queries agains the '_all' view could be very expensive. Additionally, for scanning for table metadata, as governed by RowScanDepth, will be less accurate for '_all' views that cover very large or very heterogenous indices. See The adapter automatically infers a relational schema by retrieving the mapping of the Elasticsearch type. The columns and data types are generated from the retrieved mapping.

Detecting Arrays

Any field within Elasticsearch can be an array of values, but this is not explicitly defined within the mapping. To account for this, the adapter will query the data to detect if any fields contain arrays. The number of Elasticsearch documents retrieved during this array scanning is based on the RowScanDepth property.

Elasticsearch nested types are special types that denote an array of objects and thus will always be treated as such when generating the metadata.

Detecting Columns

The columns identified during the discovery process depend on the FlattenArrays and FlattenObjects properties.

Example Data Set

To provide an example of how these options work, consider the following mapping (where 'insured' is the name of the table):

```
{
  "insured": {
    "properties": {
      "name": { "type":"string" },
      "address": {
        "street": { "type":"string" },
        "city": { "type":"string" },
        "state": { "type":"string" }
      },
      "insured_ages": { "type": "integer" },
      "vehicles": {
        "type": "nested",
        "properties": {
          "year": { "type":"integer" },
          "make": { "type":"string" },
          "model": { "type":"string" },
          "body_style" { "type": "string" }
        }
      }
    }
  }
}
```

Also consider the following example data for the above mapping:

```
  {
    "_source": {
      "name": "John Smith",
      "address": {
        "street": "Main Street",
        "city": "Chapel Hill",
        "state": "NC"
      },
      "insured_ages": [ 17, 43, 45 ],
      "vehicles": [
        {
          "year": 2015,
          "make": "Dodge",
          "model": "RAM 1500",
          "body_style": "TK"
        },
        {
          "year": 2015,
          "make": "Suzuki",
          "model": "V-Strom 650 XT",
          "body_style": "MC"
        },
        {
          "year": 2012,
          "make": "Honda",
          "model": "Accord",
          "body_style": "4D"
        }
    ]
  }
}
```

Using FlattenObjects

If FlattenObjects is set, all nested objects will be flattened into a series of columns. The above example will be represented by the following columns:

Column Name	Data Type	Example Value
name	String	John Smith
address.street	String	Main Street
address.city	String	Chapel Hill
address.state	String	NC
insured_ages	String	[17, 43, 45]
vehicles	String	[{ "year": "2015", "make": "Dodge", ... }, { "year": "2015", "make": "Suzuki", ... }, { "year": "2012", "make": "Honda", ... }]

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 the following:

```
{street: "Main Street", city: "Chapel Hill", state: "NC"}
```

See JSON Functions for more details on working with JSON aggregates.

Using FlattenArrays

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. It is best to leave unbounded arrays as they are and piece out the data for them as needed using JSON Functions.

Note: Only the top-most array will be flattened. Any subarrays will be represented as the entire array.

The FlattenArrays property can be set to 3 to represent the arrays in the example above as follows (this example is with FlattenObjects not set):

Column Name	Data Type	Example Value
insured_ages	String	[17, 43, 45]
insured_ages.0	Integer	17
insured_ages.1	Integer	43
insured_ages.2	Integer	45
vehicles	String	[{ "year": "2015", "make": "Dodge", ... }, { "year": "2015", "make": "Suzuki", ... }, { "year": "2012", "make": "Honda", ... }]
vehicles.0	String	{ "year": "2015", "make": "Dodge", "model": "RAM 1500", "body_style": "TK" }
vehicles.1	String	{ "year": "2015", "make": "Suzuki", "model": "V-Strom 650 XT", "body_style": "MC" }
vehicles.2	String	{ "year": "2012", "make": "Honda", "model": "Accord", "body_style": "4D" }

Using Both FlattenObjects and FlattenArrays

If FlattenObjects is set along with FlattenArrays (set to 1 for brevity), the vehicles field will be represented as follows:

Column Name	Data Type	Example Value
vehicles	String	[{ "year": "2015", "make": "Dodge", ... }, { "year": "2015", "make": "Suzuki", ... }, { "year": "2012", "make": "Honda", ... }]
vehicles.0.year	String	2015
vehicles.0.make	String	Dodge
vehicles.0.model	String	RAM 1500
vehicles.0.body_style	String	TK

for more information about this.

Stored Procedures

Stored Procedures are function-like interfaces to Elasticsearch which can be used to perform various tasks.

Stored Procedures

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

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

Elasticsearch Adapter Stored Procedures

Name	Description
CreateIndex	Submits a request to create an index with specified settings.

Name	Description
CreateSchema	Creates a schema file for the collection.
ModifyIndexAliases	Submits an alias request to modify index aliases.

CreateIndex

Submits a request to create an index with specified settings.

EXECUTE Example:

```
EXECUTE CreateIndex Index = 'firstindex', Alias = 'search',
NumberOfShards = '3'
```

Input

Name	Type	Description
Index	*String*	The name of the index.
Alias	*String*	The name of the alias to optionally associate the index with.
AliasFilter	*String*	Raw Query DSL object used to limit documents the alias can access.
AliasIndexRouting	*String*	Value used for the alias to route indexing operations to a specific shard. If specified, this overwrites the routing value for indexing operations.
AliasIsHidden	*String*	Boolean value controlling whether or not the alias is hidden. All indices for the alias must have the same is_hidden value.

Name	Type	Description
AliasIsWriteIndex	*String*	Boolean value controlling whether the index is the write index for the alias.
AliasRouting	*String*	Value used for the alias to route indexing and search operations to a specific shard. May be overwritten by AliasIndexRouting or AliasSearchRouting for certain operations.
AliasSearchRouting	*String*	Value used for the alias to route search operations to a specific shard. If specified, this overwrites the routing value for search operations.
Mappings	*String*	Raw JSON object specifying explicit mapping for the index.
NumberOfShards	*String*	The number of primary shards that the created index should have.
NumberOfRoutingShards	*String*	Number used by Elasticsearch internally with the value from NumberOfShards to route documents to a primary shard.
OtherSettings	*String*	Raw JSON object of settings. Cannot be used in conjunction with NumberOfRoutingShards or NumberOfShards.

Result Set Columns

Name	Type	Description
CompletedBeforeTimeout	*String*	Returns True if the index was created before timeout. Note that if this value is false, the index could still be created successfully on Elasticsearch. In this case, completion of creating the index, updating the cluster state, and requisite sharding

Name	Type	Description
		would occur after the timeout window for the request response elapsed.
ShardsAcknowledged	*String*	Boolean indicating whether the requisite number of shard copies were started for each shard in the index before timing out.
IndexName	*String*	Name in Elasticsearch of the created index.

CreateSchema

Creates a schema file for the collection.

Input

Name	Type	Accepts Output Streams	Description
SchemaName	*String*	*False*	For use with Elasticsearch versions earlier than 7.0. The name of the schema (the Elasticsearch index).
TableName	*String*	*False*	Pre Elasticsearch 7.0, the name of Elasticsearch mapping type. Post Elasticsearch 7.0, the name of the Elasticsearch index.
FileName	*String*	*False*	The file name sans extension of the generated schema.
FileStream	*String*	*True*	OutputStream to write the created schema.

Result Set Columns

Name	Type	Description
Result	*String*	Returns Success or Failure.
FileData	*String*	The generated schema encoded in base64. Only returned if none of FileName, FileLocation, or FileStream are set.

ModifyIndexAliases

Submits an alias request to modify index aliases.

EXECUTE Example:

```
EXECUTE ModifyIndexAliases Action = 'add;add', Index = 'index_1;index_
2', Alias = 'my_alias;my_alias'
```

Note: The Index parameter supports the asterisk (*) character to perform a pattern match to add all matching indices to the alias.

Note: This procedure makes use of **indexed parameters**. These input parameters are denoted with a '**#**' character at the end of their names.

Indexed parameters facilitate providing multiple instances a single parameter as inputs for the procedure.

Suppose there is an input parameter named Param#. Input multiple instances of an indexed parameter like this:

```
EXEC ProcedureName Param#1 = "value1", Param#2 = "value2", Param#3 =
"value3"
```

Input

Name	Type	Description
Action#	*String*	The action to perform such as 'add', 'remove', or 'remove_ index'. Multiple actions are semi-colon separated.
Index#	*String*	The name of the index. Multiple indexes are semi-colon separated.
Alias#	*String*	The name of the alias. Multiple aliases are semi-colon separated.
Filter#	*String*	A filter to use when creating the alias. This takes the raw JSON filter using Query DSL. Multiple filters are semi-colon separated.
Routing#	*String*	The routing value to associate with the alias. Multiple routing values are semi-colon separated.
SearchRouting#	*String*	The routing value to associate with the alias for searching operations. Multiple search routing values are semi-colon separated.
IndexRouting#	*String*	The routing value to associate with the alias for indexing operations. Multiple index routing values are semi-colon separated.

Result Set Columns

Name	Type	Description
Success	*String*	Returns True if successful.

Data Type Mapping

Data Type Mappings

The adapter maps types from the data source to the corresponding data type available in the schema. The table below documents these mappings.

Elasticsearch	CData Schema
array	A JSON structure*
binary	binary
boolean	boolean
byte	string
completion	string
date	datetime
date_range	datetime (one field per value)
double	double
double_range	double (one field per value)
float	float
float_range	float (one field per value)
geo_point	string
geo_shape	string
half_float	float
integer	integer
integer_range	integer (one field per value)

Elasticsearch	CData Schema
ip	string
keyword	string
long	long
long_range	long (one field per value)
nested	A JSON structure.*
object	Flattened into multiple fields.
scaled_float	float
short	short
text>	string

*Parsed into multiple fields with individual types (see FlattenArrays)

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.

Authentication

Property	Description
AuthScheme	The scheme used for authentication. Accepted entries are None, Basic, Negotiate (Kerberos), AwsRootKeys, AwsIAMRoles, and APIKey. None is the default.

Property	Description
User	The user who is authenticating to Elasticsearch.
Password	The password used to authenticate to Elasticsearch.
Server	The host name or IP address of the Elasticsearch REST server. Alternatively, multiple nodes in a single cluster can be specified, though all such nodes must be able to support REST API calls.
Port	The port for the Elasticsearch REST server.
APIKey	The APIKey used to authenticate to Elasticsearch.
APIKeyId	The APIKey Id to authenticate to Elasticsearch.

Connection

Property	Description
DataModel	Specifies the data model to use when parsing Elasticsearch documents and generating the database metadata.
UseLakeFormation	When this property is set to true, AWSLakeFormation service will be used to retrieve temporary credentials, which enforce access policies against the user based on the configured IAM role. The service can be used when authenticating through OKTA, ADFS, AzureAD, PingFederate, while providing a SAML assertion.

AWS Authentication

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

Property	Description
AWSSecretKey	Your AWS account secret key. This value is accessible from your AWS security credentials page.
AWSRoleARN	The Amazon Resource Name of the role to use when authenticating.
AWSRegion	The hosting region for your Amazon Web Services.
AWSSessionToken	Your AWS session token.
TemporaryTokenDuration	The amount of time (in seconds) an AWS temporary token will last.
AWSExternalId	A unique identifier that might be required when you assume a role in another account.

Kerberos

Property	Description
KerberosKDC	The Kerberos Key Distribution Center (KDC) service used to authenticate the user.
KerberosRealm	The Kerberos Realm used to authenticate the user.
KerberosSPN	The service principal name (SPN) for the Kerberos Domain Controller.
KerberosKeytabFile	The Keytab file containing your pairs of Kerberos principals and encrypted keys.
KerberosServiceRealm	The Kerberos realm of the service.
KerberosServiceKDC	The Kerberos KDC of the service.
KerberosTicketCache	The full file path to an MIT Kerberos credential cache file.

SSL

Property	Description
SSLClientCert	The TLS/SSL client certificate store for SSL Client Authentication (2-way SSL).
SSLClientCertType	The type of key store containing the TLS/SSL client certificate.
SSLClientCertPassword	The password for the TLS/SSL client certificate.
SSLClientCertSubject	The subject of the TLS/SSL client certificate.
SSLServerCert	The certificate to be accepted from the server when connecting using TLS/SSL.

Firewall

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

Proxy

Property	Description
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.
ProxyServer	The hostname or IP address of a proxy to route HTTP traffic through.
ProxyPort	The TCP port the ProxyServer proxy is running on.
ProxyAuthScheme	The authentication type to use to authenticate to the ProxyServer proxy.
ProxyUser	A user name to be used to authenticate to the ProxyServer proxy.
ProxyPassword	A password to be used to authenticate to the ProxyServer proxy.
ProxySSLType	The SSL type to use when connecting to the ProxyServer proxy.
ProxyExceptions	A semicolon separated list of destination hostnames or IPs that are exempt from connecting through the ProxyServer .

Logging

Property	Description
LogModules	Core modules to be included in the log file.

Schema

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

Property	Description
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.
FlattenArrays	Set FlattenArrays to the number of nested array elements you want to return as table columns. By default, nested arrays are returned as strings of JSON.

Miscellaneous

Property	Description
ClientSideEvaluation	Set ClientSideEvaluation to true to perform Evaluation client side on nested objects.
GenerateSchemaFiles	Indicates the user preference as to when schemas should be generated and saved.
MaxResults	The maximum number of total results to return from Elasticsearch when using the default Search API.
MaxRows	Limits the number of rows returned when no aggregation or GROUP BY is used in the query. This takes precedence over LIMIT clauses.
Other	These hidden properties are used only in specific use cases.
PageSize	The number of results to return per request from Elasticsearch.
QueryPassthrough	This option allows you to pass exact queries to Elasticsearch.
Readonly	You can use this property to enforce read-only access to Elasticsearch from the provider.

Property	Description
RowScanDepth	The maximum number of rows to scan when generating table metadata. Set this property to gain more control over how the provider detects arrays.
ScrollDuration	Specifies the time unit to use when retrieving results via the Scroll API.
Timeout	The value in seconds until the timeout error is thrown, canceling the operation.
UseFullyQualifiedNestedTableName	Set this to true to set the generated table name as the complete source path when flattening nested documents using Relational DataModel .
UserDefinedViews	A filepath pointing to the JSON configuration file containing your custom views.

Authentication

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

Property	Description
AuthScheme	The scheme used for authentication. Accepted entries are None, Basic, Negotiate (Kerberos), AwsRootKeys, AwsIAMRoles, and APIKey. None is the default.
User	The user who is authenticating to Elasticsearch.
Password	The password used to authenticate to Elasticsearch.
Server	The host name or IP address of the Elasticsearch REST server. Alternatively, multiple nodes in a single cluster can be specified, though all such nodes must be able to support REST API calls.

Property	Description
Port	The port for the Elasticsearch REST server.
APIKey	The APIKey used to authenticate to Elasticsearch.
APIKeyId	The APIKey Id to authenticate to Elasticsearch.

AuthScheme

The scheme used for authentication. Accepted entries are None, Basic, Negotiate (Kerberos), AwsRootKeys, AwsIAMRoles, and APIKey. None is the default.

Possible Values

AwsIAMRoles, None, APIKey, BASIC, TemporaryCredentials, Negotiate, AwsRootKeys

Data Type

string

Default Value

"None"

Remarks

This field is used to authenticate against the server. Use the following options to select your authentication scheme:

- None: No authentication is performed, unless User and Password properties are set in which BASIC authentication will be performed.
- Basic: Basic authentication is performed.
- Negotiate: If AuthScheme is set to Negotiate, the adapter will negotiate an authentication mechanism with the server. Set AuthScheme to Negotiate if you want to use Kerberos authentication.

- 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.
- APIKey: Set to use APIKey and APIKeyId for the connection.

User

The user who is authenticating to Elasticsearch.

Data Type

string

Default Value

""

Remarks

The user who is authenticating to Elasticsearch.

Password

The password used to authenticate to Elasticsearch.

Data Type

string

Default Value

""

Remarks

The password used to authenticate to Elasticsearch.

Server

The host name or IP address of the Elasticsearch REST server. Alternatively, multiple nodes in a single cluster can be specified, though all such nodes must be able to support REST API calls.

Data Type

string

Default Value

""

Remarks

The host name or IP address of the Elasticsearch REST server. Alternatively, multiple nodes in a single cluster can be specified, though all such nodes must be able to support REST API calls.

To use SSL, prefix the host name or IP address with 'https://' and set SSL connection properties such as SSLServerCert.

To specify multiple nodes, set the property to a comma delimited list of addresses, with ports optionally specified after the address and delimited from the address by a colon. For example, you could specify two dedicated, coordinating nodes for your cluster with 'https://01.01.01.01:1234,https://02.02.02.02:5678'. If a port is specified with a node, that port will take precedence over the Port property for connections to that node only.

Port

The port for the Elasticsearch REST server.

Data Type

string

Default Value

"9200"

Remarks

The port the Elasticsearch REST server is bound to.

APIKey

The APIKey used to authenticate to Elasticsearch.

Data Type

string

Default Value

""

Remarks

The APIKey used to authenticate to Elasticsearch.

APIKeyId

The APIKey Id to authenticate to Elasticsearch.

Data Type

string

Default Value

""

Remarks

The APIKey Id to authenticate to Elasticsearch.

Connection

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

Property	Description
DataModel	Specifies the data model to use when parsing Elasticsearch documents and generating the database metadata.
UseLakeFormation	When this property is set to true, AWSLakeFormation service will be used to retrieve temporary credentials, which enforce access policies against the user based on the configured IAM role. The service can be used when authenticating through OKTA, ADFS, AzureAD, PingFederate, while providing a SAML assertion.

DataModel

Specifies the data model to use when parsing Elasticsearch documents and generating the database metadata.

Possible Values

Document, FlattenedDocuments, Relational

Data Type

string

Default Value

"Document"

Remarks

Select a DataModel configuration to configure how the adapter models nested documents into tables. See Parsing Hierarchical Data for examples of querying the data in the different configurations.

Selecting a Data Modeling Strategy

The following DataModel configurations are available. See Parsing Hierarchical Data for examples of querying the data in the different configurations.

- **Document**

  Returns a single table representing a row for each document. In this data model, any nested documents will not be flattened and will be returned as aggregates.

- **FlattenedDocuments**

  Returns a single table representing a JOIN of the parent and nested documents. In this data model, nested documents will act in the same manner as a SQL JOIN. Additionally, nested sibling documents (nested documents at same height), will be treated as a SQL CROSS JOIN. The adapter will identify the nested documents available by parsing the returned document.

- **Relational**

  Returns multiple tables, one for each nested document (including the parent document) in the document. In this data model, any nested documents will be returned as relational tables that contain a primary key and a foreign key that links to the parent table.

See Also

- FlattenArrays and FlattenObjects: Customize the columns that will be identified for each of these data models. See Automatic Schema Discovery for examples of using these properties.

- Parsing Hierarchical Data: Compare the schemas resulting from different DataModel settings, with example queries.

- Searching with SQL: Learn about the data modeling and flattening techniques available in the adapter.

UseLakeFormation

When this property is set to true, AWSLakeFormation service will be used to retrieve temporary credentials, which enforce access policies against the user based on the configured IAM role. The service can be used when authenticating through OKTA, ADFS, AzureAD, PingFederate, while providing a SAML assertion.

Data Type

bool

Default Value

false

Remarks

When this property is set to true, AWSLakeFormation service will be used to retrieve temporary credentials, which enforce access policies against the user based on the configured IAM role. The service can be used when authenticating through OKTA, ADFS, AzureAD, PingFederate, while providing a SAML assertion.

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

Property	Description
AWSSecretKey	Your AWS account secret key. This value is accessible from your AWS security credentials page.
AWSRoleARN	The Amazon Resource Name of the role to use when authenticating.
AWSRegion	The hosting region for your Amazon Web Services.
AWSSessionToken	Your AWS session token.
TemporaryTokenDuration	The amount of time (in seconds) an AWS temporary token will last.
AWSExternalId	A unique identifier that might be required when you assume a role in another account.

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.

AWSSecretKey

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 Elasticsearch 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, GOVCLOUDWEST

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 GOVCLOUDWEST.

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.

TemporaryTokenDuration

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

Data Type

string

Default Value

"3600"

Remarks

Temporary tokens are used with Role based authentication. Temporary tokens will eventually time out, at which time a new temporary token must be obtained. The

Elasticsearch Adapter will internally request a new temporary token once the temporary token has expired.

For Role based authentication, the minimum duration is 900 seconds (15 minutes) while the maximum if 3600 (1 hour).

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.

Kerberos

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

Property	Description
KerberosKDC	The Kerberos Key Distribution Center (KDC) service used to authenticate the user.
KerberosRealm	The Kerberos Realm used to authenticate the user.
KerberosSPN	The service principal name (SPN) for the Kerberos Domain Controller.

Property	Description
KerberosKeytabFile	The Keytab file containing your pairs of Kerberos principals and encrypted keys.
KerberosServiceRealm	The Kerberos realm of the service.
KerberosServiceKDC	The Kerberos KDC of the service.
KerberosTicketCache	The full file path to an MIT Kerberos credential cache file.

KerberosKDC

The Kerberos Key Distribution Center (KDC) service used to authenticate the user.

Data Type

string

Default Value

""

Remarks

The Kerberos properties are used when using SPNEGO or Windows Authentication. The adapter will request session tickets and temporary session keys from the Kerberos KDC service. The Kerberos KDC service is conventionally colocated with the domain controller.

If Kerberos KDC is not specified, the adapter will attempt to detect these properties automatically from the following locations:

- **KRB5 Config File (krb5.ini/krb5.conf):** If the KRB5_CONFIG environment variable is set and the file exists, the adapter will obtain the KDC from the specified file. Otherwise, it will attempt to read from the default MIT location based on the OS: *C:\ProgramData\MIT\Kerberos5\krb5.ini* (Windows) or */etc/krb5.conf* (Linux).
- **Java System Properties:** Using the system properties *java.security.krb5.realm* and *java.security.krb5.kdc*.

- **Domain Name and Host:** If the Kerberos Realm and Kerberos KDC could not be inferred from another location, the adapter will infer them from the configured domain name and host.

Note: Windows authentication is supported in JRE 1.6 and above only.

KerberosRealm

The Kerberos Realm used to authenticate the user.

Data Type

string

Default Value

""

Remarks

The Kerberos properties are used when using SPNEGO or Windows Authentication. The Kerberos Realm is used to authenticate the user with the Kerberos Key Distribution Service (KDC). The Kerberos Realm can be configured by an administrator to be any string, but conventionally it is based on the domain name.

If Kerberos Realm is not specified, the adapter will attempt to detect these properties automatically from the following locations:

- **KRB5 Config File (krb5.ini/krb5.conf):** If the KRB5_CONFIG environment variable is set and the file exists, the adapter will obtain the default realm from the specified file. Otherwise, it will attempt to read from the default MIT location based on the OS: *C:\ProgramData\MIT\Kerberos5\krb5.ini* (Windows) or */etc/krb5.conf* (Linux)
- **Java System Properties:** Using the system properties *java.security.krb5.realm* and *java.security.krb5.kdc*.
- **Domain Name and Host:** If the Kerberos Realm and Kerberos KDC could not be inferred from another location, the adapter will infer them from the user-configured domain name and host. This might work in some Windows environments.

Note: Kerberos-based authentication is supported in JRE 1.6 and above only.

KerberosSPN

The service principal name (SPN) for the Kerberos Domain Controller.

Data Type

string

Default Value

""

Remarks

If the SPN on the Kerberos Domain Controller is not the same as the URL that you are authenticating to, use this property to set the SPN.

KerberosKeytabFile

The Keytab file containing your pairs of Kerberos principals and encrypted keys.

Data Type

string

Default Value

""

Remarks

The Keytab file containing your pairs of Kerberos principals and encrypted keys.

KerberosServiceRealm

The Kerberos realm of the service.

Data Type

string

Default Value

""

Remarks

The KerberosServiceRealm is the specify the service Kerberos realm when using cross-realm Kerberos authentication.

In most cases, a single realm and KDC machine are used to perform the Kerberos authentication and this property is not required.

This property is available for complex setups where a different realm and KDC machine are used to obtain an authentication ticket (AS request) and a service ticket (TGS request).

KerberosServiceKDC

The Kerberos KDC of the service.

Data Type

string

Default Value

""

Remarks

The KerberosServiceKDC is used to specify the service Kerberos KDC when using cross-realm Kerberos authentication.

In most cases, a single realm and KDC machine are used to perform the Kerberos authentication and this property is not required.

This property is available for complex setups where a different realm and KDC machine are used to obtain an authentication ticket (AS request) and a service ticket (TGS request).

KerberosTicketCache

The full file path to an MIT Kerberos credential cache file.

Data Type

string

Default Value

""

Remarks

This property can be set if you wish to use a credential cache file that was created using the MIT Kerberos Ticket Manager or kinit command.

SSL

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

Property	Description
SSLClientCert	The TLS/SSL client certificate store for SSL Client Authentication (2-way SSL).
SSLClientCertType	The type of key store containing the TLS/SSL client certificate.
SSLClientCertPassword	The password for the TLS/SSL client certificate.
SSLClientCertSubject	The subject of the TLS/SSL client certificate.

Property	Description
SSLServerCert	The certificate to be accepted from the server when connecting using TLS/SSL.

SSLClientCert

The TLS/SSL client certificate store for SSL Client Authentication (2-way SSL).

Data Type

string

Default Value

""

Remarks

The name of the certificate store for the client certificate.

The SSLClientCertType field specifies the type of the certificate store specified by SSLClientCert. If the store is password protected, specify the password in SSLClientCertPassword.

SSLClientCert is used in conjunction with the SSLClientCertSubject field in order to specify client certificates. If SSLClientCert has a value, and SSLClientCertSubject is set, a search for a certificate is initiated. See SSLClientCertSubject for more information.

Designations of certificate stores are platform-dependent.

The following are designations of the most common User and Machine certificate stores in Windows:

MY	A certificate store holding personal certificates with their associated private keys.
CA	Certifying authority certificates.
ROOT	Root certificates.
SPC	Software publisher certificates.

In Java, the certificate store normally is a file containing certificates and optional private keys.

When the certificate store type is PFXFile, this property must be set to the name of the file. When the type is PFXBlob, the property must be set to the binary contents of a PFX file (for example, PKCS12 certificate store).

SSLClientCertType

The type of key store containing the TLS/SSL client certificate.

Possible Values

USER, MACHINE, PFXFILE, PFXBLOB, JKSFILE, JKSBLOB, PEMKEY_FILE, PEMKEY_BLOB, PUBLIC_KEY_FILE, PUBLIC_KEY_BLOB, SSHPUBLIC_KEY_FILE, SSHPUBLIC_KEY_BLOB, P7BFILE, PPKFILE, XMLFILE, XMLBLOB

Data Type

string

Default Value

"USER"

Remarks

This property can take one of the following values:

USER - default	For Windows, this specifies that the certificate store is a certificate store owned by the current user. Note that this store type is not available in Java.
MACHINE	For Windows, this specifies that the certificate store is a machine store. Note that this store type is not available in Java.
PFXFILE	The certificate store is the name of a PFX (PKCS12) file containing certificates.
PFXBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in PFX (PKCS12) format.
JKSFILE	The certificate store is the name of a Java key store (JKS) file containing certificates. Note that this store type is only available in Java.
JKSBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in JKS format. Note that this store type is only available in Java.
PEMKEY_FILE	The certificate store is the name of a PEM-encoded file that contains a private key and an optional certificate.
PEMKEY_BLOB	The certificate store is a string (base64-encoded) that contains a private key and an optional certificate.
PUBLIC_KEY_FILE	The certificate store is the name of a file that contains a PEM- or DER-encoded public key certificate.
PUBLIC_KEY_BLOB	The certificate store is a string (base-64-encoded) that contains a PEM- or DER-encoded public key certificate.
SSHPUBLIC_KEY_FILE	The certificate store is the name of a file that contains an SSH-style public key.
SSHPUBLIC_KEY_BLOB	The certificate store is a string (base-64-encoded) that contains an SSH-style public key.
P7BFILE	The certificate store is the name of a PKCS7 file containing

USER - default	**For Windows, this specifies that the certificate store is a certificate store owned by the current user. Note that this store type is not available in Java.**
	certificates.
PPKFILE	The certificate store is the name of a file that contains a PuTTY Private Key (PPK).
XMLFILE	The certificate store is the name of a file that contains a certificate in XML format.
XMLBLOB	The certificate store is a string that contains a certificate in XML format.

SSLClientCertPassword

The password for the TLS/SSL client certificate.

Data Type

string

Default Value

""

Remarks

If the certificate store is of a type that requires a password, this property is used to specify that password to open the certificate store.

SSLClientCertSubject

The subject of the TLS/SSL client certificate.

Data Type

string

Default Value

"*"

Remarks

When loading a certificate the subject is used to locate the certificate in the store.

If an exact match is not found, the store is searched for subjects containing the value of the property. If a match is still not found, the property is set to an empty string, and no certificate is selected.

The special value "*" picks the first certificate in the certificate store.

The certificate subject is a comma separated list of distinguished name fields and values. For example, "CN=www.server.com, OU=test, C=US, E=support@company.com". The common fields and their meanings are shown below.

Field	Meaning
CN	Common Name. This is commonly a host name like www.server.com.
O	Organization
OU	Organizational Unit
L	Locality
S	State
C	Country
E	Email Address

If a field value contains a comma, it must be quoted.

SSLServerCert

The certificate to be accepted from the server when connecting using TLS/SSL.

Data Type

string

Default Value

""

Remarks

If using a TLS/SSL connection, this property can be used to specify the TLS/SSL certificate to be accepted from the server. Any other certificate that is not trusted by the machine is rejected.

This property can take the following forms:

Description	Example
A full PEM Certificate (example shortened for brevity)	-----BEGIN CERTIFICATE----- MIIChTCCAe4CAQAwDQYJKoZIhv......Qw== -----END CERTIFICATE-----
A path to a local file containing the certificate	C:\cert.cer
The public key (example shortened for brevity)	-----BEGIN RSA PUBLIC KEY----- MIGfMA0GCSq......AQAB -----END RSA PUBLIC KEY-----
The MD5 Thumbprint (hex values can also be either space or colon separated)	ecadbdda5a1529c58a1e9e09828d70e4
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
FirewallType	The protocol used by a proxy-based firewall.
FirewallServer	The name or IP address of a proxy-based firewall.
FirewallPort	The TCP port for a proxy-based firewall.
FirewallUser	The user name to use to authenticate with a proxy-based firewall.
FirewallPassword	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 Elasticsearch 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 FirewallServer and FirewallPort and passes the FirewallUser 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 FirewallServer and FirewallPort. If your proxy requires authentication, set FirewallUser and FirewallPassword 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
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.
ProxyServer	The hostname or IP address of a proxy to route HTTP traffic through.
ProxyPort	The TCP port the ProxyServer proxy is running on.
ProxyAuthScheme	The authentication type to use to authenticate to the ProxyServer proxy.
ProxyUser	A user name to be used to authenticate to the ProxyServer proxy.
ProxyPassword	A password to be used to authenticate to the ProxyServer proxy.
ProxySSLType	The SSL type to use when connecting to the ProxyServer proxy.
ProxyExceptions	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:

AUTO	**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.**
ALWAYS	The connection is always SSL enabled.
NEVER	The connection is not SSL enabled.
TUNNEL	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
LogModules	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
Location	A path to the directory that contains the schema files defining tables, views, and stored procedures.
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.
FlattenArrays	Set FlattenArrays to the number of nested array elements you want to return as table columns. By default, nested arrays are returned as strings of JSON.

Location

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

Data Type

string

Default Value

"%APPDATA%\\CData\\Elasticsearch 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\\Elasticsearch 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

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 a period to generate the column name.

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

```
"manager": {
  "name": "Alice White",
  "age": 30
}
```

When FlattenObjects is set to true, the preceding object is flattened into the following table:

Column Name	Column Value
manager.name	Alice White
manager.age	30

FlattenArrays

Set FlattenArrays to the number of nested array elements you want to return as table columns. By default, nested arrays are returned as strings of JSON.

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:

```
"employees": [
  {
    "name": "John Smith",
    "age": 34
  },
  {
    "name": "Peter Brown",
    "age": 26
  },
  {
    "name": "Paul Jacobs",
    "age": 30
  }
]
```

When FlattenArrays is set to 2, the preceding array is flattened into the following table:

Column Name	Column Value
employees.0.name	John Smith
employees.0.age	34
employees.1.name	Peter Brown
employees.1.age	26

See JSON Functions to use JSON paths to work with unbounded arrays.

Miscellaneous

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

Property	Description
ClientSideEvaluation	Set ClientSideEvaluation to true to perform

Property	Description
	Evaluation client side on nested objects.
GenerateSchemaFiles	Indicates the user preference as to when schemas should be generated and saved.
MaxResults	The maximum number of total results to return from Elasticsearch when using the default Search API.
MaxRows	Limits the number of rows returned when no aggregation or GROUP BY is used in the query. This takes precedence over LIMIT clauses.
Other	These hidden properties are used only in specific use cases.
PageSize	The number of results to return per request from Elasticsearch.
QueryPassthrough	This option allows you to pass exact queries to Elasticsearch.
Readonly	You can use this property to enforce read-only access to Elasticsearch from the provider.
RowScanDepth	The maximum number of rows to scan when generating table metadata. Set this property to gain more control over how the provider detects arrays.
ScrollDuration	Specifies the time unit to use when retrieving results via the Scroll API.
Timeout	The value in seconds until the timeout error is thrown, canceling the operation.
UseFullyQualifiedNestedTableName	Set this to true to set the generated table name as the complete source path when flattening nested documents using Relational DataModel .

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

ClientSideEvaluation

Set ClientSideEvaluation to true to perform Evaluation client side on nested objects.

Data Type

bool

Default Value

false

Remarks

Set ClientSideEvaluation to true to perform Evaluation (GROUP BY, filtering) client side on nested objects.

For example, with ClientSideEvaluation set to false(default value), GROUP BY on nested object 'property.0.name' would be grouped as 'property.*.name', while if set to true, results would be grouped as 'property.0.name'.

Similarly, with ClientSideEvaluation set to false(default value), filtering on nested object 'property.0.name' would be filtered as 'property.*.name', while if set to true, results would be filtered as 'property.0.name'.

This would affect performance as query is evaluated client side.

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.

MaxResults

The maximum number of total results to return from Elasticsearch when using the default Search API.

Data Type

string

Default Value

"10000"

Remarks

This property corresponds to the Elasticsearch *index.max_result_window* index setting. Thus the default value is 10000, which is Elasticsearch's default limit.

This value is not applicable when using the Scroll API. Set ScrollDuration to use this API.

When a LIMIT is specified in a query, the LIMIT will be taken into account provided it is less than MaxResults. Otherwise the number of results returned will be limited to the MaxResults value.

If you receive an error stating that the result window is too large, this is caused by the MaxResults value being greater than the Elasticsearch *index.max_result_window* index setting. You can either change the MaxResults value to match the *index.max_result_window* index setting or use the Scroll API by setting ScrollDuration.

MaxRows

Limits the number of rows returned when no aggregation or GROUP BY is used in the query. This takes precedence over LIMIT clauses.

Data Type

int

Default Value

-1

Remarks

Limits the number of rows returned when no aggregation or GROUP BY is used in the query. This takes precedence over LIMIT clauses.

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 number of results to return per request from Elasticsearch.

Data Type

string

Default Value

"10000"

Remarks

The PageSize can control the number of results received per request from Elasticsearch on a given query.

The default value is 10000, which is Elasticsearch's default limit (based on the Elasticsearch *index.max_result_window* index setting).

QueryPassthrough

This option allows you to pass exact queries to Elasticsearch.

Data Type

bool

Default Value

false

Remarks

Setting this property to True enables the adapter to pass an Elasticsearch query as-is to Elasticsearch. There are two options for submitting as-is queries to Elasticsearch: SQL and Search DSL.

SQL API

Elasticsearch version 6.3 and above supports a SQL API endpoint. When set to true, this option allows you to pass SQL queries directly to the Elasticsearch SQL API. Columns will be identified based on the metadata returned in the response.

Supported SQL syntax and commands can be found in the Elasticsearch documentation.

Note: SQL functionality is limited to what is supported by Elasticsearch.

Search DSL

Alternatively, queries can be submitted using Elasticsearch's Search DSL language, which includes Query DSL. This functionality is available in all versions of Elasticsearch.

The supported query syntax is JSON using the query passthrough syntax described below.

The JSON Passthrough Query Syntax supports the following elements:

Element Name	Function
index	The Elasticsearch index (or schema) to query. This is a JSON element that takes a string value.
type	The Elasticsearch type (or table) to query within *index*. This is a JSON element that takes a string value.
docid	The Id of the document to query within *index.type*. This is a JSON element that takes a string value.
apiendpoint	The Elasticsearch API Endpoint to query. Default value is '_search'. This is a JSON element that takes a string value.
requestdata	The raw Elasticsearch Search DSL that will be sent to Elasticsearch as is. The value is a JSON object that maps directly to the format required by Elasticsearch.

The *index*, *type*, *docid*, and *apiendpoint* are used to generate the URL where the *requestdata* will be sent. The URL is generated using the following format: [Server]:[Port]/[index]/[type]/[docid]/[apiendpoint]. If any of the JSON passthrough elements are not specified, they will not be added to the URL.

Below is an example of a passthrough query. This example will retrieve the first 10 documents from *megacorp.employee* that contain a *last_name* of 'smith'. The results will be ordered by *first_name* in descending order.

```
{
  "index": "megacorp",
  "type": "employee",
  "requestdata":
  {
    "from": 0,
    "size": 10,
    "query": {"bool":{"must":{"term":{"last_name":"smith"}}}},
    "sort": {"first_name":{"order":"desc"}}
  }
}
```

When using QueryPassthrough queries, the metadata is determined by the data returned in the response. RowScanDepth identifies the depth of the records that will be scanned to determine the metadata (columns and types). Since the metadata is based on the response data, passthrough queries may display different metadata than a similar query performed using the SQL syntax (where the metadata is retrieved directly from Elasticsearch).

Readonly

You can use this property to enforce read-only access to Elasticsearch 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.

RowScanDepth

The maximum number of rows to scan when generating table metadata. Set this property to gain more control over how the provider detects arrays.

Data Type

string

Default Value

"100"

Remarks

This property is used when generating table metadata and specifically is used to identify arrays within the data. Elasticsearch allows any field to be an array and does not identify which fields are arrays in the mapping data. Thus RowScanDepth rows will be queried and scanned to identify if any of the fields contain arrays.

When QueryPassthrough is set to True, the columns in a table must be determined by scanning the data returned in the request. This value determines the maximum number of rows that will be scanned to determine the table metadata. The default value is 100.

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 or when the scanned documents are very heterogenous.

ScrollDuration

Specifies the time unit to use when retrieving results via the Scroll API.

Data Type

string

Default Value

"1m"

Remarks

When a nonzero value is specified, the Scroll API will be used.

The time unit specified will be sent in each request made to Elasticsearch to specify how long the server should keep the search context alive. The value specified only needs to be long enough to process the previous batch of results (not to process all the data). This is because the ScrollDuration value will be sent in each request, which will extend the context time.

Once all the results have been retrieved, the search context will be cleared.

The format for this value is: [integer][time unit]. For example: 1m = 1 minute.

Setting this property to '0' will cause the default Search API to be used. In such a case, the maximum number of results that can be returned are equal to MaxResults.

Supported Time Units:

Value	Description
y	Year
M	Month
w	Week
d	Day
h	Hour
m	Minute
s	Second
ms	Milli-second

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.

UseFullyQualifiedNestedTableName

Set this to true to set the generated table name as the complete source path when flattening nested documents using Relational DataModel .

Data Type

bool

Default Value

false

Remarks

Set this to true to set the generated table name as the complete source path when flattening nested documents using Relational DataModel.

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 [CData].[Elasticsearch].Employee 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"
```

TIBCO Documentation and Support Services

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

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the Product Documentation website, mainly in HTML and PDF formats.

The 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

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.

Release Version Support

TDV 8.5 and 8.8 are designated as Long Term Support (LTS) versions. 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 Long Term Support.

How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our product Support website.
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the our product Support website. If you do not have a username, you can request one by clicking **Register** on the website.

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, go to TIBCO Community.

Legal and Third-Party Notices