TIBCO Data Virtualization®

Elasticsearch Adapter Guide

Version 8.4

***Last Updated:** January 25, 2021*

Two-Second Advantage®

TIBCO®

Important Information

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

PURPOSE.

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

TIBCO and the TIBCO logo are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries

TIBCO, Two-Second Advantage, TIBCO Spotfire, TIBCO ActiveSpaces, TIBCO Spotfire Developer, TIBCO EMS, TIBCO Spotfire Automation Services, TIBCO Enterprise Runtime for R, TIBCO Spotfire Server, TIBCO Spotfire Web Player, TIBCO Spotfire Statistics Services, S-PLUS, and TIBCO Spotfire S+ are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

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

THIS SOFTWARE MAY BE AVAILABLE ON MULTIPLE OPERATING SYSTEMS. HOWEVER, NOT ALL OPERATING SYSTEM PLATFORMS FOR A SPECIFIC SOFTWARE VERSION ARE RELEASED AT THE SAME TIME. SEE THE README FILE FOR THE AVAILABILITY OF THIS SOFTWARE VERSION ON A SPECIFIC OPERATING SYSTEM PLATFORM.

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

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

Contents

Preface

Forinformationonthefollowing,seethe *TDVUserGuide*:

- AddingaDataSource
- IntrospectingaDataSource
- TestingtheConnectiontoYourDataSource

Documentation for this and other TIBCO products is available on the TIBCO Documentation site. This site is updated more frequently than any documentation that might be included with the product. To ensure that you are accessing the latest available help topics, please visit:

- https://docs.tibco.com

Product-Specific Documentation

The following documents form the TIBCO® Data Virtualization(TDV) documentation set:

- Users
  - TDV Getting Started Guide
  - TDV User Guide
  - TDV Client Interfaces Guide
  - TDV Tutorial Guide
  - TDV Northbay Example
- Administration
  - TDV Installation and Upgrade Guide
  - TDV Administration Guide
  - TDV Active Cluster Guide
  - TDV Security Features Guide
- Data Sources
  - TDV Adapter Guides
  - TDV Data Source Toolkit Guide (Formerly Extensibility Guide)

- **References**

  TDV Reference Guide

  TDV Application Programming Interface Guide

- **Other**

  TDV Business Directory Guide

  TDV Discovery Guide

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

How to Access TIBCO Documentation

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

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product. To access the latest documentation, visit https://docs.tibco.com.

Documentation for TIBCO Data Virtualization is available on https://docs.tibco.com/products/tibco-data-virtualization-server.

How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

- For an overview of TIBCO Support, visit https://www.tibco.com/services/support.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at https://support.tibco.com.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to https://support.tibco.com. If you do not have a user name, 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 https://community.tibco.com.

TDV Elasticsearch Adapter

Requirements and Restrictions

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

Deploying the Adapter

For instructions on deploying the adapter, refer to the *Installation Guide, section Installing the Advanced Adapters*.

Basic Tab

Obtaining the Access Key

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 select the Security Credentials tab.

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

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

Authenticating Using Standard Authentication

Set the 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://'.

Authenticating with 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.

Authenticating with 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.

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

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

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

Authenticating as an AWS Role

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

To authenticate as an AWS role, set the following:

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

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

- AWSAccessKey: The access key of the IAM user to assume the role for.
- AWSSecretKey: The secret key of the IAM user to assume the role for.

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

Connecting to Elasticsearch Service

Once you have configured the adapter for your desired authentication method, 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.

Advanced Tab

The connection string properties describe the various options that can be used to establish a connection.

Connection String Options

The following is the full list of the options you can configure in the connection string for this provider.

Auth Scheme	The scheme used for authentication. Accepted entries are NONE, BASIC, and NEGOTIATE (Kerberos). NONE is the default.
Auth Scheme	Your AWS account access key. This value is accessible from your AWS security credentials page.
AWS Region	The hosting region for your Amazon Web Services.
AWS Role ARN	The Amazon Resource Name of the role to use when authenticating.
AWS Secret Key	Your AWS account secret key. This value is accessible from your AWS security credentials page.
AWS Session Token	Your AWS session token.
Client Side Evaluation	Set ClientSideEvaluation to true to perform Evaluation client side on nested objects.
Data Model	Specifies the data model to use when parsing Elasticsearch documents and generating the database metadata.
Firewall Password	A password used to authenticate to a proxy-based firewall.
Firewall Port	The TCP port for a proxy-based firewall.
Firewall Server	The name or IP address of a proxy-based firewall.
Firewall Type	The protocol used by a proxy-based firewall.
Firewall User	The user name to use to authenticate with a proxy-based firewall.
Flatten Arrays	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.
Flatten Objects	Set FlattenObjects to true to flatten object properties into columns of their own. Otherwise, objects nested in arrays are returned as strings of JSON.

Generate Schema Files	Indicates the user preference as to when schemas should be generated and saved.
Kerberos KDC	The Kerberos Key Distribution Center (KDC) service used to authenticate the user.
Kerberos Keytab File	The Keytab file containing your pairs of Kerberos principals and encrypted keys.
Kerberos Realm	The Kerberos Realm used to authenticate the user with.
Kerberos Service KDC	The Kerberos KDC of the service.
Kerberos Service Realm	The Kerberos realm of the service.
Kerberos SPN	The service principal name (SPN) for the Kerberos Domain Controller.
Kerberos Ticket Cache	The full file path to an MIT Kerberos credential cache file.
Location	A path to the directory that contains the schema files defining tables, views, and stored procedures.
Log Modules	Core modules to be included in the log file.
Max Results	The maximum number of total results to return from Elasticsearch when using the default Search API.
Max Rows	Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.
Other	The other parameters necessary to connect to a data source, such as username and password, when applicable.
Page Size	The number of results to return per request from Elasticsearch.
Password	The password used to authenticate to Elasticsearch.
Port	The port for the Elasticsearch REST server.
Proxy Auth Scheme	The authentication type to use to authenticate to the ProxyServer proxy.

Proxy Auto Detect	This indicates whether to use the system proxy settings or not. Set ProxyAutoDetect to FALSE to use custom proxy settings. This takes precedence over other proxy settings.
Proxy Exceptions	A semicolon separated list of hosts or IPs that will be exempt from connecting through the ProxyServer .
Proxy Password	A password to be used to authenticate to the ProxyServer proxy.
Proxy Port	The TCP port the ProxyServer proxy is running on.
Proxy Server	The hostname or IP address of a proxy to route HTTP traffic through.
Proxy SSL Type	The SSL type to use when connecting to the ProxyServer proxy.
Proxy User	A user name to be used to authenticate to the ProxyServer proxy.
Query Passthrough	This option allows you to pass queries to Elasticsearch using Elasticsearch's Search DSL language, which includes Query DSL.
Readonly	You can use this property to enforce read-only access to Elasticsearch from the provider.
Row Scan Depth	The maximum number of rows to scan when generating table metadata. Set this property to gain more control over how the provider detects arrays.
Scroll Duration	Specifies the time unit to use when retrieving results via the Scroll API.
Server	The host name or IP address of the Elasticsearch REST server.
SSL Client Cert	The TLS/SSL client certificate store for SSL Client Authentication (2-way SSL).
SSL Client Cert Password	The password for the TLS/SSL client certificate.
SSL Client Cert Subject	The subject of the TLS/SSL client certificate.
SSL Client Cert Type	The type of key store containing the TLS/SSL client certificate.
SSL Server Cert	The certificate to be accepted from the server when connecting using TLS/SSL.
Temporary Token Duration	The amount of time (in seconds) an AWS temporary token will last.

Timeout	The value in seconds until the timeout error is thrown, canceling the operation.
User	The user who is authenticating to Elasticsearch.

Auth Scheme

The scheme used for authentication. Accepted entries are NONE, BASIC, and NEGOTIATE (Kerberos). NONE is the default.

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.

Access Key

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.

AWS Region

The hosting region for your Amazon Web Services.

Data Type

string

Default Value

"NORTHERNVIRGINIA"

Remarks

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

AWS Role ARN

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.

AWS Secret Key

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.

AWS Session Token

Your AWS session token.

Data Type

string

Default Value

""

Remarks

Your AWS session token. This value can be retrieved in different ways.

Client Side Evaluation

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.

Data Model

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

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

- Flatten Arrays and Flatten<XREF>Objects: 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.

Firewall Password

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 Firewall Server and Firewall Port, following the authentication method specified by Firewall Type.

Firewall Port

The TCP port for a proxy-based firewall.

Data Type

string

Default Value

""

Remarks

This specifies the TCP port for a proxy allowing traversal of a firewall. Use Firewall Server to specify the name or IP address. Specify the protocol with Firewall Type.

Firewall Server

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 Firewall Type: Use Firewall Server with this property to connect through SOCKS or do tunneling. Use Proxy Server to connect to an HTTP proxy.

Note that the adapter uses the system proxy by default. To use a different proxy, set Proxy Auto Detect to false.

Firewall Type

The protocol used by a proxy-based firewall.

Data Type

string

Default Value

"NONE"

Remarks

This property specifies the protocol that the adapter will use to tunnel traffic through the Firewall Server 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 Proxy Auto Detect 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 Firewall Server and Firewall Port and passes the Firewall User 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 Firewall Server and Firewall Port. If your proxy requires authentication, set Firewall User and Firewall Password to credentials the proxy recognizes.

To connect to HTTP proxies, use Proxy Server and Proxy Port. To authenticate to HTTP proxies, use Proxy Auth Scheme, Proxy User, and Proxy Password.

Firewall User

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

Data Type

string

Default Value

""

Remarks

The Firewall User and Firewall Password properties are used to authenticate against the proxy specified in Firewall Server and Firewall Port, following the authentication method specified in Firewall Type.

Flatten Arrays

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.

Flatten Objects

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

Generate Schema Files

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

Data Type

string

Default Value

"Never"

Remarks

GenerateSchemaFiles enables you to persist a relational view of Elasticsearch types (tables) or queries; this property outputs schemas to .rsd files in the path specified by Location.

See the following sections for ways to generate schema files or for alternatives. Schema files are easy to edit, but they are static: If you want to regenerate a file, you will first need to delete it.

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.

Generate Schemas with SQL

By setting GenerateSchemaFiles to OnUse, you can create views on queries you execute. For example, you could specify the XPath to a column value in your SELECT query with an SQL function -- see JSON Functions for examples. See Query Mapping for more information on how to write SQL queries to Elasticsearch.

If you want to write a new query for the view, you will need to first delete the old schema from the Location folder.

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.

See Automatic Schema Discovery to use connection properties to fine-tune the tables rendered.

Editing Schemas

Schema files have a simple format that makes them easy to modify. See Custom Schema Definitions for an end-to-end guide.

Alternatives to Static Schemas

If your data structures are volatile, consider setting GenerateSchemaFiles to Never and using dynamic schemas. See Automatic Schema Discovery for more information about dynamic schemas.

Kerberos KDC

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 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:

Java System Properties: Kerberos settings can be configured in Java using the configuration file krb5.conf, or using the system properties java.security.krb5.realm and java.security.krb5.kdc. The adapter will use the system settings if Kerberos Realm and Kerberos KDC are not explicitly set.

Domain Name and Host: The adapter will infer the Kerberos Realm and Kerberos KDC from the configured domain name and host as a last resort.

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

Kerberos Keytab File

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.

Kerberos Realm

The Kerberos Realm used to authenticate the user with.

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:

Java System Properties: Kerberos settings can be configured in Java using a configuration file (krb5.conf) or using the system properties java.security.krb5.realm and java.security.krb5.kdc. The adapter will use the system settings if Kerberos Realm and Kerberos KDC are not explicitly set.

Domain Name and Host: The adapter will infer the Kerberos Realm and Kerberos KDC from the user-configured domain name and host as a last resort. This might work in some Windows environments.

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

Kerberos Service KDC

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).

Kerberos Service Realm

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).

Kerberos SPN

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.

Kerberos Ticket Cache

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.

Location

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

Data Type

string

Default Value

""

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

Log Modules

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.

Max Results

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 Scroll Duration 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 Scroll Duration.

Max Rows

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

Data Type

string

Default Value

"-1"

Remarks

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

Other

The other parameters necessary to connect to a data source, such as username and password, when applicable.

Data Type

string

Default Value

""

Remarks

The Other property is a semicolon-separated list of name-value pairs used in connection parameters specific to a data source.

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	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.

Page Size

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).

Password

The password used to authenticate to Elasticsearch.

Data Type

string

Default Value

""

Remarks

The password used to authenticate to Elasticsearch.

Port

The port for the Elasticsearch REST server.

Data Type

string

Default Value

"9200"

Remarks

The port the Elasticsearch REST server is bound to.

Proxy Auth Scheme

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

Data Type

string

Default Value

"BASIC"

Remarks

This value specifies the authentication type to use to authenticate to the HTTP proxy specified by Proxy Server and Proxy Port.

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 Proxy Auto Detect to false, in addition to Proxy Server and Proxy Port. To authenticate, set Proxy Auth Scheme and set Proxy User and Proxy Password, 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 Firewall Type.

Proxy Auto Detect

This indicates whether to use the system proxy settings or not. Set ProxyAutoDetect to FALSE to use custom proxy settings. This takes precedence over other proxy settings.

Data Type

bool

Default Value

true

Remarks

By default, the adapter uses the system HTTP proxy. Set this to false if you want to connect to another proxy.

To connect to an HTTP proxy, see Proxy Server.

For other proxies, such as SOCKS or tunneling, see Firewall Type.

Proxy Exceptions

A semicolon separated list of hosts or IPs that will be exempt from connecting through the ProxyServer .

Data Type

string

Default Value

""

Remarks

The Proxy Server will be used for all addresses, except for addresses defined in this property. Use semicolons to separate entries.

Note that the adapter will use the system proxy settings by default, without further configuration needed; if you want to explicitly configure proxy exceptions for this connection, you will need to set Proxy Auto Detect to false, and configure Proxy Server and Proxy Port. To authenticate, set Proxy Auth Scheme and set Proxy User and Proxy Password, if needed.

Proxy Password

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 Proxy Server and Proxy Port. To specify the authentication type, set Proxy Auth Scheme.

If you are using HTTP authentication, additionally set Proxy User and ProxyPassword to HTTP proxy.

If you are using NTLM authentication, set Proxy User and Proxy Password to your Windows password. You may also need these to complete Kerberos authentication.

For SOCKS 5 authentication or tunneling, see Firewall Type.

By default, the adapter uses the system proxy. If you want to connect to another proxy, set Proxy Auto Detect to false.

Proxy Port

The TCP port the ProxyServer proxy is running on.

Data Type

string

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 Proxy Server. For other proxy types, see Firewall Type.

Proxy Server

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 Firewall Type.

By default, the adapter uses the system proxy. If you need to use another proxy, set Proxy Auto Detect to false.

Proxy SSL Type

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

Data Type

string

Default Value

"AUTO"

Remarks

This property determines when to use SSL for the connection to an HTTP proxy specified by Proxy Server. 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.

Proxy User

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

Data Type

string

Default Value

""

Remarks

The Proxy User and Proxy Password options are used to connect and authenticate against the HTTP proxy specified in Proxy Server.

You can select one of the available authentication types in Proxy Auth Scheme. If you are using HTTP authentication, set this to the username of a user recognized by the HTTP proxy. If you are using Windows or Kerberos authentication, set this property to a username in one of the following formats:

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

Query Passthrough

This option allows you to pass queries to Elasticsearch using Elasticsearch's Search DSL language, which includes Query DSL.

Data Type

bool

Default Value

false

Remarks

Setting this property to True enables the adapter to pass an Elasticsearch query as-is to 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 Query Passthrough queries, the metadata is determined by the data returned in the response. Row Scan Depth 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.

Row Scan Depth

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 Row Scan Depth rows will be queried and scanned to identify if any of the fields contain arrays.

When Query Passthrough 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.

Scroll Duration

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 Max Results.

Supported Time Units:

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

Server

The host name or IP address of the Elasticsearch REST server.

Data Type

string

Default Value

""

Remarks

The host name or IP address of the Elasticsearch REST server.

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

SSL Client Cert

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 SSL Client Cert field specifies the type of the certificate store specified by SSL Client Cert. If the store is password protected, specify the password in SSL Client Cert Password.

SSL Client Cert is used in conjunction with the SSL Client Cert Subject field in order to specify client certificates. If SSL Client Cert has a value, and SSL Client Cert Subject is set, a search for a certificate is initiated. Please refer to the SSL Client Cert Subject field for details.

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 (i.e. PKCS12 certificate store).

SSL Client Cert Password

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 in order to open the certificate store.

SSL Client Cert Subject

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 instance "CN=www.server.com, OU=test, C=US, E=support@cdata.com". Common fields and their meanings are displayed 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.

SSL Client Cert Type

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

Data Type

string

Default Value

""

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*: This store type is not available in Java.

MACHINE	For Windows, this specifies that the certificate store is a machine store. *Note*: 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*: this store type is only available in Java.
JKSBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in Java key store (JKS) format. *Note*: 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 certificates.
PPKFILE	The certificate store is the name of a file that contains a PPK (PuTTY Private Key).
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.

SSL Server Cert

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 will be rejected.

This property can take the 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)	34a929226ae0819f2ec14b4a3d904f801c bb150d

If not specified, any certificate trusted by the machine will be accepted. Use '*' to signify to accept all certificates (not recommended for security concerns).

Temporary Token Duration

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).

Timeout

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

Data Type

string

Default Value

"60"

Remarks

If the Timeout property is set to 0, operations do not time out: They run until they complete successfully or encounter an error condition.

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

User

The user who is authenticating to Elasticsearch.

Data Type

string

Default Value

""

Remarks

The user who is authenticating to Elasticsearch.

Logging

The adapter uses log4j to generate log files. The settings within the log4j configuration file will be used by the adapter to determine the type of messages to log. The following categories can be specified:

Error: Only error messages will be logged.

Info: Both Error and Info messages will be logged.

Debug: Error, Info, and Debug messages will be 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, i.e.
`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 will never log 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 will default to the minimum value for that particular category. For example, if Verbosity is set to a value less than three and the Debug category is specified, the Verbosity will default to 3.

Here is a breakdown 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, the following line from the TDV Server's log4j.properties file can be uncommented (default location of this file is: C:\Program Files\TIBCO\TDV Server <version>\conf\server).

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

The TDV Server will need to 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>\conf\server. Note reauthenticating to the TDV Studio will be required after restarting the server.

An example of the calls would be:

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

All logs for the adapter will be written to the "cs_cdata.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 accomplish this, navigate to Administrator -> Configuration to display the configuration window. Then expand Server -> Configuration -> Debugging and set the Debug Output Enabled option to True.

Using Kerberos

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

Authenticating with 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 will be 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 **won't** 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
3. **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 will be created when generating your ticket with MIT Kerberos Ticket Manager.
4. To obtain a ticket, open the MIT Kerberos Ticket Manager application, click
5. **Get Ticket, enter your principal name and password, then click OK**. If successful, ticket information will appear in Kerberos Ticket Manager and will now be stored in the credential cache file.
6. Now that the credential cache file has been created, the adapter will use 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 will use 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 this, 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 combo that you use to authenticate with Elasticsearch.

Cross-Realm Authentication

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 used for obtaining the service ticket).

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

Advanced Settings

Accessing NoSQL Tables

The adapter implements Automatic Schema Discovery that is highly configurable. The following sections outline the adapter's defaults and link to ways to further customize.

Flattening Nested JSON

By default, the adapter projects columns over the properties of objects, including objects nested in objects. Arrays are returned as JSON strings, by default. You can use the following properties to access array elements, including objects nested in arrays.

FlattenArrays: Set this property to the number of array elements that you want to return as column values. You can also use this property with FlattenObjects to extract the properties of objects nested in arrays.

FlattenObjects: By default, this is true; that is, the properties of objects and nested objects are returned as columns. When you set FlattenArrays, objects nested in the specified array elements are also flattened and returned as columns.

Other mechanisms for accessing nested objects are detailed in Searching with SQL.

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]

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.

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.

Connecting Through a Firewall or Proxy

To connect through the Windows system proxy, set only the connection properties needed to authenticate and connect. To connect to other proxies, set ProxyAutoDetect to false and in addition set the following.

To authenticate to an HTTP proxy, set ProxyAuthScheme, ProxyUser, and ProxyPassword, in addition to ProxyServer and ProxyPort.

To connect to other proxies, set FirewallType, FirewallServer, and FirewallPort. To tunnel the connection, set FirewallType to TUNNEL. To authenticate to a SOCKS proxy, set FirewallType to SOCKS5. Additionally, specify FirewallUser and FirewallPassword.

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.

Troubleshooting the Connection

To show adapter activity from query execution to HTTP calls, use Logfile and Verbosity. The examples of common connection errors below show how to use these properties to get more context. Contact the support team for help tracing the source of an error or circumventing a performance issue.

Authentication errors: Typically, recording a Logfile at Verbosity 4 is necessary to get full details on an authentication error.

Queries time out: A server that takes too long to respond will exceed the adapter's client-side timeout. Often, setting the Timeout property to a higher value will avoid a connection error. Another option is to disable the timeout by setting the property to 0. Setting Verbosity to 2 will show where the time is being spent.

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 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"}
```

SeeJSON 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 a breakdown 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>
```

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-Za-z0-9_:@].

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').

Transactions and Batching

Transactions are not currently supported.

Additionally, the adapter does not support batching of SQL statements. To execute multiple commands, you can create multiple instances and execute each separately. Or, use Batch Processing.

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 ] } [ , ... ]
  ]
  [
    LIMIT <expression>
    [
      { OFFSET | , }
      <expression>
    ]
  ]
} | SCOPE_IDENTITY()

  <expression> ::=
    | <column_reference>
    | @ <parameter>
    | ?
    | COUNT( * | { [ DISTINCT ] <expression> } )
    | { AVG | MAX | MIN | SUM | COUNT } ( <expression> )
    | <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 Account
```

2. Rename a column:

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

3. Search data:

```
SELECT * FROM Account WHERE Industry = 'Floppy Disks';
```

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

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

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 Account WHERE Industry = 'Floppy Disks';
```

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

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

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

```
SELECT COUNT(DISTINCT Name) FROM Account
```

8. Return the unique items matching the query criteria:

```
SELECT DISTINCT Name FROM Account
```

9. Summarize data:

```
SELECT Name, MAX(AnnualRevenue) FROM Account  GROUP BY Name
```

See Aggregate Functions for details.

10. Sort a result set in ascending order:

```
SELECT Id, Name FROM Account  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 Account 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 Account 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 Account WHERE
Industry = 'Floppy Disks'
```

AVG

Returns the average of the column values.

```
SELECT Name, AVG(AnnualRevenue) FROM Account WHERE Industry =
'Floppy Disks' GROUP BY Name
```

MIN

Returns the minimum column value.

```
SELECT MIN(AnnualRevenue), Name FROM Account WHERE Industry =
'Floppy Disks' GROUP BY Name
```

MAX

Returns the maximum column value.

```
SELECT Name, MAX(AnnualRevenue) FROM Account WHERE Industry =
'Floppy Disks' GROUP BY Name
```

SUM

Returns the total sum of the column values.

```
SELECT SUM(AnnualRevenue) FROM Account 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,"le
ft":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":"envel
ope","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(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_n
ame\",\"query\":\"\\\\\"Smith\\\\\"\"}}]}')
SELECT * FROM employee WHERE
DSLQuery('{\"query_string\":{\"default_field\":\"last_name\",\"que
ry\":\"\\\\\"Smith\\\\\"\"}}')
```

Elasticsearch Query:

```
{"bool":{"must":[{"query_string":{"default_field":"last_name","que
ry":"\"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":{"d
efault_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.

SELECT INTO Statements

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

Data Export with an SQL Query

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

```
SELECT Id, Name INTO "csv://Account.txt" FROM "Account" WHERE
Industry = 'Floppy Disks'
```

You can specify other formats in the file URI. The possible delimiters are tab, semicolon, and comma with the default being comma. The following example exports tab-separated values:

```
SELECT Id, Name INTO "csv://Account.txt;delimiter=tab" FROM
"Account" WHERE Industry = 'Floppy Disks'
```

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:

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


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

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

```
String cmd = "INSERT INTO Account (Name) VALUES (?)";
```

```
PreparedStatement pstmt =
connection.prepareStatement(cmd,Statement.RETURN_GENERATED_KEYS);
pstmt.setString(1, "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.

```
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.

```
String cmd = "UPDATE Account 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 will update an existing record or create 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 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 = "UPSERT INTO Account (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 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.

```
<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.

```
Connection connection =
DriverManager.getConnection("jdbc:elasticsearch:Server=127.0.0.1;P
ort=9200;",);
String cmd = "DELETE FROM Account WHERE Id = ?";
PreparedStatement pstmt = connection.prepareStatement(cmd);
pstmt.setString(1, "1");
int count=pstmt.executeUpdate();
connection.close();
```

EXECUTE Statements

To execute stored procedures, you can use EXECUTE or EXEC statements. EXEC and EXECUTE assign stored procedure inputs, referenced by name, to values or parameter names.

Stored Procedure Syntax

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

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

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

Example Statements

Reference stored procedure inputs by name:

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

Execute a parameterized stored procedure statement:

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

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 behind the scenes 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.

Stored Procedures

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

Stored Procedures

Stored procedures are available to complement the data available from the Data Model. It may be necessary to update data available from a view using a stored procedure because the data does not provide for direct, table-like, two-way updates. In these situations, the retrieval of the data is done using the appropriate view or table, while the update is done by calling a stored procedure. Stored procedures take a list of parameters and return back a dataset that contains the collection of tuples that constitute the response.

Elasticsearch Adapter Stored Procedures

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

CreateSchema

Creates a schema file for the collection.

Input

Name	Type	Description
SchemaName	*String*	The name of the schema (the Elasticsearch index).
TableName	*String*	The name of the collection (the Elasticsearch Type).
FileLocation	*String*	The folder path where the generated schema (RSD) file will be stored. When specified the TableName will be used as the schema file name.
FileName	*String*	The complete schema (RSD) file name of the generated schema. This input takes precedence of FileLocation.

Output

Name	Type	Description
Result	*String*	Returns Success or Failure.

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.

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.

Output

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)
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 Flatten Arrays)