TIBCO Data Virtualization®

Cosmos DB Adapter Guide

Version 8.5.0
Last Updated: July 5, 2021

Contents

TDV Cosmos DB Adapter

Cosmos DB Version Support

The adapter enables standards-based access to Cosmos DB.

Requirements and Restrictions

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

Getting Started

Deploying the Adapter

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

Connecting to Cosmos DB

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

Basic Tab

Connecting to Cosmos DB

Rest API (SQL API)

To obtain the connection string needed to connect to a Cosmos DB account using the SQL API, log in to the Azure Portal, select Azure Cosmos DB, and select your account. In the Settings section, click Connection String and set the following values:

- AccountEndpoint: The value should be the Cosmos DB account URL from the Keys blade of the Cosmos DB account.
- AccountKey: In the Azure portal, navigate to the Cosmos DB service and select your Azure Cosmos DB account. From the resource menu, go to the Keys page. Find the PRIMARY KEY value and set Token to this value.

Advanced Tab

Connection String Options

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.

Account Endpoint	The value should be the Cosmos DB account URL from the Keys blade of the Cosmos DB account.
Account Key	A master key token or a resource token for connecting to the Cosmos DB REST API.
Calculate Aggregates	Specifies whether will return the calculated value of the aggregates or grouped by partiton range.
Consistency Level	Denotes the type of token: master or resource.
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	By default, nested arrays are returned as strings of JSON. The FlattenArrays property can be used to flatten the elements of nested arrays into columns of their own. Set FlattenArrays to the number of elements you want to return from nested arrays.
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.
Location	A path to the directory that contains the schema files defining tables, views, and stored procedures.
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.
Multi Thread Count	Aggregate queries in partitioned collections will require parallel requests for different partition ranges. Set this to the number of parallel request to be issued in the same time.
Other	These hidden properties are used only in specific use cases.
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. This takes precedence over other proxy settings, so you'll need to set ProxyAutoDetect to FALSE in order use custom proxy settings.
Proxy Exceptions	A semicolon separated list of hosts or IPs that are 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.

Readonly	You can use this property to enforce read-only access to Cosmos DB from the provider.
Row Scan Depth	The maximum number of rows to scan to look for the columns available in a table. Set this property to gain more control over how the provider applies data types to collections.
Schema	Specify the Cosmos DB database you want to work with.
Separator Character	The character or characters used to denote hierarchy.
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.
Timeout	The value in seconds until the timeout error is thrown, canceling the operation.
Token Type	Denotes the type of token: master or resource.
Type Detection Scheme	Comma-separated options for how the provider will scan the data to determine the fields and datatypes in each document collection.

Account Endpoint

The value should be the Cosmos DB account URL from the Keys blade of the Cosmos DB account.

Data Type

string

Default Value

""

Remarks

The value should be the Cosmos DB account URL from the Keys blade of the Cosmos DB account.

Account Key

A master key token or a resource token for connecting to the Cosmos DB REST API.

Data Type

string

Default Value

""

Remarks

In the Azure portal, navigate to the Cosmos DB service and select your Azure Cosmos DB account. From the resource menu, go to the Keys page. Find the PRIMARY KEY value and set Token to this value.

Calculate Aggregates

Specifies whether will return the calculated value of the aggregates or grouped by partiton range.

Data Type

bool

Default Value

true

Remarks

Specifies whether will return the calculated value of the aggregates or grouped by partiton range.

Consistency Level

Denotes the type of token: master or resource.

Data Type

string

Default Value

"SESSION"

Remarks

The consistency level override for read options against documents and attachments. The valid values are: Strong, Bounded, Session, or Eventual (in order of strongest to weakest). The override must be the same or weaker than the account's configured consistency level.

The consistency level override for read options against documents and attachments. The valid values are: Strong, Bounded, Session, or Eventual (in order of strongest to weakest). The override must be the same or weaker than the account's configured consistency level.

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 FirewallServer and FirewallPort, following the authentication method specified by FirewallType.

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 FirewallServer to specify the name or IP address. Specify the protocol with FirewallType.

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 FirewallType: Use FirewallServer with this property to connect through SOCKS or do tunneling. Use ProxyServer to connect to an HTTP proxy.

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

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 FirewallServer proxy. Note that by default, the adapter connects to the system proxy; to disable this behavior and connect to one of the following proxy types, set ProxyAutoDetect to false.

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

To connect to HTTP proxies, use ProxyServer and ProxyPort. To authenticate to HTTP proxies, use ProxyAuthScheme, ProxyUser, and ProxyPassword.

Firewall User

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

Data Type

string

Default Value

""

Remarks

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

Flatten Arrays

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

Data Type

string

Default Value

"0"

Remarks

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

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

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

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

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

Column Name	Column Value
languages.0	FLOW-MATIC

Setting FlattenArrays to -1 will flatten all the elements of nested 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 dot to generate the column name.

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

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

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

Column Name	Column Value
grades.0.grade	A
grades.0.score	2

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 the table definitions identified by Automatic Schema Discovery. This property outputs schemas to .rsd files in the path specified by Location.

Available settings are the following:

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

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

Generate Schemas with SQL

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

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.

Editing Schemas

Schema files have a simple format that makes them easy to modify. See Custom Schema Definitions for an end-to-end guide using the Cosmos DB restaurants collection.

Using Dynamic Schemas Instead

If your data structures are volatile, consider setting GenerateSchemaFiles to **Never** and using dynamic schemas, which change based on the metadata retrieved when you connect. Also, note that you cannot execute Free-Form Queries queries to a table that has a static schema definition.

Location

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

Data Type

string

Default Value

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

Remarks

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

If left unspecified, the default location is "%APPDATA%\\CData\\CosmosDB 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

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.

Multi Thread Count

Aggregate queries in partitioned collections will require parallel requests for different partition ranges. Set this to the number of parallel request to be issued in the same time.

Data Type

string

Default Value

"5"

Remarks

Aggregate queries in partitioned collections will require parallel requests for different partition ranges. Set this to the number of parallel request to be issued in the same time.

Other

These hidden properties are used only in specific use cases.

Data Type

string

Default Value

""

Remarks

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

Specify multiple properties in a semicolon-separated list.

Integration and Formatting

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

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 ProxyServer and ProxyPort.

Note that the adapter will use the system proxy settings by default, without further configuration needed; if you want to connect to another proxy, you will need to set ProxyAutoDetect to false, in addition to ProxyServer and ProxyPort. To authenticate, set ProxyAuthScheme and set ProxyUser and ProxyPassword, if needed.

The authentication type can be one of the following:

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

If you need to use another authentication type, such as SOCKS 5 authentication, see Firewall Type.

Proxy Auto Detect

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

Data Type

bool

Default Value

true

Remarks

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

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

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

To connect to an HTTP proxy, see Proxy Server. For other proxies, such as SOCKS or tunneling, see Firewall Type.

Proxy Exceptions

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

Data Type

string

Default Value

""

Remarks

The ProxyServer is used for all addresses, except for addresses defined in this property. Use semicolons to separate entries.

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

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 ProxyServer and ProxyPort. To specify the authentication type, set ProxyAuthScheme.

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

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

For SOCKS 5 authentication or tunneling, see Firewall Type.

By default, the adapter uses the system proxy. If you want to connect to another proxy, set ProxyAutoDetect 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 ProxyServer. 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 ProxyAutoDetect 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 ProxyServer. This value can be AUTO, ALWAYS, NEVER, or TUNNEL. The applicable values are the following:

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

Proxy User

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

Data Type

string

Default Value

""

Remarks

The ProxyUser and ProxyPassword options are used to connect and authenticate against the HTTP proxy specified in ProxyServer.

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

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

Readonly

You can use this property to enforce read-only access to Cosmos DB 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 to look for the columns available in a table. Set this property to gain more control over how the provider applies data types to collections.

Data Type

string

Default Value

"100"

Remarks

Since Cosmos DB using the Cosmos DB API is schemaless, the columns in a table must be determined by scanning table rows. This value determines the maximum number of rows that will be scanned. 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.

Schema

Specify the Cosmos DB database you want to work with.

Data Type

string

Default Value

""

Remarks

Specify the Cosmos DB database you want to work with.

Separator Character

The character or characters used to denote hierarchy.

Data Type

string

Default Value

"."

Remarks

In order to flatten out hierarchical structures, the adapter needs some specifier that states the path to a column through the hierarchy. If this value is "." and a column comes back with the name address.city, this indicates that there is a mapped attribute with a child called city. If your data has columns that already use a single period within the attribute name, set the SeparatorCharacter to a different character or characters.

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

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

Designations of certificate stores are platform-dependent.

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

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

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

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

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 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 example, "CN=www.server.com, OU=test, C=US, E=support@company.com". The common fields and their meanings are shown below.

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

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

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 that this store type is not available in Java.
MACHINE	For Windows, this specifies that the certificate store is a machine store. Note that this store type is not available in Java.
PFXFILE	The certificate store is the name of a PFX (PKCS12) file containing certificates.
PFXBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in PFX (PKCS12) format.
JKSFILE	The certificate store is the name of a Java key store (JKS) file containing certificates. Note that this store type is only available in Java.
JKSBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in JKS format. Note that this store type is only available in Java.
PEMKEY_FILE	The certificate store is the name of a PEM-encoded file that contains a private key and an optional certificate.
PEMKEY_BLOB	The certificate store is a string (base64-encoded) that contains a private key and an optional certificate.

PUBLIC_KEY_FILE	The certificate store is the name of a file that contains a PEM- or DER-encoded public key certificate.
PUBLIC_KEY_BLOB	The certificate store is a string (base-64-encoded) that contains a PEM- or DER-encoded public key certificate.
SSHPUBLIC_KEY_FILE	The certificate store is the name of a file that contains an SSH-style public key.
SSHPUBLIC_KEY_BLOB	The certificate store is a string (base-64-encoded) that contains an SSH-style public key.
P7BFILE	The certificate store is the name of a PKCS7 file containing certificates.
PPKFILE	The certificate store is the name of a file that contains a PuTTY Private Key (PPK).
XMLFILE	The certificate store is the name of a file that contains a certificate in XML format.
XMLBLOB	The certificate store is a string that contains a certificate in XML format.

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 is rejected.

This property can take the following forms:

Description	Example

A full PEM Certificate (example shortened for brevity)	-----BEGIN CERTIFICATE----- MIIChTCCAe4CAQAwDQYJKoZIhv.....Qw== -----END CERTIFICATE-----
A path to a local file containing the certificate	C:\cert.cer
The public key (example shortened for brevity)	-----BEGIN RSA PUBLIC KEY----- MIGfMA0GCSq.....AQAB -----END RSA PUBLIC KEY-----
The MD5 Thumbprint (hex values can also be either space or colon separated)	ecadbdda5a1529c58a1e9e09828d70e4
The SHA1 Thumbprint (hex values can also be either space or colon separated)	34a929226ae0819f2ec14b4a3d904f801cbb150d

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

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

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

Timeout

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

Data Type

string

Default Value

"60"

Remarks

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

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

Token Type

Denotes the type of token: master or resource.

Data Type

string

Default Value

"master"

Remarks

The master key is created during the creation of an account. There are two sets of master keys, the primary key and the secondary key. The administrator of the account can then exercise key rotation using the secondary key. In addition, the account administrator can also regenerate the keys as needed.

Resource tokens are created when users in a database are set up with access permissions for precise access control on a resource, also known as a permission resource. A permission resource contains a hash resource token constructed with the information regarding the resource path and access type a user has access to. The permission resource token is time bound and the validity period can be overridden. When a permission resource is acted upon on (POST, GET, PUT), a new resource token is generated.

Type Detection Scheme

Comma-separated options for how the provider will scan the data to determine the fields and datatypes in each document collection.

Data Type

string

Default Value

"RowScan,Recent"

	Remarks
None	Setting TypeDetectionScheme to None will return all columns as a string type. Cannot be combined with other options.
RowScan	Setting TypeDetectionScheme to RowScan will scan rows to heuristically determine the data type. The RowScanDepth determines the number of rows to be scanned. Can be used with Recent.
Recent	Setting TypeDetectionScheme to Recent will determine whether RowScan is executed on the most recent documents in the collection. Can be used with RowScan.

Logging

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

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

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

```
Verbosity=4;
```

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

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

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

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

The following list is 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 Cosmos DB Adapter

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

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

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

Here is an example of the calls:

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

All logs for the adapter are written to the "cs_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 set this option, navigate to **Administrator > Configuration**. Select **Server > Configuration > Debugging** and set the Debug Output Enabled option to **True**.

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. 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 NoSQL Database.

Fine Tuning Data Access

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

- RowScanDepth: This property determines the number of rows that will be scanned to detect column data types when generating table metadata.
- TypeDetectionScheme: This property allows more control over the strategy implemented by the RowScanDepth property.
- 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. Or, you can generate schemas as you execute SELECT queries to tables. The resulting schemas are based on the connection properties you use to configure Automatic Schema Discovery.

To use the resulting schema files, set the Location property to the folder containing the schemas.

Customizing the SSL Configuration

By default, the adapter attempts to negotiate SSL/TLS by checking the server's certificate against the system's trusted certificate store. To specify another certificate, see the SSL Client Cert property for the available formats to do so.

Connecting Through a Firewall or Proxy

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

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.
- **The certificate presented by the server cannot be validated**: This error indicates that the adapter cannot validate the server's certificate through the chain of trust. (If you are using a self-signed certificate, there is only one certificate in the chain).
- To resolve this error, you must verify yourself that the certificate can be trusted and specify to the adapter that you trust the certificate. One way you can specify that you trust a certificate is to add the certificate to the trusted system store; another is to set SSLServerCert.

Changes in 2019

DocumentDB API

The Cosmos DB Adapter has been completely rewritten. We now support querying documents using SQL (Structured Query Language) as a JSON query language on SQL API accounts instead of using the MongoDB API.

MongoDB API

We are no longer supporting the MongoDB API. Instead you can use the MongoDB driver in order to connect with a MongoDB account. You will find the required connection information in the help documentation of the MongoDB driver.

NoSQL Database

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

Working with Cosmos DB Objects as Tables

The adapter models the schemaless Cosmos DB objects into relational tables and translates SQL queries into Cosmos DB queries to get the requested data. See Query Mapping (Sql API) for more details on how various Cosmos DB operations are represented as SQL.

Discovering Schemas Automatically

The Automatic Schema Discovery scheme automatically finds the data types in a Cosmos DB object by scanning a configured number of rows of the object. You can use RowScanDepth, FlattenArrays, and FlattenObjects to control the relational representation of the collections in Cosmos DB. You can also write Free-Form Queries not tied to the schema.

Customizing Schemas

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

Set GenerateSchemaFiles to save the detected schemas as simple configuration files that are easy to extend. You can persist schemas for all collections in the database or for the results of SELECT queries.

Automatic Schema Discovery

By default the adapter automatically infers a relational schema by inspecting a series of Cosmos DB documents in a collection. You can use the RowScanDepth property to define the number of documents the adapter will scan to do so. The columns identified during the discovery process depend on the FlattenArrays and FlattenObjects properties.

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

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

This document will be represented by the following columns:

Column Name	Data Type	Example Value
id	Integer	12
name	String	Lohia Manufacturers Inc.
address.street	String	Main Street
address.city	String	Chapel Hill
address.state	String	NC

offices	String	["Chapel Hill", "London", "New York"]
annual_revenue	Double	35,600,000

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

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

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

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

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

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

Free-Form Queries

As discussed in Automatic Schema Discovery, intuited table schemas enable SQL access to unstructured Cosmos DB data. JSON Functions enable you to use standard JSON functions to summarize Cosmos DB data and extract values from any nested structures. Custom Schema Definitions enable you to define static tables and give you more granular control over the relational view of your data; for example, you can write schemas defining parent/child tables or fact/dimension tables. However, you are not limited to these schemes.

After connecting you can query any nested structure without flattening the data. Any relations that you can access with FlattenArrays and FlattenObjects can also be accessed with an ad hoc SQL query.

Let's consider an example document from the following Restaurant data set:

```
{
```

```
  "address": {
    "building": "1007",
    "coord": [
      -73.856077,
      40.848447
    ],
    "street": "Morris Park Ave",
    "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
    {
      "grade": "A",
      "score": 2,
      "date": {
        "$date": "1393804800000"
      }
    },
    {
      "date": {
        "$date": "1378857600000"
      },
      "grade": "B",
      "score": 6
    },
    {
      "score": 10,
      "date": {
        "$date": "1358985600000"
      },
      "grade": "C"
    }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```

You can access any nested structure in this document as a column. Use the dot notation to drill down to the values you want to access as shown in the query below. Note that arrays have a zero-based index. For example, the following query retrieves the second grade for the restaurant in the example:

```
SELECT "address.building", "grades.1.grade" FROM restaurants
WHERE restaurant_id = '30075445'
```

The preceding query returns the following results:

Column Name	Data Type	Example Value
address.building	String	1007
grades.1.grade	String	A

Vertical Flattening

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

```
{
  "_id" : ObjectId("568c37b748ddf53c5ed98932"),
  "address" : {
    "building" : "1007",
    "coord" : [-73.856077, 40.848447],
    "street" : "Morris Park Ave",
    "zipcode" : "10462"
  },
  "borough" : "Bronx",
  "cuisine" : "Bakery",
  "grades" : [{
      "date" : ISODate("2014-03-03T00:00:00Z"),
      "grade" : "A",
      "score" : 2
    }, {
      "date" : ISODate("2013-09-11T00:00:00Z"),
      "grade" : "A",
      "score" : 6
    }, {
      "date" : ISODate("2013-01-24T00:00:00Z"),
      "grade" : "A",
      "score" : 10
```

```
    }, {
      "date" : ISODate("2011-11-23T00:00:00Z"),
      "grade" : "A",
      "score" : 9
    }, {
      "date" : ISODate("2011-03-10T00:00:00Z"),
      "grade" : "B",
      "score" : 14
    }],
  "name" : "Morris Park Bake Shop",
  "restaurant_id" : "30075445"
}
```

Vertical flattening will allow you to retrieve the grades array as a separate table:

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

This query returns the following data set:

date	grade	score	P_id	_index
2014-03-03T00:00:00.000Z	A	2	568c37b748ddf53c5ed98932	1
2013-09-11T00:00:00.000Z	A	6	568c37b748ddf53c5ed98932	2
2013-01-24T00:00:00.000Z	A	10	568c37b748ddf53c5ed98932	3

You may also want to include information from the base restaurants table. You can do this with a join. Flattened arrays can only be joined with the root document. The adapter expects the left part of the join is the array document you want to flatten vertically. Disable SupportEnhancedSQL to join nested Cosmos DB documents -- this type of query is supported through the Cosmos DB API.

```
SELECT "restaurants"."restaurant_id", "restaurants.grades".*
FROM "restaurants.grades" JOIN "restaurants" WHERE
"restaurants".name = 'Morris Park Bake Shop'
```

This query returns the following data set:

restaurant_id	date	grade	score	P_id	_index
30075445	2014-03-03T00:00:00.000Z	A	2	568c37b748ddf53c5ed98932	1
30075445	2013-09-11T00:00:00.000Z	A	6	568c37b748ddf53c5ed98932	2

30075445	2013-01-24T00:00:00.000Z	A	10	568c37b748ddf53c5ed98932	3
30075445	2011-11-23T00:00:00.000Z	A	9	568c37b748ddf53c5ed98932	4
30075445	2011-03-10T00:00:00.000Z	B	14	568c37b748ddf53c5ed98932	5

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 Customers;

The query above will return the entire document as shown.

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

Sql API Built-In Functions

Cosmos DB also supports a number of built-in functions for common operations, that can be used inside queries.

Function group	Operations
Mathematical functions	ABS, CEILING, EXP, FLOOR, LOG, LOG10, POWER, ROUND, SIGN, SQRT, SQUARE, TRUNC, ACOS, ASIN, ATAN, ATN2, COS, COT, DEGREES, PI, RADIANS, SIN, and TAN
Type checking functions	IS_ARRAY, IS_BOOL, IS_NULL, IS_NUMBER, IS_OBJECT, IS_STRING, IS_DEFINED, and IS_PRIMITIVE
String functions	CONCAT, CONTAINS, ENDSWITH, INDEX_OF, LEFT, LENGTH, LOWER, LTRIM, REPLACE, REPLICATE, REVERSE, RIGHT, RTRIM, STARTSWITH, SUBSTRING, and UPPER
Array functions	ARRAY_CONCAT, ARRAY_CONTAINS, ARRAY_LENGTH, and ARRAY_SLICE

Mathematical functions

The mathematical functions each perform a calculation, based on input values that are provided as arguments, and return a numeric value. Here's a table of supported built-in mathematical functions.

Usage	Description

ABS (num_expr)	Returns the absolute (positive) value of the specified numeric expression.
CEILING (num_expr)	Returns the smallest integer value greater than, or equal to, the specified numeric expression.
FLOOR (num_expr)	Returns the largest integer less than or equal to the specified numeric expression.
EXP (num_expr)	Returns the exponent of the specified numeric expression.
LOG (num_expr [,base])	Returns the natural logarithm of the specified numeric expression, or the logarithm using the specified base
LOG10 (num_expr)	Returns the base-10 logarithmic value of the specified numeric expression.
ROUND (num_expr)	Returns a numeric value, rounded to the closest integer value.
TRUNC (num_expr)	Returns a numeric value, truncated to the closest integer value.
SQRT (num_expr)	Returns the square root of the specified numeric expression.
SQUARE (num_expr)	Returns the square of the specified numeric expression.
POWER (num_expr, num_expr)	Returns the power of the specified numeric expression to the value specified.
SIGN (num_expr)	Returns the sign value (-1, 0, 1) of the specified numeric expression.
ACOS (num_expr)	Returns the angle, in radians, whose cosine is the specified numeric expression; also called arccosine.
ASIN (num_expr)	Returns the angle, in radians, whose sine is the specified numeric expression. This is also called arcsine.
ATAN (num_expr)	Returns the angle, in radians, whose tangent is the specified numeric expression. This is also called arctangent.

ATN2 (num_expr)	Returns the angle, in radians, between the positive x-axis and the ray from the origin to the point (y, x), where x and y are the values of the two specified float expressions.
COS (num_expr)	Returns the trigonometric cosine of the specified angle, in radians, in the specified expression.
COT (num_expr)	Returns the trigonometric cotangent of the specified angle, in radians, in the specified numeric expression.
DEGREES (num_expr)	Returns the corresponding angle in degrees for an angle specified in radians.
PI ()	Returns the constant value of PI.
RADIANS (num_expr)	Returns radians when a numeric expression, in degrees, is entered.
SIN (num_expr)	Returns the trigonometric sine of the specified angle, in radians, in the specified expression.
TAN (num_expr)	Returns the tangent of the input expression, in the specified expression.

Type checking functions

The type checking functions allow you to check the type of an expression within SQL queries. Type checking functions can be used to determine the type of properties within documents on the fly when it is variable or unknown. Here's a table of supported built-in type checking functions.

Usage	Description
IS_ARRAY (expr)	Returns a Boolean indicating if the type of the value is an array.
IS_BOOL (expr)	Returns a Boolean indicating if the type of the value is a Boolean.
IS_NULL (expr)	Returns a Boolean indicating if the type of the value is null.
IS_NUMBER (expr)	Returns a Boolean indicating if the type of the value is a number.
IS_OBJECT (expr)	Returns a Boolean indicating if the type of the value is a JSON object.
IS_STRING (expr)	Returns a Boolean indicating if the type of the value is a string.

IS_DEFINED (expr)	Returns a Boolean indicating if the property has been assigned a value.
IS_PRIMITIVE (expr)	Returns a Boolean indicating if the type of the value is a string, number, Boolean or null.

String functions

The following scalar functions perform an operation on a string input value and return a string, numeric or Boolean value. Here's a table of built-in string functions:

Usage	Description
LENGTH (str_expr)	Returns the number of characters of the specified string expression
CONCAT (str_expr, str_expr [, str_expr])	Returns a string that is the result of concatenating two or more string values.
SUBSTRING (str_expr, num_expr, num_expr)	Returns part of a string expression.
STARTSWITH (str_expr, str_expr)	Returns a Boolean indicating whether the first string expression starts with the second
ENDSWITH (str_expr, str_expr)	Returns a Boolean indicating whether the first string expression ends with the second
CONTAINS (str_expr, str_expr)	Returns a Boolean indicating whether the first string expression contains the second.
INDEX_OF (str_expr, str_expr)	Returns the starting position of the first occurrence of the second string expression within the first specified string expression, or -1 if the string is not found.
LEFT (str_expr, num_expr)	Returns the left part of a string with the specified number of characters.
RIGHT (str_expr, num_expr)	Returns the right part of a string with the specified number of characters.
LTRIM (str_expr)	Returns a string expression after it removes leading blanks.

RTRIM (str_expr)	Returns a string expression after truncating all trailing blanks.
LOWER (str_expr)	Returns a string expression after converting uppercase character data to lowercase.
UPPER (str_expr)	Returns a string expression after converting lowercase character data to uppercase.
REPLACE (str_expr, str_expr, str_expr)	Replaces all occurrences of a specified string value with another string value.
REPLICATE (str_expr, num_expr)	Repeats a string value a specified number of times.
REVERSE (str_expr)	Returns the reverse order of a string value.

Array functions

The following scalar functions perform an operation on an array input value and return numeric, Boolean or array value. Here's a table of built-in array functions:

Usage	Description
ARRAY_LENGTH (arr_expr)	Returns the number of elements of the specified array expression.
ARRAY_CONCAT (arr_expr, arr_expr [, arr_expr])	Returns an array that is the result of concatenating two or more array values.
ARRAY_CONTAINS (arr_expr, expr [, bool_expr])	Returns a Boolean indicating whether the array contains the specified value. Can specify if the match is full or partial.
ARRAY_SLICE (arr_expr, num_expr [, num_expr])	Returns part of an array expression.

Query Mapping (Sql API)

The adapter maps SQL queries into the corresponding Cosmos DB SQL API queries. A detailed description of all the transformations is out of scope, but we will describe some of the common elements that are used. The adapter takes advantage of SQL API features such as the aggregation framework to compute the desired results.

SELECT Queries

Since all requests can be submitted to a specific collection, we can send any constant string as table name to the API. Following the Azure Portal standard we are using the "C" character as table name.

SQL Query	Sql API Query
SELECT id, name FROM Users	SELECT C.id, C.name FROM C
SELECT * FROM Users WHERE name = 'A'	SELECT * FROM C WHERE C.name = 'A'
SELECT * FROM Users WHERE name = 'A' OR email = 'zoe55@gmail.com'	SELECT * FROM C WHERE C.name = 'A' OR C.email = 'zoe55@gmail.com'
SELECT id, grantamt FROM WorldBank WHERE grantamt IN (4500000, 85400000) OR grantamt = 16200000	SELECT C.id, C.grantamt FROM C WHERE C.grantamt IN (4500000, 85400000) OR C.grantamt = 16200000
SELECT * FROM WorldBank WHERE CountryCode = 'A' ORDER BY TotalCommAmt ASC	SELECT * FROM C WHERE C.countrycode = 'AL' ORDER BY C.totalcommamt ASC
SELECT * FROM WorldBank WHERE CountryCode = 'A' ORDER BY TotalCommAmt DESC	SELECT * FROM C WHERE C.countrycode = 'AL' ORDER BY C.totalcommamt DESC

Aggregate Queries

The adapter makes extensive use of this for various aggregate queries. See some examples below:

SQL Query	Sql API Query
SELECT COUNT(grantamt) AS COUNT_GRAMT FROM WorldBank	SELECT COUNT(C.grantamt) AS COUNT_GRAMT FROM C
SELECT SUM(grantamt) AS SUM_GRAMT FROM WorldBank	SELECT SUM(C.grantamt) AS SUM_GRAMT FROM C

Built-In functions

SQL Query	Sql API Query
SELECT IS_NUMBER(grantamt) AS ISN_ATTR, IS_NUMBER(id) AS ISN_ID FROM WorldBank	SELECT IS_NUMBER(C.grantamt) AS ISN_ATTR, IS_NUMBER(C.id) AS ISN_ID FROM C
SELECT POWER(totalamt, 2) AS POWERS_A, LENGTH(id) AS LENGTH_ID, PI() AS ThePI FROM WorldBank	SELECT POWER(C.totalamt, 2) AS POWERS_A, LENGTH(C.id) AS LENGTH_ID, PI() AS ThePI FROM C

Custom Schema Definitions

You can extend the table schemas created with Automatic Schema Discovery by saving them into schema files. The schema files have a simple format that makes the schemas to edit.

Generating Schema Files

Set GenerateSchemaFiles to "OnStart" to persist schemas for all tables when you connect. You can also generate table schemas as needed: Set GenerateSchemaFiles to "OnUse" and execute a SELECT query to the table.

For example, consider a schema for the restaurants data set. This is a sample data set provided by Cosmos DB.

Below is an example document from the collection:

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

```
}
```

Customizing a Schema

When GenerateSchemaFiles is set, the adapter saves schemas into the folder specified by the Location property. You can then change column behavior in the resulting schema.

The following schema uses the *other:bsonpath* property to define where in the collection to retrieve the data for a particular column. Using this model you can flatten arbitrary levels of hierarchy.

The *collection* attribute specifies the collection to parse. The *collection* attribute gives you the flexibility to use multiple schemas for the same collection. If *collection* is not specified, the filename determines the collection that is parsed.

Below are the corresponding column definitions for the restaurants data set. In Custom Schema Example, you will find the complete schema.

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

  <rsb:info title="StaticRestaurants" description="Custom
Schema for the restaurants data set.">
    <!-- Column definitions -->
    <attr  name="borough"   xs:type="string"
other:bsonpath="$.borough"              />
    <attr  name="cuisine"   xs:type="string"
other:bsonpath="$.cuisine"              />
    <attr  name="building"  xs:type="string"
other:bsonpath="$.address.building"     />
    <attr  name="street"    xs:type="string"
other:bsonpath="$.address.street"       />
    <attr  name="latitude"  xs:type="double"
other:bsonpath="$.address.coord.0"      />
    <attr  name="longitude" xs:type="double"
other:bsonpath="$.address.coord.1"      />
    <input name="rows@next" desc="Internal attribute used
for paging through data."  />
  </rsb:info>

  <rsb:set attr="collection" value="restaurants"/>

</rsb:script>
```

Custom Schema Example

In this section is a complete schema. The *info* section enables a relational view of a Cosmos DB object. For more details, see Custom Schema Definitions. The table below allows the SELECT, INSERT, UPDATE, and DELETE commands as implemented in the GET, POST, MERGE, and DELETE sections of the schema below.

Use the *collection* attribute to specify the name of the collection you want to parse. You can use the *collection* attribute to define multiple schemas for the same collection.

If *collection* is not specified, the filename determines the collection that is parsed.

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

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

  <rsb:info title="StaticRestaurants" description="Custom
Schema for the restaurants data set.">
    <!-- Column definitions -->
    <attr  name="borough"   xs:type="string"
other:bsonpath="$.borough"              />
    <attr  name="cuisine"   xs:type="string"
other:bsonpath="$.cuisine"              />
    <attr  name="building"  xs:type="string"
other:bsonpath="$.address.building"     />
    <attr  name="street"    xs:type="string"
other:bsonpath="$.address.street"       />
    <attr  name="latitude"  xs:type="double"
other:bsonpath="$.address.coord.0"      />
    <attr  name="longitude" xs:type="double"
other:bsonpath="$.address.coord.1"      />
    <input name="rows@next" desc="Internal attribute used
for paging through data."  />
  </rsb:info>

  <rsb:set attr="collection" value="restaurants"/>

  <rsb:script method="GET">
    <rsb:call op="cosmosdbadoSysData">
      <rsb:push />
```

```
    </rsb:call>
  </rsb:script>

  <rsb:script method="POST">
    <rsb:call op="cosmosdbadoSysData">
      <rsb:push />
    </rsb:call>
  </rsb:script>

  <rsb:script method="MERGE">
    <rsb:call op="cosmosdbadoSysData">
      <rsb:push />
    </rsb:call>
  </rsb:script>

  <rsb:script method="DELETE">
    <rsb:call op="cosmosdbadoSysData">
      <rsb:push />
    </rsb:call>
  </rsb:script>

</rsb:script>
```

Stored Procedures

Stored procedures are available to complement the data available from the NoSQL Database. 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.

Cosmos DB Adapter Stored Procedures

Name	Description
CreateSchema	Creates a schema file for the collection.

CreateSchema

Creates a schema file for the collection.

Input

Name	Type	Description
SchemaName	*String*	The schema of the collection.
TableName	*String*	The name of the collection.
FileName	*String*	The full name of the generated schema.

Result Set Columns

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

SQL Compliance

The Cosmos DB Adapter supports several operations on data, including querying, deleting, modifying, and inserting.

SELECT Statements

See SELECT Statements for a syntax reference and examples.

See NoSQL Database for information on the capabilities of the Cosmos DB API.

INSERT Statements

See INSERT Statements for a syntax reference and examples.

UPDATE Statements

The primary key _id is required to update a record. See UPDATE Statements for a syntax reference and examples.

DELETE Statements

The primary key _id is required to delete a record. See DELETE Statements for a syntax reference and examples.

EXECUTE Statements

Use EXECUTE or EXEC statements to execute stored procedures. See EXECUTE Statements for a syntax reference and examples.

Names and Quoting

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

SELECT Statements

A SELECT statement can consist of the following basic clauses.

- SELECT
- INTO
- FROM
- JOIN
- WHERE
- GROUP BY
- HAVING
- UNION
- ORDER BY
- LIMIT

SELECT Syntax

The following syntax diagram outlines the syntax supported by the Cosmos DB adapter:

```
SELECT {
  [ TOP <numeric_literal> ]
  {
    *
    | {
        <expression> [ [ AS ] <column_reference> ]
        | { <table_name> | <correlation_name> } .*
      } [ , ... ]
  }
  [ INTO csv:// [ filename= ] <file_path> [ ;delimiter=tab ]
]
  {
    FROM <table_reference> [ [ AS ] <identifier> ]
  } [ , ... ]
  [
   JOIN <table_reference> [ ON <search_condition> ] [ [ AS ]
<identifier> ]
  ] [ ... ]
  [ WHERE <search_condition> ]
  [
    LIMIT <expression>
  ]
} | SCOPE_IDENTITY()

  <expression> ::=
    | <column_reference>
    | @ <parameter>
    | ?
    | COUNT( * | { [ DISTINCT ] <expression> } )
    | { AVG | MAX | MIN | SUM | COUNT } ( <expression> )
    | NULLIF ( <expression> , <expression> )
    | COALESCE ( <expression> , ... )
    | CASE <expression>
        WHEN { <expression> | <search_condition> } THEN {
<expression> | NULL } [ ... ]
    [ ELSE { <expression> | NULL } ]
      END
```

```
    | <literal>
    | <sql_function>

  <search_condition> ::=
    {
      <expression> { = | > | < | >= | <= | <> | != | IN | AND
| OR } [ <expression> ]
    } [ { AND | OR } ... ]
```

Examples

1. Return all columns:
   ```
   SELECT * FROM Customers
   ```
2. Rename a column:
   ```
   SELECT "CompanyName" AS MY_CompanyName FROM Customers
   ```
3. Cast a column's data as a different data type:
   ```
   SELECT CAST(Balance AS VARCHAR) AS Str_Balance FROM
   Customers
   ```
4. Search data:
   ```
   SELECT * FROM Customers WHERE Country = 'US';
   ```
5. The Cosmos DB APIs support the following operators in the WHERE clause: =, >, <, >=, <=, <>, !=, IN, AND, OR.
   ```
   SELECT * FROM Customers WHERE Country = 'US';
   ```
6. Return the number of items matching the query criteria:
   ```
   SELECT COUNT(*) AS MyCount FROM Customers
   ```
7. Summarize data:
   ```
   SELECT  MAX(Balance) FROM Customers
   ```

See Aggregate Functions for details.

8. Retrieve data from multiple tables.
   ```
   SELECT "restaurants"."restaurant_id",
   "restaurants".name, "restaurants.grades".* FROM
   "restaurants.grades" JOIN "restaurants" WHERE
   "restaurants".name = 'Morris Park Bake Shop'
   ```

See JOIN Queries for details.

Aggregate Functions

Examples of Aggregate Functions

Below are several examples of SQL aggregate functions.

COUNT

Returns the number of rows matching the query criteria.

```
SELECT COUNT(*) FROM Customers WHERE Country = 'US'
```

AVG

Returns the average of the column values.

```
SELECT  AVG(Balance) FROM Customers WHERE Country = 'US'
```

MIN

Returns the minimum column value.

```
SELECT MIN(Balance) FROM Customers WHERE Country = 'US'
```

MAX

Returns the maximum column value.

```
SELECT  MAX(Balance) FROM Customers WHERE Country = 'US'
```

SUM

Returns the total sum of the column values.

```
SELECT SUM(Balance) FROM Customers WHERE Country = 'US'
```

JOIN Queries

The Cosmos DB Adapter supports joins of a nested array with its parent document.

Joining Nested Structures

The adapter expects the left part of the join is the array document you want to flatten vertically. This type of query is supported through the Cosmos DB API.

For example, consider the following query from Cosmos DB's restaurants collection:

```
SELECT "restaurants"."restaurant_id", "restaurants".name,
"restaurants.grades".*
FROM "restaurants.grades"
JOIN "restaurants"
WHERE "restaurants".name = 'Morris Park Bake Shop'
```

See Vertical Flattening for more details.

Projection Functions

ABS(numeric_expr)

Returns the absolute (positive) value of the specified numeric expression.

- numeric_expr: A numeric expression.

ACOS(numeric_expr)

Returns the angle, in radians, whose cosine is the specified numeric expression; also called arccosine.

- numeric_expr: A numeric expression.

ASIN(numeric_expr)

Returns the angle, in radians, whose sine is the specified numeric expression. This is also called arcsine.

- numeric_expr: A numeric expression.

ATAN(numeric_expr)

Returns the angle, in radians, whose tangent is the specified numeric expression. This is also called arctangent.

- numeric_expr: A numeric expression.

CEILING(numeric_expr)

Returns the smallest integer value greater than, or equal to, the specified numeric expression.

- numeric_expr: A numeric expression.

COS(numeric_expr)

Returns the trigonometric cosine of the specified angle, in radians, in the specified expression.

- numeric_expr: A numeric expression.

COT(numeric_expr)

Returns the trigonometric cotangent of the specified angle, in radians, in the specified numeric expression.

- numeric_expr: A numeric expression.

DEGREES(numeric_expr)

Returns the corresponding angle in degrees for an angle specified in radians.

- numeric_expr: A numeric expression.

FLOOR(numeric_expr)

Returns the largest integer less than or equal to the specified numeric expression.

- numeric_expr: A numeric expression.

EXP(numeric_expr)

Returns the exponential value of the specified numeric expression.

- numeric_expr: A numeric expression.

LOG10(numeric_expr)

Returns the base-10 logarithm of the specified numeric expression.

- numeric_expr: A numeric expression.

RADIANS(numeric_expr)

Returns radians when a numeric expression, in degrees, is entered.

- numeric_expr: A numeric expression.

ROUND(numeric_expr)

Returns a numeric value, rounded to the closest integer value.

- numeric_expr: A numeric expression.

SIGN(numeric_expr)

Returns the positive (+1), zero (0), or negative (-1) sign of the specified numeric expression.

- numeric_expr: A numeric expression.

SIN(numeric_expr)

Returns the trigonometric sine of the specified angle, in radians, in the specified expression.

- numeric_expr: A numeric expression.

SQRT(numeric_expr)

Returns the square root of the specified numeric value.

- numeric_expr: A numeric expression.

SQUARE(numeric_expr)

Returns the square of the specified numeric value.

- numeric_expr: A numeric expression.

TAN(numeric_expr)

Returns the tangent of the specified angle, in radians, in the specified expression.

- numeric_expr: A numeric expression.

TRUNC(numeric_expr)

Returns a numeric value, truncated to the closest integer value.

- numeric_expr: A numeric expression.

ATAN2(y_expr, x_expr)

Returns the principal value of the arc tangent of y/x, expressed in radians.

- y_expr: The y numeric expression.
- x_expr: The x numeric expression.

LOG(numeric_expr [, base])

Returns the natural logarithm of the specified numeric expression.

- numeric_expr: A numeric expression.
- base: Optional numeric argument that sets the base for the logarithm.

PI()

Returns the constant value of PI.

POWER(numeric_expr, power_expr)

Returns the value of the specified expression to the specified power.

- numeric_expr: A numeric expression.
- power_expr: Is the power to which to raise numeric_expr.

IS_ARRAY(expr)

Returns a Boolean value indicating if the type of the specified expression is an array.

- expr: Any valid expression.

IS_BOOL(expr)

Returns a Boolean value indicating if the type of the specified expression is a Boolean.

- expr: Any valid expression.

IS_DEFINED(expr)

Returns a Boolean indicating if the property has been assigned a value.

- expr: Any valid expression.

IS_NULL(expr)

Returns a Boolean value indicating if the type of the specified expression is null.

- expr: Any valid expression.

IS_NUMBER(expr)

Returns a Boolean value indicating if the type of the specified expression is a number.

- expr: Any valid expression.

IS_OBJECT(expr)

Returns a Boolean value indicating if the type of the specified expression is a JSON object.

- **expr**: Any valid expression.

IS_PRIMITIVE(expr)

Returns a Boolean value indicating if the type of the specified expression is a primitive (string, Boolean, numeric, or null).

- **expr**: Any valid expression.

IS_STRING(expr)

Returns a Boolean value indicating if the type of the specified expression is a string.

- **expr**: Any valid expression.

CONCAT(str1, str2 [, str3] [, ...])

Returns a string that is the result of concatenating two or more string values.

- **str1**: The first string to concatenate.
- **str2**: The second string to concatenate.
- **str3**: The third string to concatenate.

CONTAINS(str1, str2)

Returns a Boolean indicating whether the first string expression contains the second.

- **str1**: The string to search in.
- **str2**: The string to search for.

ENDSWITH(str1, str2)

Returns a Boolean indicating whether the first string expression ends with the second.

- **str1**: The string to search in.
- **str2**: The string to search for.

INDEX_OF(str1, str2)

Returns the starting position of the first occurrence of the second string expression within the first specified string expression, or -1 if the string is not found.

- **str1**: The string to search in.
- **str2**: The string to search for.

LEFT(str, num_expr)

Returns the left part of a string with the specified number of characters.

- **str**: A valid string expression.
- **num_expr**: The number of characters to return.

LENGTH(str)

Returns the number of characters of the specified string expression.

- **str**: Any valid string expression.

LOWER(str)

Returns a string expression after converting uppercase character data to lowercase.

- **str**: Any valid string expression.

LTRIM(str)

Returns a string expression after it removes leading blanks.

- **str**: Any valid string expression.

REPLACE(original_value, from_value, to_value)

Replaces all occurrences of a specified string value with another string value.

- **original_value**: The string to search in.
- **from_value**: The string to search for.
- **to_value**: The string to replace instances of from_value.

REPLICATE(str, repeat_num)

Repeats a string value a specified number of times.

- **str**: The string expression to repeat.
- **repeat_num**: The number of times to repeat the str expression.

REVERSE(str)

Returns the reverse order of a string value.

- **str**: Any valid string expression.

RIGHT(str, num_expr)

Returns the right part of a string with the specified number of characters.

- **str**: Any valid string expression.
- **num_expr**: The starting index.

RTRIM(str)

Returns a string expression after it removes trailing blanks.

- **str**: Any valid string expression.

STARTSWITH(str1, str2)

Returns a Boolean indicating whether the first string expression starts with the second.

- **str1**: The string to search in.
- **str2**: The string to search for.

SUBSTRING(str, start_index, length)

Returns part of a string expression starting at the specified character zero-based position and continues to the specified length, or to the end of the string.

- **str**: Any valid string expression.
- **start_index**: The starting index.
- **length**: The length of the string to return.

TOSTRING(expr)

Returns a string representation of scalar expression.

- **expr**: Any valid expression.

TRIM(str)

Returns a string expression after it removes leading and trailing blanks.

- **str**: Any valid string expression.

UPPER(str)

Returns a string expression after converting lowercase character data to uppercase.

- **str**: Any valid string expression.

ARRAY_CONCAT(array_exp1, array_exp2 [, array_exp3])

Returns an array that is the result of concatenating two or more array values.

- **array_exp1**: Any valid array expression.
- **array_exp2**: Any valid array expression.
- **array_exp3**: Any valid array expression.

ARRAY_CONTAINS(array_exp, expr [, bool_expr])

Returns a Boolean indicating whether the array contains the specified value. You can check for a partial or full match of an object by using a boolean expression within the command.

- **array_exp1**: Any array expression.
- **expr**: The expression to search for.
- **bool_expr**: If it's set to 'true'and if the specified search value is an object, the command checks for a partial match (the search object is a subset of one of the objects). If it's set to 'false', the command checks for a full match of all objects within the array. The default value if not specified is false.

ARRAY_LENGTH(array_exp)

Returns the number of elements of the specified array expression.

- **array_exp**: Any valid array expression.

ARRAY_SLICE(array_exp, start_index, max_size)

Returns part of an array expression.

- **array_exp**: Any valid array expression.

- **start_index**: Zero-based numeric index at which to begin the array. Negative values may be used to specify the starting index relative to the last element of the array i.e. -1 references the last element in the array.
- **max_size**: Maximum number of elements in the resulting array.

ST_DISTANCE(spatial_expr1, spatial_expr2)

Returns the distance between the two GeoJSON Point, Polygon, or LineString expressions.

- **spatial_expr1**: Is any valid GeoJSON Point, Polygon, or LineString object expression.
- **spatial_expr2**: Is any valid GeoJSON Point, Polygon, or LineString object expression.

ST_WITHIN(spatial_expr1, spatial_expr2)

Returns a Boolean expression indicating whether the GeoJSON object (Point, Polygon, or LineString) specified in the first argument is within the GeoJSON (Point, Polygon, or LineString) in the second argument.

- **spatial_expr1**: Is any valid GeoJSON Point, Polygon, or LineString object expression.
- **spatial_expr2**: Is any valid GeoJSON Point, Polygon, or LineString object expression.

ST_INTERSECTS(1, 2)

Returns a Boolean expression indicating whether the GeoJSON object (Point, Polygon, or LineString) specified in the first argument intersects the GeoJSON (Point, Polygon, or LineString) in the second argument.

- **spatial_expr1**: Is any valid GeoJSON Point, Polygon, or LineString object expression.
- **spatial_expr2**: Is any valid GeoJSON Point, Polygon, or LineString object expression.

ST_ISVALID(spatial_expr)

Returns a Boolean value indicating whether the specified GeoJSON Point, Polygon, or LineString expression is valid.

- **spatial_expr**: Is any valid GeoJSON Point, Polygon, or LineString object expression.

ST_ISVALIDDETAILED(spatial_expr)

Returns a JSON value containing a Boolean value if the specified GeoJSON Point, Polygon, or LineString expression is valid, and if invalid, additionally the reason as a string value.

- spatial_expr: Is any valid GeoJSON Point, Polygon, or LineString object expression.

Predicate Functions

ABS(numeric_expr)

Returns the absolute (positive) value of the specified numeric expression.

- numeric_expr: A numeric expression.

ACOS(numeric_expr)

Returns the angle, in radians, whose cosine is the specified numeric expression; also called arccosine.

- numeric_expr: A numeric expression.

ASIN(numeric_expr)

Returns the angle, in radians, whose sine is the specified numeric expression. This is also called arcsine.

- numeric_expr: A numeric expression.

ATAN(numeric_expr)

Returns the angle, in radians, whose tangent is the specified numeric expression. This is also called arctangent.

- numeric_expr: A numeric expression.

CEILING(numeric_expr)

Returns the smallest integer value greater than, or equal to, the specified numeric expression.

- numeric_expr: A numeric expression.

COS(numeric_expr)

Returns the trigonometric cosine of the specified angle, in radians, in the specified expression.

- numeric_expr: A numeric expression.

COT(numeric_expr)

Returns the trigonometric cotangent of the specified angle, in radians, in the specified numeric expression.

- numeric_expr: A numeric expression.

DEGREES(numeric_expr)

Returns the corresponding angle in degrees for an angle specified in radians.

- numeric_expr: A numeric expression.

FLOOR(numeric_expr)

Returns the largest integer less than or equal to the specified numeric expression.

- numeric_expr: A numeric expression.

EXP(numeric_expr)

Returns the exponential value of the specified numeric expression.

- numeric_expr: A numeric expression.

LOG10(numeric_expr)

Returns the base-10 logarithm of the specified numeric expression.

- numeric_expr: A numeric expression.

RADIANS(numeric_expr)

Returns radians when a numeric expression, in degrees, is entered.

- numeric_expr: A numeric expression.

ROUND(numeric_expr)

Returns a numeric value, rounded to the closest integer value.

- numeric_expr: A numeric expression.

SIGN(numeric_expr)

Returns the positive (+1), zero (0), or negative (-1) sign of the specified numeric expression.

- numeric_expr: A numeric expression.

SIN(numeric_expr)

Returns the trigonometric sine of the specified angle, in radians, in the specified expression.

- numeric_expr: A numeric expression.

SQRT(numeric_expr)

Returns the square root of the specified numeric value.

- numeric_expr: A numeric expression.

SQUARE(numeric_expr)

Returns the square of the specified numeric value.

- numeric_expr: A numeric expression.

TAN(numeric_expr)

Returns the tangent of the specified angle, in radians, in the specified expression.

- numeric_expr: A numeric expression.

TRUNC(numeric_expr)

Returns a numeric value, truncated to the closest integer value.

- numeric_expr: A numeric expression.

ATAN2(y_expr, x_expr)

Returns the principal value of the arc tangent of y/x, expressed in radians.

- y_expr: The y numeric expression.
- x_expr: The x numeric expression.

LOG(numeric_expr [, base])

Returns the natural logarithm of the specified numeric expression.

- **numeric_expr**: A numeric expression.
- **base**: Optional numeric argument that sets the base for the logarithm.

PI()

Returns the constant value of PI.

POWER(numeric_expr, power_expr)

Returns the value of the specified expression to the specified power.

- **numeric_expr**: A numeric expression.
- **power_expr**: Is the power to which to raise numeric_expr.

IS_ARRAY(expr)

Returns a Boolean value indicating if the type of the specified expression is an array.

- **expr**: Any valid expression.

IS_BOOL(expr)

Returns a Boolean value indicating if the type of the specified expression is a Boolean.

- **expr**: Any valid expression.

IS_DEFINED(expr)

Returns a Boolean indicating if the property has been assigned a value.

- **expr**: Any valid expression.

IS_NULL(expr)

Returns a Boolean value indicating if the type of the specified expression is null.

- **expr**: Any valid expression.

IS_NUMBER(expr)

Returns a Boolean value indicating if the type of the specified expression is a number.

- **expr**: Any valid expression.

IS_OBJECT(expr)

Returns a Boolean value indicating if the type of the specified expression is a JSON object.

- **expr**: Any valid expression.

IS_PRIMITIVE(expr)

Returns a Boolean value indicating if the type of the specified expression is a primitive (string, Boolean, numeric, or null).

- **expr**: Any valid expression.

IS_STRING(expr)

Returns a Boolean value indicating if the type of the specified expression is a string.

- **expr**: Any valid expression.

CONCAT(str1, str2 [, str3] [, ...])

Returns a string that is the result of concatenating two or more string values.

- **str1**: The first string to concatenate.
- **str2**: The second string to concatenate.
- **str3**: The third string to concatenate.

CONTAINS(str1, str2)

Returns a Boolean indicating whether the first string expression contains the second.

- **str1**: The string to search in.
- **str2**: The string to search for.

ENDSWITH(str1, str2)

Returns a Boolean indicating whether the first string expression ends with the second.

- **str1**: The string to search in.
- **str2**: The string to search for.

INDEX_OF(str1, str2)

Returns the starting position of the first occurrence of the second string expression within the first specified string expression, or -1 if the string is not found.

- **str1**: The string to search in.
- **str2**: The string to search for.

LEFT(str, num_expr)

Returns the left part of a string with the specified number of characters.

- **str**: A valid string expression.
- **num_expr**: The number of characters to return.

LENGTH(str)

Returns the number of characters of the specified string expression.

- **str**: Any valid string expression.

LOWER(str)

Returns a string expression after converting uppercase character data to lowercase.

- **str**: Any valid string expression.

LTRIM(str)

Returns a string expression after it removes leading blanks.

- **str**: Any valid string expression.

REPLACE(original_value, from_value, to_value)

Replaces all occurrences of a specified string value with another string value.

- **original_value**: The string to search in.
- **from_value**: The string to search for.
- **to_value**: The string to replace instances of from_value.

REPLICATE(str, repeat_num)

Repeats a string value a specified number of times.

- **str**: The string expression to repeat.
- **repeat_num**: The number of times to repeat the str expression.

REVERSE(str)

Returns the reverse order of a string value.

- **str**: Any valid string expression.

RIGHT(str, num_expr)

Returns the right part of a string with the specified number of characters.

- **str**: Any valid string expression.
- **num_expr**: The starting index.

RTRIM(str)

Returns a string expression after it removes trailing blanks.

- **str**: Any valid string expression.

STARTSWITH(str1, str2)

Returns a Boolean indicating whether the first string expression starts with the second.

- **str1**: The string to search in.
- **str2**: The string to search for.

SUBSTRING(str, start_index, length)

Returns part of a string expression starting at the specified character zero-based position and continues to the specified length, or to the end of the string.

- **str**: Any valid string expression.
- **start_index**: The starting index.
- **length**: The length of the string to return.

TOSTRING(expr)

Returns a string representation of scalar expression.

- **expr**: Any valid expression.

TRIM(str)

Returns a string expression after it removes leading and trailing blanks.

- **str**: Any valid string expression.

UPPER(str)

Returns a string expression after converting lowercase character data to uppercase.

- **str**: Any valid string expression.

ARRAY_CONCAT(array_exp1, array_exp2 [, array_exp3])

Returns an array that is the result of concatenating two or more array values.

- **array_exp1**: Any valid array expression.
- **array_exp2**: Any valid array expression.
- **array_exp3**: Any valid array expression.

ARRAY_CONTAINS(array_exp, expr [, bool_expr])

Returns a Boolean indicating whether the array contains the specified value. You can check for a partial or full match of an object by using a boolean expression within the command.

- **array_exp1**: Any array expression.
- **expr**: The expression to search for.
- **bool_expr**: If it's set to 'true'and if the specified search value is an object, the command checks for a partial match (the search object is a subset of one of the objects). If it's set to 'false', the command checks for a full match of all objects within the array. The default value if not specified is false.

ARRAY_LENGTH(array_exp)

Returns the number of elements of the specified array expression.

- **array_exp**: Any valid array expression.

ARRAY_SLICE(array_exp, start_index, max_size)

Returns part of an array expression.

- **array_exp**: Any valid array expression.

- **start_index**: Zero-based numeric index at which to begin the array. Negative values may be used to specify the starting index relative to the last element of the array i.e. -1 references the last element in the array.
- **max_size**: Maximum number of elements in the resulting array.

ST_DISTANCE(spatial_expr1, spatial_expr2)

Returns the distance between the two GeoJSON Point, Polygon, or LineString expressions.

- **spatial_expr1**: Is any valid GeoJSON Point, Polygon, or LineString object expression.
- **spatial_expr2**: Is any valid GeoJSON Point, Polygon, or LineString object expression.

ST_WITHIN(spatial_expr1, spatial_expr2)

Returns a Boolean expression indicating whether the GeoJSON object (Point, Polygon, or LineString) specified in the first argument is within the GeoJSON (Point, Polygon, or LineString) in the second argument.

- **spatial_expr1**: Is any valid GeoJSON Point, Polygon, or LineString object expression.
- **spatial_expr2**: Is any valid GeoJSON Point, Polygon, or LineString object expression.

ST_INTERSECTS(1, 2)

Returns a Boolean expression indicating whether the GeoJSON object (Point, Polygon, or LineString) specified in the first argument intersects the GeoJSON (Point, Polygon, or LineString) in the second argument.

- **spatial_expr1**: Is any valid GeoJSON Point, Polygon, or LineString object expression.
- **spatial_expr2**: Is any valid GeoJSON Point, Polygon, or LineString object expression.

ST_ISVALID(spatial_expr)

Returns a Boolean value indicating whether the specified GeoJSON Point, Polygon, or LineString expression is valid.

- **spatial_expr**: Is any valid GeoJSON Point, Polygon, or LineString object expression.

ST_ISVALIDDETAILED(spatial_expr)

Returns a JSON value containing a Boolean value if the specified GeoJSON Point, Polygon, or LineString expression is valid, and if invalid, additionally the reason as a string value.

- spatial_expr: Is any valid GeoJSON Point, Polygon, or LineString object expression.

SELECT INTO Statements

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

Data Export with an SQL Query

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

```
boolean ret = stat.execute("SELECT City, CompanyName INTO
"csv://c:/Customers.txt" FROM "Customers" WHERE Country =
'US'");
System.out.println(stat.getUpdateCount()+" rows affected");
```

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

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

INSERT Statements

To create new records, use INSERT statements.

INSERT Syntax

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

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

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

You can use the executeUpdate method of the Statement and PreparedStatement classes to execute data manipulation commands and retrieve the rows affected.

```
String cmd = "INSERT INTO Customers (CompanyName) VALUES
(?)";
PreparedStatement pstmt = connection.prepareStatement(cmd);
pstmt.setString(1, "Caterpillar");
int count = pstmt.executeUpdate();
System.out.println(count+" rows were affected");
connection.close();
```

UPDATE Statements

To modify existing records, use UPDATE statements.

Update Syntax

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

```
UPDATE <table_name> SET { <column_reference> = <expression>
} [ , ... ] WHERE { _id = <expression>  } [ { AND | OR } ...
]

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

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

```
String cmd = "UPDATE Customers SET CompanyName='Caterpillar'
WHERE _id = ?";
PreparedStatement pstmt = connection.prepareStatement(cmd);
pstmt.setString(1, "22");
```

```
int count = pstmt.executeUpdate();
System.out.println(count + " rows were affected");
connection.close();
```

DELETE Statements

To delete information from a table, use DELETE statements.

DELETE Syntax

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

```
<delete_statement> ::= DELETE FROM <table_name> WHERE { _id
= <expression> } [ { AND | OR } ... ]

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

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

```
Connection connection =
DriverManager.getConnection("jdbc:cosmosdb:AccountEndpoint=m
yAccountEndpoint;AccountKey=myAccountKey;",);
String cmd = "DELETE FROM Customers WHERE _id = ?";
PreparedStatement pstmt = connection.prepareStatement(cmd);
pstmt.setString(1, "22");
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;
```

TIBCO Product Documentation and Support Services

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

How to Access TIBCO Documentation

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

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

Product-Specific Documentation

The following documentation for this product is available on the TIBCO Data Virtualization page.

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

- **Other**

  TDV Business Directory Guide

  TDV Discovery Guide

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

How to Contact TIBCO Support

Get an overview of TIBCO Support. You can contact TIBCO Support in the following ways:

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

How to Join TIBCO Community

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

Legal and Third-Party Notices

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

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

TIBCO, TIBCO logo, 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.

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

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

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

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

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

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

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