



TIBCO® Data Virtualization

Couchbase Adapter Guide

Version 8.7.0 | October 2023

Contents

Contents	2
Couchbase Adapter	4
Getting Started	4
Basic Tab	5
Logging	7
NoSQL Database	8
Automatic Schema Discovery	9
Query Mapping	14
Vertical Flattening	23
User-Defined Functions	24
JSON Functions	26
Custom Schema Definitions	29
Custom Schema Example	32
Changelog	33
Advanced Features	42
User Defined Views	43
SSL Configuration	45
Firewall and Proxy	46
Query Processing	46
Logging	47
SQL Compliance	50
SELECT Statements	52
SELECT INTO Statements	95
INSERT Statements	96
UPDATE Statements	97
DELETE Statements	97
EXECUTE Statements	98

PIVOT and UNPIVOT	99
Data Model	100
Stored Procedures	101
Connection String Options	116
Authentication	122
SSL	130
Firewall	136
Proxy	140
Logging	146
Schema	147
Miscellaneous	161
TIBCO Product Documentation and Support Services	182
How to Access TIBCO Documentation	182
How to Contact TIBCO Support	183
Release Version Support	183
How to Join TIBCO Community	184
Legal and Third-Party Notices	185

Couchbase Adapter

Couchbase Version Support

The Couchbase Adapter models Couchbase documents in a bucket as tables in a relational database; connect to Couchbase Server versions 4.0 and up, Enterprise Edition or Community Edition.

SQL Compliance

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

Getting Started

Connecting to Couchbase

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

Deploying the Couchbase Adapter

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

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

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

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

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

Basic Tab

Connecting to Couchbase

To connect to data, set the Server property to the hostname or IP address of the Couchbase server(s) you are authenticating to.

If your Couchbase server is configured to use SSL, you can enable it either by using an https URL for Server (like `https://couchbase.server`), or by setting the UseSSL property to **True**.

Couchbase Analytics

By default, the adapter connects to the N1QL Query service. In order to connect to the Couchbase Analytics service, you will also need to set the CouchbaseService property to **Analytics**.

Couchbase Cloud

Set the following to connect to Couchbase Cloud:

- AuthScheme: Set this to **Basic**.
- ConnectionMode: Set this to **Cloud**.
- DNSServer: Set this to a DNS server. In most cases, this should be a public DNS service like **1.1.1.1** or **8.8.8.8**.
- SSLServerCert: Set this to the TLS/SSL certificate to be accepted from the server. Any other certificate that is not trusted by the machine is rejected. Alternatively, set "*" to accept all certificates.

Authenticating to Couchbase

The adapter supports several forms of authentication. Couchbase Cloud only accepts Standard authentication, while Couchbase Server accepts Standard authentication, client certificates, and credentials files.

Standard Authentication

To authenticate with standard authentication, set the following:

- AuthScheme: Set this to **Basic**.
- User: The user authenticating to Couchbase.
- Password: The password of the user authenticating to Couchbase.

Client Certificates

The adapter supports authenticating with client certificates when SSL is enabled. To use client certificate authentication, set the following properties:

- AuthScheme: Set this to **SSLCertificate**.
- SSLClientCertType: The type of client certificate set within SSLClientCert.
- SSLClientCert: The client certificate in the format given by SSLClientCertType.
- SSLClientCertPassword (optional): The password of the client certificate, if it is encrypted.
- SSLClientCertSubject (optional): The subject of the client certificate, which, by default, is the first certificate found in the store. This is required if more than one certificate is available in the certificate store.

Credentials File

You can also authenticate using using a credentials file containing multiple logins. This is included for legacy use and is not recommended when connecting to a Couchbase Server that supports role-based authentication.

- AuthScheme: Set this to **CredentialsFile**.
- CredentialsFile: The path to the credentials file. Refer to [Couchbase's documentation](#) for more information on the format of this file.

Logging

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

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

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

```
Verbosity=4;
```

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

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

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

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

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

- 1 - Will log the query, the number of rows returned by it, the start of execution and the time taken, and any errors.
- 2 - Will log everything included in Verbosity 1 and HTTP headers.
- 3 - Will additionally log the body of the HTTP requests.

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

Configure Logging for the Couchbase Adapter

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

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

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

Here is an example of the calls:

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

All logs for the adapter are written to the "cs_server_dsrc.log" file as specified in the log4j properties.

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

NoSQL Database

Couchbase is a schema-free document database that provides high performance, availability, and scalability. These features are not necessarily incompatible with a standards-compliant query language like SQL-92.

The adapter models the schema-free Couchbase objects into relational tables and translates SQL queries into N1QL or SQL++ (Analytics) queries to get the requested data. In this section we will show various schemes that the adapter offers to bridge the gap with relational SQL and a document database.

Automatic Schema Discovery

When the adapter first connects to Couchbase, it opens each bucket and scans a configurable number of rows from that bucket. It uses those rows to determine the columns in that bucket and their data types, as well as how to build flavored and child tables for any arrays within those documents. For Couchbase Enterprise version 4.5.1 and later, the adapter may also be configured to use the INFER command when TypeDetectionScheme is set to INFER. This allows the adapter to get a more accurate column listing for the bucket, and to detect more complex flavors.

When using the Analytics service, the adapter only does column and child table detection. Flavored tables are provided by Couchbase itself using shadow datasets. Also, Analytics mode does not currently have INFER support, so only row scan is supported.

For more details, refer to [Automatic Schema Discovery](#) to see how flavored tables and child tables are modelled from Couchbase data. Setting NumericStrings is also recommended as it can avoid type detection issues with certain kinds of text data.

Custom Schema Definitions

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

Query Mapping

See [Query Mapping](#) for more details on how various N1QL and SQL++ operations are represented as SQL.

Vertical Flattening

See [Vertical Flattening](#) for more details on how arrays and objects are mapped into fields.

JSON Functions

See [JSON Functions](#) for more details on how to extract data from raw JSON strings.

Automatic Schema Discovery

Child Tables

If the documents within a bucket contain fields with arrays, then the adapter will expose those fields as their own tables in addition to exposing them as JSON aggregates on the main table. The structure of these child tables depends upon whether the array contains objects or primitive values.

Array Child Tables

If the arrays contain primitive values like numbers or strings, the child table will have only two columns: one called "Document.Id" which is the primary key of the document containing the array, and one called "value" which contains the value within the array. For example, if the bucket "Games" contains these documents:

```
/* Primary key "1" */
{
  "scores": [1,2,3]
}
/* Primary key "2" */
{
  "scores": [4,5,6]
}
```

The adapter will build a table called "Games_scores" containing these rows:

Document.Id	value
1	1
1	2
1	3
2	4
2	5
2	6

Object Child Tables

If the arrays contain objects, the child table will have a column for each field that occurs within the objects, as well as a "Document.Id" column which contains the primary key of the document containing the array. For example, if the bucket "Games" contains these documents:

```
/* Primary key "1" */
{
  "moves": [
    {"piece": "pawn", "square": "c3"},
    {"piece": "rook", "square": "d5"}
  ]
}
/* Primary key "2" */
{
  "moves": [
    {"piece": "knight", "square": "f1"},
    {"piece": "bishop", "square": "e4"}
  ]
}
```

The adapter will build a table called "Games_moves" containing these rows:

Document.Id	piece	square
1	pawn	c3
1	rook	d5
2	knight	f1
2	bishop	e4

NewChildJoinsMode

Note that the above data model is not fully relational, which has important limitations for use-cases that involve complex JOINS or DML operations on child tables. The NewChildJoinsMode connection property exposes an alternative data model which avoids

these limitations. Please refer to its page in the connection property section of the documentation for more details.

Flavored Tables

The adapter can also detect when there are multiple types of documents within the same bucket, as long as TypeDetectionScheme is set to Infer or DocType and CouchbaseService is set to N1QL. These different types of documents are exposed as their own tables containing only the appropriate rows.

For example, the bucket "Games" contains documents which have a "type" value of either "chess" or "football":

```
/* Primary key "1" */
{
  "type": "chess",
  "result": "stalemate"
}
/* Primary key "2" */
{
  "type": "chess",
  "result": "black win"
}
/* Primary key "3" */
{
  "type": "football",
  "score": 23
}
/* Primary key "4" */
{
  "type": "football",
  "score": 18
}
```

The adapter will create three tables for this bucket: one called "Games" which contains all the documents:

Document.Id	result	score	type
1	stalemate	NULL	chess
2	black win	NULL	chess

3	NULL	23	football
4	NULL	18	football

One called "Games.chess" which contains only documents where the type is "chess":

Document.Id	result	type
1	stalemate	chess
2	black win	chess

And one called "Games.football" which contains only documents where the type is "football":

Document.Id	score	type
3	23	football
4	18	football

Note that the adapter will not include columns in a flavored table that are not defined on the documents in that flavor. For example, even though both the "result" and "score" columns are included on the base table, "Games.chess" only includes "result" and "Games.football" only includes "score".

Flavored Child Tables

It is also possible for a flavored table to contain arrays, which will become their own child tables. For example, if the bucket "Games" contains these documents:

```
/* Primary key "1" */
{
  "type": "chess",
  "results": ["stalemate", "white win"]
}
```

```

}
/* Primary key "2" */
{
  "type": "chess",
  "results": ["black win", "stalemate"]
}
/* Primary key "3" */
{
  "type": "football",
  "scores": [23, 12]
}
/* Primary key "4" */
{
  "type": "football",
  "scores": [18, 36]
}

```

Then the adapter will generate these tables:

Table Name	Child Field	Flavor Condition
Games		
Games_results	results	
Games_scores	scores	
Games.chess		"type" = "chess"
Games.chess_results	results	"type" = "chess"
Games.football		"type" = "football"
Games.football_scores	scores	"type" = "football"

Query Mapping

The adapter maps SQL-92-compliant queries into corresponding N1QL or SQL++ queries. Although the mapping below is not complete, it should help you get a sense for the common patterns the adapter uses during this transformation.

SELECT Queries

The SELECT statements are translated to the appropriate N1QL SELECT query as shown below. Due to the similarities between SQL-92 and N1QL, many queries will simply be direct translations.

One major difference is that when the schema for a given Couchbase bucket exists in the adapter, a SELECT * query will be translated to directly select the individual fields in the bucket. The adapter will also automatically create a **Document.Id** column based on the primary key of each document in the bucket.

SQL Query	N1QL Query
SELECT * FROM users	SELECT META (`users`).id AS `id`, ... FROM `users`
SELECT [Document.Id], status FROM users	SELECT META (`users`).id AS `Document.Id`, `users`.`status` FROM `users`
SELECT * FROM users WHERE status = 'A' OR age = 50	SELECT META (`users`).id AS `id`, ... FROM `users` WHERE TOSTRING (`users`.`status`) = "A" OR TONUMBER (`users`.`age`) = 50
SELECT * FROM users WHERE name LIKE 'A%'	SELECT META (`users`).id AS `id`, ... FROM `users` WHERE TOSTRING (`users`.`name`) LIKE "A%"
SELECT * FROM users WHERE status = 'A' ORDER BY [Document.Id] DESC	SELECT META (`users`).id AS `id`, ...

	FROM `users` WHERE TOSTRING (`users`.`status`) = "A" ORDER BY META (`users`).id DESC
SELECT * FROM users WHERE status IN ('A', 'B')	SELECT META (`users`).id, ... FROM `users` WHERE TOSTRING (`users`.`status`) IN ["A", "B"]

Note that conditions can include extra type functions if the adapter detects that a type conversion may be necessary. You can disable these type conversions using the [StrictComparison](#) property. For clarity, the rest of the N1QL samples are shown without these extra conversion functions.

USE KEYS Optimizations

When a query has either equals or IN clause that targets the **Document.Id** column, and there is no OR clause to override it, the adapter will convert the **Document.Id** filter into a USE KEYS clause. This avoids the overhead of scanning an index because the document keys are already known to the N1QL engine (this optimization does not apply to the Analytics [CouchbaseService](#)).

SQL Query	N1QL Query
SELECT * FROM users WHERE [Document.Id] = '1'	SELECT ... FROM `users` USE KEYS ["1"]
SELECT * FROM users WHERE [Document.Id] IN ('2', '3')	SELECT ... FROM `users` USE KEYS ["2", "3"]
SELECT * FROM users WHERE [Document.Id] = '4' OR [Document.Id] = '5'	SELECT ... FROM `users` USE KEYS

	["4", "5"]
SELECT * FROM users WHERE [Document.Id] = '6' AND status = 'A'	SELECT ... FROM `users` USE KEYS ["6"] WHERE `status` = "A"

In addition to being used for SELECT queries, the same optimization is performed for DML operations as shown below.

Child Tables

As long as all the child tables in a query share the same parent, and they are combined using INNER JOINS on their **Document.Id** columns, the adapter will combine the JOINS into a single UNNEST expression. Unlike N1QL UNNEST queries, you must explicitly JOIN with the base table if you want to access its fields.

SQL Query	N1QL Query
SELECT * FROM users_posts	SELECT META(`users`).id, `users_posts`.`text`, ... FROM `users` UNNEST `users`.`posts` AS `users_ posts`
SELECT * FROM users INNER JOIN users_posts ON users.[Document.Id] = users_posts.[Document.Id]	SELECT META(`users`).id, `users`.`name`, ..., `users_ posts`.`text`, ... FROM `users` UNNEST `users`.`posts` AS `users_ posts`
SELECT * FROM users INNER JOIN users_posts ... INNER JOIN users_comments ON ...	SELECT ... FROM `users` UNNEST `users`.`posts` AS `users_posts` UNNEST `users`.`comments` AS `users_comments`

Flavor Tables

Flavored tables always have the appropriate condition included when you query, so that only documents from the flavor will be returned:

SQL Query	N1QL Query
SELECT * FROM [users.subscriber]	SELECT ... FROM `users` WHERE `docType` = "subscriber"
SELECT * FROM [users.subscriber] WHERE age > 50	SELECT ... FROM `users` WHERE `docType` = "subscriber" AND `age` > 50

Aggregate Queries

N1QL has several built-in aggregate functions. The adapter makes extensive use of this for various aggregate queries. See some examples below:

SQL Query	N1QL Query
SELECT Count(*) As Count FROM Orders	SELECT Count(*) AS `count` FROM `Orders`
SELECT Sum(price) As total FROM Orders	SELECT Sum(`price`) As `total` FROM `Orders`
SELECT cust_id, Sum(price) As total FROM Orders GROUP BY cust_id ORDER BY total	SELECT `cust_id`, Sum(`price`) As `total` FROM `Orders` GROUP BY `cust_id` ORDER BY `total`
SELECT cust_id, ord_date, Sum(price) As total FROM Orders GROUP	SELECT `cust_id`,

BY cust_id, ord_date Having total > 250

`ord_date`, Sum
(`price`) As `total`
FROM `Orders`
GROUP BY `cust_id`,
`ord_date` Having
`total` > 250

Insert Statements

The SQL INSERT statement is mapped to the N1QL INSERT statement as shown below. This works the same for both top-level fields as well as fields produced by [Vertical Flattening](#):

SQL Query	N1QL Query
INSERT INTO users([Document.Id], age, status) VALUES ('bcd001', 45, 'A')	INSERT INTO `users` (KEY, VALUE) VALUES ('bcd001', { "age" : 45, "status" : "A" })
INSERT INTO users([Document.Id], [metrics.posts]) VALUES ('bcd002', 0)	INSERT INTO `users` (KEY, VALUE) VALUES ('bcd002', {"metrics": {"posts": 0}})

Child Table Inserts

Inserts on child tables are converted internally into N1QL UPDATEs using array operations. Since that this does not create the top-level document, the Document.Id provided must refer to a document that already exists.

Another limitation of child table inserts is that multi-valued inserts must all use the same Document.Id. The provider will verify this before modifying any data and raise an error if this constraint is violated.

SQL Query	N1QL Query

INSERT INTO users_ratings([Document.Id], value) VALUES ('bcd001', 4.8), ('bcd001', 3.2)	UPDATE `users` USE KEYS "bcd001" SET `ratings` = ARRAY_PUT(`ratings`, 4.8, 3.2)
INSERT INTO users_reviews([Document.Id], score) VALUES ('bcd002', 'Great'), ('bcd002', 'Lacking')	UPDATE `users` USE KEYS "bcd001" SET `ratings` = ARRAY_PUT(`ratings`, {"score": "Great"}, {"score": "Lacking"})

Bulk Insert Statements

Bulk inserts are also supported the SQL Bulk Insert is converted as shown below:

```
INSERT INTO users#Temp([Document.Id], KEY, VALUE) VALUES('bcd001', 45, "A")
INSERT INTO users#Temp([Document.Id], KEY, VALUE) VALUES('bcd002', 24, "B")
INSERT INTO users SELECT * FROM users#Temp
```

is converted to:

```
INSERT INTO `users` (KEY, VALUE) VALUES
('bcd001', {"age": 45, "status": "A"}),
('bcd002', {"age": 24, "status": "B"})
```

Like multi-valued inserts on child tables, all the rows in a bulk insert must also have the same Document.Id.

Update Statements

The SQL UPDATE statement is mapped to the N1SQL UPDATE statement as shown below:

SQL Query	N1QL Query
UPDATE users SET status = 'C' WHERE [Document.Id] = 'bcd001'	UPDATE `users` USE KEYS ["bcd001"] SET `status` = "C"

```
UPDATE users SET status = 'C' WHERE age > 45
```

```
UPDATE `users` SET
`status` = "C" WHERE
`age` > 45
```

Child Table Updates

When updating a child table, the SQL query is converted to an UPDATE query using either a "FOR" expression or an "ARRAY" expression:

SQL Query	N1QL Query
UPDATE users_ratings SET value = 5.0 WHERE value > 5.0	UPDATE `users` SET `ratings` = ARRAY CASE WHEN `value` > 5.0 THEN 5 ELSE `value` END FOR `value` IN `ratings` END
UPDATE users_reviews SET score = 'Unknown' WHERE score = ''	UPDATE `users` SET `\$child`.`score` = 'Unknown' FOR `\$child` IN `reviews` WHEN `\$child`.`score` = "" END

Flavor Table Updates

Like flavor table SELECTs, UPDATEs on flavor tables always include the appropriate condition, so only documents belonging to the flavor are affected:

SQL Query	N1QL Query
UPDATE [users.subscriber] SET status = 'C' WHERE age > 45	UPDATE `users` SET `status` = "C" WHERE `docType` = "subscriber" AND `age` > 45

Delete Statements

The SQL DELETE statement is mapped to the N1QL DELETE statement as shown below:

SQL Query	N1QL Query
DELETE FROM users WHERE [Document.Id] = 'bcd001'	DELETE FROM `users` USE KEYS ["bcd001"]
DELETE FROM users WHERE status = 'inactive'	DELETE FROM `users` WHERE `status` = "inactive"

Child Table Deletes

When deleting from a child table, the SQL query is converted to an UPDATE query using an "ARRAY" expression:

SQL Query	N1QL Query
DELETE FROM users_ratings WHERE value < 0	UPDATE `users` SET `ratings` = ARRAY `value` FOR `value` IN `ratings` WHEN NOT (`value` < 0) END
DELETE FROM users_reviews WHERE score = ""	UPDATE `users` SET `reviews` = ARRAY `\$child` FOR `\$child` IN `reviews` WHEN NOT (`\$child`.`score` = "") END

Flavor Tables Deletes

Like flavor table SELECTs, DELETEs on flavor tables always include the appropriate condition, so only documents belonging to the flavor are affected:

SQL Query	N1QL Query
DELETE FROM [users.subscriber] WHERE status = 'inactive'	DELETE FROM `users` WHERE

```
`docType` = "subscriber" AND  
status = "inactive"
```

Vertical Flattening

Example Document

```
/* Primary key "1" */  
{  
  "address" : {  
    "building" : "1007",  
    "coord" : [-73.856077, 40.848447],  
    "street" : "Morris Park Ave",  
    "zipcode" : "10462"  
  },  
  "borough" : "Bronx",  
  "cuisine" : "Bakery",  
  "grades" : [{  
    "date" : "2014-03-03T00:00:00Z",  
    "grade" : "A",  
    "score" : 2  
  }, {  
    "date" : "2013-09-11T00:00:00Z",  
    "grade" : "A",  
    "score" : 6  
  }, {  
    "date" : "2013-01-24T00:00:00Z",  
    "grade" : "A",  
    "score" : 10  
  }, {  
    "date" : "2011-11-23T00:00:00Z",  
    "grade" : "A",  
    "score" : 9  
  }, {  
    "date" : "2011-03-10T00:00:00Z",  
    "grade" : "B",  
    "score" : 14  
  }],  
  "name" : "Morris Park Bake Shop",  
  "restaurant_id" : "30075445"  
}
```

Selecting Values In Objects

If the `FlattenObjects` property is configured to allow object flattening, then the adapter will traverse objects and map the fields inside them as columns. For example, this query:

```
SELECT [address.building], [address.street] FROM restaurants
```

Would return this resultset:

address.building	address.street
1007	Morris Park Ave

Selecting Values In Arrays

If the `FlattenArrays` property is configured to allow array flattening, then the adapter will traverse arrays and map their individual values as columns. For example, if Flatten Arrays were set to "2", then this query:

```
SELECT [address.coord.0], [address.coord.1] FROM restaurants
```

Would return this resultset:

address.coord.0	address.coord.1
-73.856077	40.838447

Note that array flattening should only be used in cases where you know the number of array items in advance, such as with "address.coord" which will always contain two items. For arrays like "grades" which can contain arbitrary numbers of items, consider using the child tables described in [Automatic Schema Discovery](#) instead, since they will allow you to read all of the values within the array.

User-Defined Functions

User-defined functions are a new feature provided by Couchbase 7 and up. They can be used with the adapter like normal functions but with a special naming convention for using scoped functions. Normally the adapter requires that functions already exist before they

are used, to define them refer to the Couchbase documentation on **CREATE FUNCTION** queries. These may be run at the Couchbase console or with the adapter in QueryPassthrough mode.

Couchbase has support for both scalar functions as well as functions that return results from subqueries. The adapter supports scalar functions within its SQL dialect but subquery functions can only be used when QueryPassthrough is enabled. The rest of this section covers the adapter's SQL dialect and assumes that QueryPassthrough is disabled.

Global Functions

In both N1QL and Analytics mode, global user-defined functions can be accessed using either their simple names or their qualified names. The simple name is just the name of the function:

```
SELECT ageInYears(birthdate) FROM users
```

Global functions may also be invoked by qualifying them with the default namespace. Qualified names are quoted names that contain internal separators, which by default is a period though this can be changed using the DataverseSeparator property. In both N1QL and Analytics the global namespace is called **Default**:

```
SELECT [Default.ageInYears](birthdate) FROM users
```

Calling global functions using simple names is recommended. While the default qualifier is supported, its only intended use is for when a UDF clashes with a standard SQL function that the adapter would otherwise translate.

Scoped Functions

Both N1QL and Analytics also allow functions to be defined outside of a global context. In Analytics functions can be attached to both dataverses and scopes which are called using two-part and three-part names respectively. In N1QL functions may only be attached to scopes so only three-part names may be used.

```
/* N1QL AND Analytics */
SELECT [socialNetwork.accounts.ageInYears](birthdate) FROM
[socialNetwork.accounts.users]
/* Analytics only */
```

```
SELECT [socialNetwork.ageInYears](birthdate) FROM
[socialNetwork.accounts.users]
```

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 return an document as a JSON string. DOCUMENT (*) can be used with any type of SELECT query, including queries including other columns, queries including just DOCUMENT(*), and even more complex queries like JOINS.

```
SELECT [Document.Id], grade, score, DOCUMENT(*) FROM grades
```

For example, that query would return:

Document.Id	grade	score	DOCUMENT
1	A	6	{"document.id":1,"grade":"A","score":6}
2	A	10	{"document.id":1,"grade":"A","score":10}
3	A	9	{"document.id":1,"grade":"A","score":9}
4	B	14	{"document.id":1,"grade":"B","score":14}

When used alone, DOCUMENT(*) returns the structure directly from Couchbase as if a N1QL or SQL++ SELECT * query were used. This means that no Document.Id value will be present since Couchbase does not include it automatically.

```
SELECT DOCUMENT(*) FROM grades
```

This query would return:

DOCUMENT

```
{"grades":{"grade":"A","score":6}}
```

```
{"grades":{"grade":"A","score":10}}
```

```
{"grades":{"grade":"A","score":9}}
```

```
{"grades":{"grade":"B","score":14}}
```

Custom Schema Definitions

In addition to [Automatic Schema Discovery](#) the adapter also allows you to statically define the schema for your Couchbase object. Schemas are defined in text-based configuration files, which makes them easy to extend. You can call the [CreateSchema](#) stored procedure to generate a schema file; see [Automatic Schema Discovery](#) for more information.

Set the [Location](#) property to the file directory that will contain the schema file. The following sections show how to extend the resulting schema or write your own.

Example Document

Let's consider the document below and extract out the nested properties as their own columns:

```
/* Primary key "1" */
{
  "id": 12,
  "name": "Lohia Manufacturers Inc.",
  "homeaddress": {"street": "Main Street", "city": "Chapel Hill",
"state": "NC"},
  "workaddress": {"street": "10th Street", "city": "Chapel Hill",
"state": "NC"}
  "offices": ["Chapel Hill", "London", "New York"]
  "annual_revenue": 35600000
}
/* Primary key "2" */
{
  "id": 15,
  "name": "Piago Industries",
  "homeaddress": {"street": "Main Street", "city": "San Francisco",
```

```

"state": "CA"},
  "workaddress": {street": "10th Street", "city": "San Francisco",
"state": "CA"}
  "offices": ["Durham", "San Francisco"]
  "annual_revenue": 42600000
}

```

Custom Schema Definition

```

<rsb:info title="Customers" description="Customers" other:dataverse=""
other:bucket=customers"" other:flavorexpr="" other:flavorvalue=""
other:isarray="false" other:pathspec="" other:childpath="">
  <attr name="document.id"          xs:type="string" key="true"
other:iskey="true" other:pathspec="" />
  <attr name="annual_revenue"       xs:type="integer" other:iskey="false"
other:pathspec="" other:field="annual_revenue" />
  <attr name="homeaddress.city"     xs:type="string" other:iskey="false"
other:pathspec="{\" other:field="homeaddress.city" />
  <attr name="homeaddress.state"    xs:type="string" other:iskey="false"
other:pathspec="{\" other:field="homeaddress.state" />
  <attr name="homeaddress.street"   xs:type="string" other:iskey="false"
other:pathspec="{\" other:field="homeaddress.street" />
  <attr name="name"                 xs:type="string" other:iskey="false"
other:pathspec="" other:field="name" />
  <attr name="id"                   xs:type="integer" other:iskey="false"
other:pathspec="" other:field="id" />
  <attr name="offices"              xs:type="string" other:iskey="false"
other:pathspec="" other:field="offices" />
  <attr name="offices.0"            xs:type="string" other:iskey="false"
other:pathspec="[" other:field="offices.0" />
  <attr name="offices.1"            xs:type="string" other:iskey="false"
other:pathspec="[" other:field="offices.1" />
  <attr name="workaddress.city"     xs:type="string" other:iskey="false"
other:pathspec="{\" other:field="workaddress.city" />
  <attr name="workaddress.state"    xs:type="string" other:iskey="false"
other:pathspec="{\" other:field="workaddress.state" />
  <attr name="workaddress.street"   xs:type="string" other:iskey="false"
other:pathspec="{\" other:field="workaddress.street" />
</rsb:info>

```

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

Table Properties

The schema above uses the following properties to define specific qualities for the whole table. All of them are required:

Property	Meaning
other:dataverse	The name of the dataverse the dataset belongs to. Empty if not an Analytics view.
other:bucket	The name of the bucket or dataset within Couchbase
other:flavorexpr	The URL encoded condition in a flavored table. For example, "%60docType%60%20%3D%20%22chess%22".
other:flavorvalue	The name of the flavor in a flavored table. For example, "chess".
other:isarray	Whether the table is an array child table.
other:pathspec	This is used to interpret the separators within other:childpath. See Column Properties for more details.
other:childpath	The path to the attribute that is used to UNNEST the child table. Empty if not a child table.

Column Properties

The schema above uses the following properties to define specific qualities for each column:

Property	Meaning
name	Required. The name of the column, lower-cased.
key	Used to mark the primary key. Required for Document.Id but optional for other columns.
xs:type	Required. The type of the column within the adapter.

other:iskey	Required. Must be the same value as key, or "false" if key is not included.
other:pathspec	Required. This is used to interpret the separators within other:field.
other:field	Required. The path to the field in Couchbase.

Note that the fields which are produced by vertical flattening use the same syntax for separating array values and field values. This introduces a potential ambiguity in cases like the following, where the adapter exposes the columns "numeric_object.0" and "array.0":

```
{
  "numeric_object": {
    "0": 0
  },
  "array": [
    0
  ]
}
```

To ensure that the adapter can distinguish between field and array accesses, the pathspec is used to determine whether each "." in the field is an array or an object. Each "{" represents a field access, while each "[" represents an array access.

For example, with a field of "a.0.b.1" and a "pathspec" of "[{[", the N1QL expression "a[0].b[1]" would be generated. If instead the "pathspec" were "{{}", then the N1QL expression "a.`0`.b.`1`" would be generated.

Custom Schema Example

This section contains a complete schema. Set the [Location](#) property to the file directory that will contain the schema file. The info section enables a relational view of a Couchbase 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. The operations, such as couchbaseadoSysData, are internal implementations.

```
<rsb:script xmlns:rsb="http://www.rssbus.com/ns/rsbscript/2">
  <rsb:info title="Customers" description="Customers" other:dataverse=""
other:bucket=customers"" other:flavorexpr="" other:flavorvalue=""
other:isarray="false" other:pathspec="" other:childpath="">
    <attr name="document.id"          xs:type="string" key="true"
other:iskey="true" other:pathspec=""  />
  </rsb:info>
</rsb:script>
```



```

    <attr name="annual_revenue"      xs:type="integer"
other:iskey="false"                other:pathspec="" other:field="annual_
revenue" />
    <attr name="homeaddress.city"   xs:type="string"
other:iskey="false"                other:pathspec="{
other:field="homeaddress.city" />
    <attr name="homeaddress.state"  xs:type="string"
other:iskey="false"                other:pathspec="{
other:field="homeaddress.state" />
    <attr name="homeaddress.street" xs:type="string"
other:iskey="false"                other:pathspec="{
other:field="homeaddress.street" />
    <attr name="name"               xs:type="string"
other:iskey="false"                other:pathspec="" other:field="name" />
    <attr name="id"                 xs:type="integer"
other:iskey="false"                other:pathspec="" other:field="id" />
    <attr name="offices"            xs:type="string"
other:iskey="false"                other:pathspec="" other:field="offices"
/>
    <attr name="offices.0"          xs:type="string"
other:iskey="false"                other:pathspec="[" other:field="offices.0"
/>
    <attr name="offices.1"          xs:type="string"
other:iskey="false"                other:pathspec="[" other:field="offices.1"
/>
    <attr name="workaddress.city"   xs:type="string"
other:iskey="false"                other:pathspec="{
other:field="workaddress.city" />
    <attr name="workaddress.state"  xs:type="string"
other:iskey="false"                other:pathspec="{
other:field="workaddress.state" />
    <attr name="workaddress.street" xs:type="string"
other:iskey="false"                other:pathspec="{
other:field="workaddress.street" />
  </rsb:info>
</rsb:script>

```

Changelog

General Changes

Date	Build Number	Change Type	Description
[8452] 02/21/2023	8452	Couchbase	<p>Added</p> <ul style="list-style-type: none"> Added support for Analytics views and tabular Analytics views. Tabular Analytics views use the metadata provided as part of the <pre>CREATE ANALYTICS VIEW</pre> <p>DDL statement instead of performing rowscan. Supported column metadata includes column names, types, nullability, primary keys, and foreign keys. Both types of views have similar limitations to external Analytics collections (they do not support Document.Id or Document.TTL, and are not scanned for child tables).</p>
12/14/2022	8383	General	<p>Changed</p> <ul style="list-style-type: none"> Added the Default column to the sys_procedureparameters table.
09/30/2022	8308	General	<p>Changed</p> <ul style="list-style-type: none"> Added the IsPath column to the sys_procedureparameters table.
08/17/2022	8264	General	<p>Changed</p> <ul style="list-style-type: none"> We now support handling the keyword "COLLATE" as standard function name as well.
[8045] 01/10/2022	8045	Couchbase	<p>Added</p> <ul style="list-style-type: none"> Added support for the

NATIVEQUERY table function. This function can be used after a FROM to execute a query using Couchbase-native N1QL instead of SQL. For example,

```
SELECT * FROM
NATIVEQUERY('SELECT META
(a).id, b.rowdata FROM
abucket AS a UNNEST
a.rows AS b')
```

will execute the inner query directly on Couchbase and return the results. This will work even in tools which are not normally compatible with QueryPassthrough=true.

12/20/2021	8024	Couchbase	Added <ul style="list-style-type: none"> Added support for UpdateNullValues. This provides control over whether NULL values written to documents via UPDATE are stored as NULL in Couchbase, or are removed from the document.
10/26/2021	7969	Couchbase	Added <ul style="list-style-type: none"> Added support for N1Q transactions. They apply to all N1QL queries that are not triggered by metadata or stored procedures. They can be either disabled entirely (the default), enabled for explicit use only (like setAutoCommit(false) or BeginTransaction()), or for implicit use where a one-statement transaction is used

			<p>when no explicit transaction is active on the connection. Connection properties were also added to control transaction durability and lifetime requirements.</p> <ul style="list-style-type: none"> Added the corresponding connection properties, UseTransactions, TransactionDurability, and TransactionTimeout.
09/13/2021	7926	Couchbase	<p>Added</p> <ul style="list-style-type: none"> Added support for calling user-defined functions in N1QL and Analytics. Global functions may be called using their unscoped names (to_meters("geo.lat")) or their scoped names ("Default.to_meters("geo.lat")). Scoped functions must be called using their fully qualified names, which are two-parts or three-parts in Analytics ("experiments.to_meters") or three-parts only in N1QL ("experiments.units.to_meters").
09/02/2021	7915	General	<p>Added</p> <ul style="list-style-type: none"> Added support for the STRING_SPLIT table-valued function in the CROSS APPLY clause.
08/07/2021	7889	General	<p>Changed</p> <ul style="list-style-type: none"> Added the KeySeq column to the sys_foreignkeys table.
08/06/2021	7888	General	<p>Changed</p>

			<ul style="list-style-type: none"> Added the new sys_primarykeys system table.
07/23/2021	7874	General	<p>Changed</p> <ul style="list-style-type: none"> Updated the Literal Function Names for relative date/datetime functions. Previously relative date/datetime functions resolved to a different value when used in the projection vs the predicate. I.e: SELECT LAST_MONTH() AS lm, Col FROM Table WHERE Col > LAST_MONTH(). Formerly the two LAST_MONTH() methods would resolve to different datetimes. Now they will match. As a replacement for the previous behavior, the relative date/datetime functions in the criteria may have an 'L' appended to them. I.e: WHERE col > L_LAST_MONTH(). This will continue to resolve to the same values that previously were calculated in the criteria. Note that the "L_" prefix will only work in the predicate - it is not available for the projection.
07/14/2021	7865	Couchbase	<p>Added</p> <ul style="list-style-type: none"> Added support for performing DML on nested child tables in NewChildJoinsMode, which completes our new relational model. You can now INSERT, UPDATE, and DELETE on every table exposed by the provider when NCJM is enabled. Added support for creating

			collections via DDL. When the hidden UseCollectionsForDDL property is enabled, CREATE TABLE "abucket.ascope.acollection"(...) will create the bucket, scope and collection which correspond to the table's name.
07/08/2021	7859	General	Added <ul style="list-style-type: none"> Added the TCP Logging Module for the logging information happening on the TCP wire protocol. The transport bytes that are incoming and ongoing will be logged at verbosity=5.
05/12/2021	7802	Couchbase	Added <ul style="list-style-type: none"> Added support for creating collections via DDL. When the hidden UseCollectionsForDDL property is enabled, CREATE TABLE "abucket.ascope.acollection"(...) will create the bucket, scope and collection which correspond to the table's name.
04/23/2021	7785	General	Added <ul style="list-style-type: none"> Added support for handling client side formulas during insert / update. For example: UPDATE Table SET Col1 = Concat(Col1, " - ", Col2) WHERE Col2 LIKE 'A%'
04/23/2021	7783	General	Changed <ul style="list-style-type: none"> Updated how display sizes are determined for varchar primary

			key and foreign key columns so they will match the reported length of the column.
04/16/2021	7776	General	<p>Added</p> <ul style="list-style-type: none"> Non-conditional updates between two columns is now available to all drivers. For example: UPDATE Table SET Col1=Col2 <p>Changed</p> <ul style="list-style-type: none"> Reduced the length to 255 for varchar primary key and foreign key columns. Updated implicit and metadata caching to improve performance and support for multiple connections. Old metadata caches are not compatible - you would need to generate new metadata caches if you are currently using CacheMetadata. Updated index naming convention to avoid duplicates Updated and standardized Getting Started connection help. Added the Advanced Features section to the help of all drivers. Categorized connection property listings in the help for all editions.
04/15 /2021	7775	General	<p>Changed</p> <ul style="list-style-type: none"> Kerberos authentication is updated to use TCP by default, but will fall back to UDP if a TCP connection cannot be established

04/12/2021	7772	Couchbase	Added <ul style="list-style-type: none"> Added support for the USE KEYS query construct. When executing an N1QL query, the driver will attempt to determine if it contains any eligible filters on Document.Id - if there are any they are removed from the WHERE clause and migrated to the USE KEYS clause. When this transformation is applied the resulting query can avoid index scans, which allows for more queries to be run without a primary index and improves execution speed.
03/12/2021	7741	Couchbase	Changed <ul style="list-style-type: none"> Updated AddDocuments and ManageIndices so the interface no longer operates based on column#1 values. AddDocuments now accepts either a single ID and Document or a SourceTable that refers to a #TEMP table (analogous to a normal bulk insert), while ManageIndices accepts JSON arrays for multiple values since these are mostly typed by hand and have very few elements.
01/19/2021	7689	Couchbase	Added <ul style="list-style-type: none"> Added support for collections and scopes within Couchbase v7. This touches on a lot of the driver but here are the main aspects: <ul style="list-style-type: none"> Tables are now generated for collections as well as

buckets/datasets in both N1QL and Analytics. They have dotted names similar to Analytics datasets. The only exceptions are N1QL default collections which just use the bucket name (the server calls them **_default**) and Analytics legacy dataverses (which have the same two-level hierarchy)

- Queries can now use data from collections in all the usual ways (flavors, UNNEST, etc), and collection tables can now be used with DROP TABLE. Buckets still work with DROP TABLE as well but only in limited contexts to prevent users from deleting data outside the default collection.
 - Added new stored procedures for creating and dropping scopes/collections, and updated index management stored procedures to report information about scopes/collections.
 - The Dataverse option now expects to follow SQL quoting rules, and can be either a single SQL identifier (**foo**) for legacy dataverses, or a two-level qualified identifier (**foo.bar**) for Analytics dataverses/scopes
-

Advanced Features

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

User Defined Views

The adapter allows you to define virtual tables, called *user defined views*, whose contents are decided by a pre-configured query. These views are useful when you cannot directly control queries being issued to the drivers. See [User Defined Views](#) for an overview of creating and configuring custom views.

SSL Configuration

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

Firewall and Proxy

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

Query Processing

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

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

Logging

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

User Defined Views

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

There are two ways to create user defined views:

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

Defining Views Using a Configuration File

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

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

This User Defined View configuration file is formatted as follows:

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

For example:

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

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

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

Defining Views Using DDL Statements

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

Create a View

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

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

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

Alter a View

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

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

The view is then updated in the JSON configuration file.

Drop a View

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

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

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

Schema for User Defined Views

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

Working with User Defined Views

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

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

An example of a query to the driver:

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

Resulting in the effective query to the source:

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

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

SSL Configuration

Customizing the SSL Configuration

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

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

Client SSL Certificates

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

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

Firewall and Proxy

Connecting Through a Firewall or Proxy

HTTP Proxies

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

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

Other Proxies

Set the following properties:

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

Query Processing

Query Processing

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

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

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

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

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

More Information

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

Logging

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

Basic Logging

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

- **Logfile:** A filepath which designates the name and location of the log file.
- **Verbosity:** This is a numerical value (1-5) that determines the amount of detail in the log. See the page in the Connection Properties section for an explanation of the five levels.

- MaxLogFileSize: When the limit is hit, a new log is created in the same folder with the date and time appended to the end. The default limit is 100 MB. Values lower than 100 kB will use 100 kB as the value instead.
- MaxLogFileCount: A string specifying the maximum file count of log files. When the limit is hit, a new log is created in the same folder with the date and time appended to the end and the oldest log file will be deleted. Minimum supported value is 2. A value of 0 or a negative value indicates no limit on the count.

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

Log Verbosity

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

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

The Verbosity should not be set to greater than 1 for normal operation. Substantial amounts of data can be logged at higher verboisities, which can delay execution times.

To refine the logged content further by showing/hiding specific categories of information, see LogModules.

Sensitive Data

Verbosity levels 3 and higher may capture information that you do not want shared outside of your organization. The following lists information of concern for each level:

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

Best Practices for Data Security

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

Java Logging

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

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

Advanced Logging

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

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

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

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

An example value for this property would be.

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

Note that these modules refine the information as it is pulled after taking the Verbosity into account.

SQL Compliance

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

SELECT Statements

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

See [Data Model](#) for information on the capabilities of the Couchbase API.

INSERT Statements

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

UPDATE Statements

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

UPSERT Statements

An UPSERT updates a record if it exists and inserts the record if it does not. See [UPSERT Statements](#) for a syntax reference and examples.

DELETE Statements

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

EXECUTE Statements

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

Names and Quoting

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

Transactions and Batching

Transactions are not currently supported.

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

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 Couchbase adapter:

```
SELECT {
  [ TOP <numeric_literal> | DISTINCT ]
  {
    *
    | {
      <expression> [ [ AS ] <column_reference> ]
      | { <table_name> | <correlation_name> } .*
    } [ , ... ]
  }
  [ INTO csv:// [ filename= ] <file_path> [ ;delimiter=tab ] ]
  {
    FROM <table_reference> [ [ AS ] <identifier> ]
  }
  [ WHERE <search_condition> ]
  [ GROUP BY <column_reference> [ , ... ] ]
  [ HAVING <search_condition> ]
  [
```

```

ORDER BY
<column_reference> [ ASC | DESC ] [ NULLS FIRST | NULLS LAST ]
]
[
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> { = | != | < | <= | > | >= | IS NULL | LIKE | AND |
OR | NOT | IN } [ <expression> ]
} [ { AND | OR } ... ]

```

Examples

1. Return all columns:

```
SELECT * FROM Customer
```

2. Rename a column:

```
SELECT "TotalDue" AS MY_TotalDue FROM Customer
```

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

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

```
Customer
```

4. Search data:

```
SELECT * FROM Customer WHERE CustomerId = '12345'
```

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

```
SELECT * FROM Customer WHERE CustomerId = '12345';
```

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

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

7. Return the unique items matching the query criteria:

```
SELECT DISTINCT TotalDue FROM Customer
```

8. Summarize data:

```
SELECT TotalDue, MAX(AnnualRevenue) FROM Customer GROUP BY  
TotalDue
```

See [Aggregate Functions](#) for details.

9. Sort a result set in ascending order:

```
SELECT Name, TotalDue FROM Customer ORDER BY TotalDue ASC
```

Aggregate Functions

Examples of Aggregate Functions

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

COUNT

Returns the number of rows matching the query criteria.

```
SELECT COUNT(*) FROM Customer WHERE CustomerId = '12345'
```

AVG

Returns the average of the column values.

```
SELECT TotalDue, AVG(AnnualRevenue) FROM Customer WHERE CustomerId =  
'12345' GROUP BY TotalDue
```

MIN

Returns the minimum column value.

```
SELECT MIN(AnnualRevenue), TotalDue FROM Customer WHERE CustomerId =  
'12345' GROUP BY TotalDue
```

MAX

Returns the maximum column value.

```
SELECT TotalDue, MAX(AnnualRevenue) FROM Customer WHERE CustomerId =  
'12345' GROUP BY TotalDue
```

SUM

Returns the total sum of the column values.

```
SELECT SUM(AnnualRevenue) FROM Customer WHERE CustomerId = '12345'
```

Projection Functions

ARRAY_AGG(column)

Returns array of the non-MISSING values in the group, including NULL values.

- **column:** Any column expression.

ARRAY_APPEND(column, value)

Returns new array with value appended.

- **column:** Any column expression.
- **value:** The value to be appended to the array.

ARRAY_CONCAT(column1, column2)

Returns new array with the concatenation of the input arrays.

- **column1:** Any column expression.
- **column2:** Any column expression.

ARRAY_DISTINCT(column)

Returns new array with distinct elements of input array.

- **column:** Any column expression.

ARRAY_IFNULL(column)

Returns the first non-NULL value in the array, or NULL.

- **column:** Any column expression.

ARRAY_PREPEND(column, value)

Returns new array with value pre-pended.

- **column:** Any column expression.

- **value:** The value to be pre-pended to the array.

ARRAY_PUT(column, value)

Returns new array with value appended, if value is not already present, otherwise returns the unmodified input array.

- **column:** Any column expression.
- **value:** The value to append to the array.

ARRAY_REMOVE(column, value)

Returns new array with all occurrences of value removed.

- **column:** Any column expression.
- **value:** The value to be removed from the array.

ARRAY_REPLACE(column, value1, value2 [, integer_n])

Returns new array with all occurrences of value removed.

- **column:** Any column expression.
- **value1:** The value to be replaced by value2.
- **value2:** The value to replace value1.
- **integer_n:** The maximum number of replacements to be performed.

ARRAY_REVERSE(column)

Returns new array with all elements in reverse order.

- **column:** Any column expression.

ARRAY_SORT(column)

Returns new array with elements sorted in N1QL collation order.

- **column:** Any column expression.

DECODE_JSON(column)

Unmarshals the JSON-encoded string into a N1QL value. The empty string is MISSING.

- **column:** Any column expression.

ENCODE_JSON(column)

Marshals the N1QL value into a JSON-encoded string. MISSING becomes the empty string.

- **column:** Any column expression.

ENCODED_SIZE(column)

Number of bytes in an uncompressed JSON encoding of the value. The exact size is implementation-dependent. Always returns an integer, and never MISSING or NULL. Returns 0 for MISSING.

- **column:** Any column expression.

POLY_LENGTH(column)

Returns length of the value after evaluating the expression. The exact meaning of length depends on the type of the value: MISSING: MISSING; NULL: NULL; String: The length of the string.; Array: The number of elements in the array.; Object: The number of name/value pairs in the object; Any other value: NULL.

- **column:** Any column expression.

OBJECT_LENGTH(column)

Returns number of name-value pairs in the object.

- **column:** Any column expression.

OBJECT_NAMES(column)

Returns array containing the attribute names of the object, in N1QL collation order.

- **column:** Any column expression.

OBJECT_PAIRS(column)

Returns array containing the attribute name and value pairs of the object, in N1QL collation order of the names.

- **column:** Any column expression.

OBJECT_VALUES(column)

Returns array containing the attribute values of the object, in N1QL collation order of the corresponding names.

- **column:** Any column expression.

ARRAY_AVG(column)

Returns arithmetic mean (average) of all the non-NULL number values in the array, or NULL if there are no such values.

- **column:** Any column expression.

ARRAY_CONTAINS(column, value)

Returns true if the array contains value.

- **column:** Any column expression.
- **value:** The value contained within the array.

ARRAY_COUNT(column)

Returns count of all the non-NULL values in the array, or zero if there are no such values.

- **column:** Any column expression.

ARRAY_LENGTH(column)

Returns the number of elements in the array.

- **column:** Any column expression.

ARRAY_MAX(column)

Returns the largest non-NULL, non-MISSING array element, in N1QL collation order.

- **column:** Any column expression.

ARRAY_MIN(column)

Returns smallest non-NULL, non-MISSING array element, in N1QL collation order.

- **column:** Any column expression.

ARRAY_POSITION(column, value)

Returns the first position of value within the array, or -1. Array position is zero-based, i.e. the first position is 0.

- **column:** Any column expression.
- **value:** The value contained within the array.

ARRAY_SUM(column)

Sum of all the non-NULL number values in the array, or zero if there are no such values.

- **column:** Any column expression.

GREATEST(column1, column2 [,column3 [,column4]])

Largest non-NULL, non-MISSING value if the values are of the same type; otherwise NULL.

- **column1**: Any column expression.
- **column2**: Any column expression.
- **column3**: Any column expression.
- **column4**: Any column expression.

LEAST(column1, column2 [,column3 [,column4]])

Returns smallest non-NULL, non-MISSING value if the values are of the same type, otherwise returns NULL.

- **column1**: Any column expression.
- **column2**: Any column expression.
- **column3**: Any column expression.
- **column4**: Any column expression.

IFMISSING(column1, column2 [,column3 [,column4]])

Returns the first non-MISSING value.

- **column1**: Any column expression.
- **column2**: Any column expression.
- **column3**: Any column expression.
- **column4**: Any column expression.

IFMISSINGORNULL(column1, column2 [,column3 [,column4]])

Returns first non-NULL, non-MISSING value.

- **column1**: Any column expression.
- **column2**: Any column expression.
- **column3**: Any column expression.
- **column4**: Any column expression.

IFNULL(column1, column2 [,column3 [,column4]])

Returns first non-NULL value. Note that this function might return MISSING if there is no non-NULL value.

- **column1:** Any column expression.
- **column2:** Any column expression.
- **column3:** Any column expression.
- **column4:** Any column expression.

MISSINGIF(column1, column2)

Returns MISSING if column1 = column2, otherwise returns column1. Returns MISSING or NULL if either input is MISSING or NULL.

- **column1:** Any column expression.
- **column2:** Any column expression.

NULLIF(column1, column2)

Returns NULL if column1 = column2, otherwise returns column1. Returns MISSING or NULL if either input is MISSING or NULL.

- **column1:** Any column expression.
- **column2:** Any column expression.

IFINF(column1, column2 [,column3 [,column4]])

Returns first non-MISSING, non-Inf number. Returns MISSING or NULL if a non-number input is encountered first.

- **column1:** Any column expression.
- **column2:** Any column expression.
- **column3:** Any column expression.
- **column4:** Any column expression.

IFNAN(column1, column2 [,column3 [,column4]])

Returns first non-MISSING, non-NaN number. Returns MISSING or NULL if a non-number input is encountered first.

- **column1:** Any column expression.
- **column2:** Any column expression.
- **column3:** Any column expression.
- **column4:** Any column expression.

IFNANORINF(column1, column2 [,column3 [,column4]])

Returns first non-MISSING, non-Inf, or non-NaN number. Returns MISSING or NULL if a non-number input is encountered first.

- **column1:** Any column expression.
- **column2:** Any column expression.
- **column3:** Any column expression.
- **column4:** Any column expression.

NANIF(column1, column2 [,column3 [,column4]])

Returns NaN if column1 = column2, otherwise returns column1. Returns MISSING or NULL if either input is MISSING or NULL.

- **column1:** Any column expression.
- **column2:** Any column expression.
- **column3:** Any column expression.
- **column4:** Any column expression.

NEGINFIF(column1, column2 [,column3 [,column4]])

Returns NegInf if column1 = column2, otherwise returns column1. Returns MISSING or NULL if either input is MISSING or NULL.

- **column1:** Any column expression.
- **column2:** Any column expression.
- **column3:** Any column expression.
- **column4:** Any column expression.

POSINFIF(column1, column2 [,column3 [,column4]])

Returns PosInf if column1 = column2, otherwise returns column1. Returns MISSING or NULL if either input is MISSING or NULL.

- **column1:** Any column expression.
- **column2:** Any column expression.
- **column3:** Any column expression.
- **column4:** Any column expression.

CLOCK_MILLIS()

Returns system clock at function evaluation time, as UNIX milliseconds. Varies during a query.

CLOCK_STR([string_fmt])

Returns system clock at function evaluation time, as a string in a supported format. Varies during a query.

- **string_fmt:** The datetime format to return the system clock in.

DATE_ADD_MILLIS(column, integer_n, string_part)

Performs date arithmetic, and returns result of computation. n and part are used to define an interval or duration, which is then added (or subtracted) to the UNIX time stamp, returning the result.

- **column:** Any column expression.
- **integer_n:** The number of string_part's to add to the column value.

- **string_part:** The part to add integer_n to, available values are: millennium, century, decade, year, quarter, month, week, day, hour, minute, second, and millisecond.

DATE_ADD_STR(column, integer_n, string_part)

Performs date arithmetic. n and part are used to define an interval or duration, which is then added (or subtracted) to the date string in a supported format, returning the result.

- **column:** Any column expression.
- **integer_n:** The number of string_part's to add to the column value.
- **string_part:** The part to add integer_n to, available values are: millennium, century, decade, year, quarter, month, week, day, hour, minute, second, and millisecond.

DATE_DIFF_MILLIS(column1, column2, string_part)

Performs date arithmetic. Returns the elapsed time between two UNIX time stamps as an integer whose unit is part.

- **column1:** Any column expression.
- **column2:** Any column expression.
- **string_part:** The unit of the result, available values are: millennium, century, decade, year, quarter, month, week, day, hour, minute, second, and millisecond.

DATE_DIFF_STR(column1, column2, string_part)

Performs date arithmetic. Returns the elapsed time between two date strings in a supported format, as an integer whose unit is part.

- **column1:** Any column expression.
- **column2:** Any column expression.
- **string_part:** The unit of the result, available values are: millennium, century, decade, year, quarter, month, week, day, hour, minute, second, and millisecond.

DATE_PART_MILLIS(column1, string_part [, tz])

Returns date part as an integer. The date expression is a number representing UNIX milliseconds, and part is one of the following date part strings.

- **column1:** Any column expression.
- **string_part:** The component of the date to extract. Available values are: millennium, century, decade, year, quarter, month, week, day, hour, minute, second, millisecond, day_of_year, day_of_week, iso_week, iso_year, iso_dow, timezone, timezone_hour, and timezone_minute.
- **tz:** The timezone to convert the local time to. Default to the system timezone if not specified. If an incorrect time zone is provided, the null is returned.

DATE_PART_STR(column1, string_part)

Returns date part as an integer. The date expression is a string in a supported format, and part is one of the supported date part strings.

- **column1:** Any column expression.
- **string_part:** The unit of the result, available values are: millennium, century, decade, year, quarter, month, week, day, hour, minute, second, millisecond, day_of_year, day_of_week, iso_week, iso_year, iso_dow, timezone, timezone_hour, and timezone_minute.

DATE_TRUNC_MILLIS(column1, string_part)

Returns UNIX time stamp that has been truncated so that the given date part string is the least significant.

- **column1:** Any column expression.
- **string_part:** The least significant date part, available values are: millennium, century, decade, year, quarter, month, week, day, hour, minute, second, and millisecond.

DATE_TRUNC_STR(column1, string_part)

Returns ISO 8601 time stamp that has been truncated so that the given date part string is the least significant.

- **column1**: Any column expression.
- **string_part**: The least significant date part, available values are: millennium, century, decade, year, quarter, month, week, day, hour, minute, second, and millisecond.

MILLIS(column1)

Returns date that has been converted in a supported format to UNIX milliseconds.

- **column1**: Any column expression.

STR_TO_MILLIS(column1)

Returns date that has been converted in a supported format to UNIX milliseconds.

- **column1**: Any column expression.

MILLIS_TO_STR(column [, string_fmt])

Returns the string in the supported format to which the UNIX milliseconds has been converted.

- **column1**: Any column expression.
- **string_fmt**: The datetime format to return the system clock in.

MILLIS_TO_UTC(column [, string_fmt])

Returns the UTC string to which the UNIX time stamp has been converted in the supported format.

- **column1**: Any column expression.
- **string_fmt**: The datetime format to return the system clock in.

MILLIS_TO_TZ(column, string_tzname [, string_fmt])

Converts the UNIX time stamp to a string in the named time zone, and returns the string.

- **column1**: Any column expression.
- **string_tzname**: The time zone name.
- **string_fmt**: The datetime format to return the system clock in.

NOW_MILLIS()

Returns statement time stamp as UNIX milliseconds; does not vary during a query.

NOW_STR([string_fmt])

Returns statement time stamp as a string in a supported format; does not vary during a query.

- **string_fmt**: The datetime format to return the timestamp in.

STR_TO_UTC(column1)

Converts the ISO 8601 time stamp to UTC.

- **column1**: Any column expression.

STR_TO_ZONE_NAME(column, string_tzname)

Converts the supported time stamp string to the named time zone.

- **column1**: Any column expression.
- **string_tzname**: The time zone name.

BASE64(expression)

Returns base64 encoding of expression.

- **expression**: Any column or literal expression.

ABS(expression)

Returns absolute value of the number.

- **expression:** Any column or literal expression.

ACOS(expression)

Returns arccosine in radians.

- **expression:** Any column or literal expression.

ASIN(expression)

Returns arcsine in radians.

- **expression:** Any column or literal expression.

ATAN(expression)

Returns arctangent in radians.

- **expression:** Any column or literal expression.

ATAN2(expression1, expression2)

Returns arctangent of expression2/expression1.

- **expression1:** Any column or literal expression.
- **expression2:** Any column or literal expression.

CEIL(expression)

Returns smallest integer not less than the number.

- **expression:** Any column or literal expression.

COS(expression)

Returns cosine.

- **expression:** Any column or literal expression.

DEGREES(expression)

Returns radians to degrees.

- **expression:** Any column or literal expression.

E()

Base of natural logarithms.

EXP(expression)

Returns $e^{\text{expression}}$.

- **expression:** Any column or literal expression.

LN(expression)

Returns log base e.

- **expression:** Any column or literal expression.

LOG(expression)

Returns log base 10.

- **expression:** Any column or literal expression.

FLOOR(expression)

Largest integer not greater than the number.

- **expression:** Any column or literal expression.

PI()

Returns PI.

POWER(expression1, expression2)

Returns $\text{expression1}^{\text{expression2}}$.

- **expression1**: Any column or literal expression.
- **expression2**: Any column or literal expression.

RADIANS(expression)

Returns degrees to radians.

- **expression**: Any column or literal expression.

RANDOM([expression])

Returns pseudo-random number with optional seed.

- **expression**: Any column or literal expression.

ROUND(expression [, integer_digits])

Rounds the value to the given number of integer digits to the right of the decimal point (left if digits is negative). Digits is 0 if not given.

- **expression**: Any column or literal expression.
- **integer_digits**: The number of digits to round to.

SIGN(expression)

Valid values: -1, 0, or 1 for negative, zero, or positive numbers respectively.

- **expression**: Any column or literal expression.

SIN(expression)

Returns sine.

- **expression:** Any column or literal expression.

SQRT(expression)

Returns square root.

- **expression:** Any column or literal expression.

TAN(expression)

Returns tangent.

- **expression:** Any column or literal expression.

TRUNC(expression [, integer_digits])

Truncates the number to the given number of integer digits to the right of the decimal point (left if digits is negative). Digits is 0 if not given.

- **expression:** Any column or literal expression.
- **integer_digits:** The number of digits to truncate.

CONTAINS(column, string_substring)

True if the string contains the substring.

- **column:** Any column or literal expression.
- **string_substring:** The substring to search for.

INITCAP(column)

Converts the string so that the first letter of each word is uppercase and every other letter is lowercase.

- **column:** Any column or literal expression.

LENGTH(column)

Returns length of the string value.

- **column:** Any column or literal expression.

LOWER(column)

Returns lowercase of the string value.

- **column:** Any column or literal expression.

LTRIM(column [, string_chars])

Returns string with all leading chars removed. White space by default.

- **column:** Any column or literal expression.
- **string_chars:** The leading characters to remove.

POSITION(column, string_substring)

Returns the first position of the substring within the string, or -1. The position is zero-based, i.e., the first position is 0.

- **column:** Any column or literal expression.
- **string_substring:** The substring to search for.

REPEAT(column, integer_n)

Returns string formed by repeating expression n times.

- **column:** Any column or literal expression.
- **integer_n:** The number of times to repeat column.

REPLACE(column, string_substring, string_replace [, integer_n])

Returns string with all occurrences of substr replaced with repl. If n is given, at most n replacements are performed.

- **column:** The column expression.
- **string_substring:** The regular expression to match.
- **string_replace:** The value to replace the matched pattern.
- **integer_n:** The maximum number of replacements to make.

RTRIM(column [, string_chars])

Returns string with all trailing chars removed. White space by default.

- **column:** Any column or literal expression.
- **string_chars:** The trailing characters to remove.

SPLIT(column [, string_sep])

Splits the string into an array of substrings separated by string_sep. If string_sep is not given, any combination of white space characters is used.

- **column:** Any column or literal expression.
- **string_sep:** The separator to split column on.

SUBSTR(column, integer_position [, integer_length])

Returns substring from the integer position of the given length, or to the end of the string. The position is zero-based, i.e. the first position is 0. If position is negative, it is counted from the end of the string; -1 is the last position in the string.

- **column:** Any column or literal expression.
- **integer_position:** The starting position.
- **integer_length:** The total length of the substring to retrieve.

TRIM(column [, string_chars])

Returns string with all leading and trailing chars removed. White space by default.

- **column**: Any column or literal expression.
- **string_chars**: The leading and trailing characters to remove.

UPPER(column)

Returns uppercase of the string value.

- **column**: Any column or literal expression.

TOARRAY(column)

Returns array as follows: MISSING is MISSING; NULL is NULL; Arrays are themselves; All other values are wrapped in an array.

- **column**: Any column expression.

TOATOM(column)

Returns array as follows: MISSING is MISSING; NULL is NULL; Arrays of length 1 are the result of TOATOM() on their single element; Objects of length 1 are the result of TOATOM() on their single value; Booleans, numbers, and strings are themselves; All other values are NULL.

- **column**: Any column expression.

TOBOOLEAN(column)

Returns array as follows: MISSING is MISSING; NULL is NULL; False is false; Numbers +0, -0, and NaN are false; Empty strings, arrays, and objects are false; All other values are true.

- **column**: Any column expression.

TONUMBER(column)

Returns array as follows: MISSING is MISSING; NULL is NULL; False is 0; True is 1; Numbers are themselves; Strings that parse as numbers are those numbers; All other values are NULL.

- **column:** Any column expression.

TOOBJECT(column)

Returns array as follows: MISSING is MISSING; NULL is NULL; Objects are themselves; All other values are the empty object.

- **column:** Any column expression.

TOSTRING(column)

Returns array as follows: MISSING is MISSING; NULL is NULL; False is "false"; True is "true"; Numbers are their string representation; Strings are themselves; All other values are NULL.

- **column:** Any column expression.

Predicate Functions

REGEXP_CONTAINS(column, string_pattern)

Returns True if the string value contains the regular expression pattern.

- **column:** The column expression.
- **string_pattern:** The regular expression to match.

REGEXP_LIKE(column, string_pattern)

Returns True if the string value matches the regular expression pattern.

- **column:** The column expression.

- **string_pattern**: The regular expression to match.

REGEXP_POSITION(column, string_pattern)

Returns first position of the regular expression pattern within the string, or -1.

- **column**: The column expression.
- **string_pattern**: The regular expression to match.

REGEXP_REPLACE(column, string_pattern, string_replace [, integer_n])

Returns new string with occurrences of pattern replaced with string_replace. If n is given, at most n replacements are performed.

- **column**: The column expression.
- **string_pattern**: The regular expression to match.
- **string_replace**: The value to replace the matched pattern.
- **integer_n**: The maximum number of replacements to make.

ISARRAY(column)

Returns True if expression is an array, otherwise returns MISSING, NULL or false.

- **column**: Any column expression.

ISATOM(column)

Returns True if expression is a Boolean, number, or string, otherwise returns MISSING, NULL or false.

- **column**: Any column expression.

ISBOOLEAN(column)

Returns True if expression is a Boolean, otherwise returns MISSING, NULL or false.

- **column:** Any column expression.

ISNUMBER(column)

Returns True if expression is a number, otherwise returns MISSING, NULL or false.

- **column:** Any column expression.

ISOBJECT(column)

Returns True if expression is an object, otherwise returns MISSING, NULL or false.

- **column:** Any column expression.

ISSTRING(column)

Returns True if expression is a string, otherwise returns MISSING, NULL or false.

- **column:** Any column expression.

TYPE(column)

Returns one of the following strings, based on the value of expression: missing, null, boolean, number, string, array, object, or binary.

- **column:** Any column expression.

ARRAY_AVG(column)

Returns arithmetic mean (average) of all the non-NULL number values in the array, or NULL if there are no such values.

- **column:** Any column expression.

ARRAY_CONTAINS(column, value)

Returns true if the array contains value.

- **column:** Any column expression.
- **value:** The value contained within the array.

ARRAY_COUNT(column)

Returns count of all the non-NULL values in the array, or zero if there are no such values.

- **column:** Any column expression.

ARRAY_LENGTH(column)

Returns the number of elements in the array.

- **column:** Any column expression.

ARRAY_MAX(column)

Returns the largest non-NULL, non-MISSING array element, in N1QL collation order.

- **column:** Any column expression.

ARRAY_MIN(column)

Returns smallest non-NULL, non-MISSING array element, in N1QL collation order.

- **column:** Any column expression.

ARRAY_POSITION(column, value)

Returns the first position of value within the array, or -1. Array position is zero-based, i.e. the first position is 0.

- **column:** Any column expression.
- **value:** The value contained within the array.

ARRAY_SUM(column)

Sum of all the non-NULL number values in the array, or zero if there are no such values.

- **column:** Any column expression.

GREATEST(column1, column2 [,column3 [,column4]])

Largest non-NULL, non-MISSING value if the values are of the same type; otherwise NULL.

- **column1:** Any column expression.
- **column2:** Any column expression.
- **column3:** Any column expression.
- **column4:** Any column expression.

LEAST(column1, column2 [,column3 [,column4]])

Returns smallest non-NULL, non-MISSING value if the values are of the same type, otherwise returns NULL.

- **column1:** Any column expression.
- **column2:** Any column expression.
- **column3:** Any column expression.
- **column4:** Any column expression.

IFMISSING(column1, column2 [,column3 [,column4]])

Returns the first non-MISSING value.

- **column1:** Any column expression.
- **column2:** Any column expression.
- **column3:** Any column expression.
- **column4:** Any column expression.

IFMISSINGORNULL(column1, column2 [,column3 [,column4]])

Returns first non-NULL, non-MISSING value.

- **column1**: Any column expression.
- **column2**: Any column expression.
- **column3**: Any column expression.
- **column4**: Any column expression.

IFNULL(column1, column2 [,column3 [,column4]])

Returns first non-NULL value. Note that this function might return MISSING if there is no non-NULL value.

- **column1**: Any column expression.
- **column2**: Any column expression.
- **column3**: Any column expression.
- **column4**: Any column expression.

MISSINGIF(column1, column2)

Returns MISSING if column1 = column2, otherwise returns column1. Returns MISSING or NULL if either input is MISSING or NULL.

- **column1**: Any column expression.
- **column2**: Any column expression.

NULLIF(column1, column2)

Returns NULL if column1 = column2, otherwise returns column1. Returns MISSING or NULL if either input is MISSING or NULL.

- **column1**: Any column expression.
- **column2**: Any column expression.

IFINF(column1, column2 [,column3 [,column4]])

Returns first non-MISSING, non-Inf number. Returns MISSING or NULL if a non-number input is encountered first.

- **column1**: Any column expression.
- **column2**: Any column expression.
- **column3**: Any column expression.
- **column4**: Any column expression.

IFNAN(column1, column2 [,column3 [,column4]])

Returns first non-MISSING, non-NaN number. Returns MISSING or NULL if a non-number input is encountered first.

- **column1**: Any column expression.
- **column2**: Any column expression.
- **column3**: Any column expression.
- **column4**: Any column expression.

IFNANORINF(column1, column2 [,column3 [,column4]])

Returns first non-MISSING, non-Inf, or non-NaN number. Returns MISSING or NULL if a non-number input is encountered first.

- **column1**: Any column expression.
- **column2**: Any column expression.
- **column3**: Any column expression.
- **column4**: Any column expression.

NANIF(column1, column2 [,column3 [,column4]])

Returns NaN if column1 = column2, otherwise returns column1. Returns MISSING or NULL if either input is MISSING or NULL.

- **column1**: Any column expression.
- **column2**: Any column expression.
- **column3**: Any column expression.

- **column4:** Any column expression.

NEGINFIF(column1, column2 [,column3 [,column4]])

Returns NegInf if column1 = column2, otherwise returns column1. Returns MISSING or NULL if either input is MISSING or NULL.

- **column1:** Any column expression.
- **column2:** Any column expression.
- **column3:** Any column expression.
- **column4:** Any column expression.

POSINFIF(column1, column2 [,column3 [,column4]])

Returns PosInf if column1 = column2, otherwise returns column1. Returns MISSING or NULL if either input is MISSING or NULL.

- **column1:** Any column expression.
- **column2:** Any column expression.
- **column3:** Any column expression.
- **column4:** Any column expression.

CLOCK_MILLIS()

Returns system clock at function evaluation time, as UNIX milliseconds. Varies during a query.

CLOCK_STR([string_fmt])

Returns system clock at function evaluation time, as a string in a supported format. Varies during a query.

- **string_fmt:** The datetime format to return the system clock in.

DATE_ADD_MILLIS(column, integer_n, string_part)

Performs date arithmetic, and returns result of computation. *n* and *part* are used to define an interval or duration, which is then added (or subtracted) to the UNIX time stamp, returning the result.

- **column:** Any column expression.
- **integer_n:** The number of *string_part*'s to add to the column value.
- **string_part:** The part to add *integer_n* to, available values are: millennium, century, decade, year, quarter, month, week, day, hour, minute, second, and millisecond.

DATE_ADD_STR(column, integer_n, string_part)

Performs date arithmetic. *n* and *part* are used to define an interval or duration, which is then added (or subtracted) to the date string in a supported format, returning the result.

- **column:** Any column expression.
- **integer_n:** The number of *string_part*'s to add to the column value.
- **string_part:** The part to add *integer_n* to, available values are: millennium, century, decade, year, quarter, month, week, day, hour, minute, second, and millisecond.

DATE_DIFF_MILLIS(column1, column2, string_part)

Performs date arithmetic. Returns the elapsed time between two UNIX time stamps as an integer whose unit is *part*.

- **column1:** Any column expression.
- **column2:** Any column expression.
- **string_part:** The unit of the result, available values are: millennium, century, decade, year, quarter, month, week, day, hour, minute, second, and millisecond.

DATE_DIFF_STR(column1, column2, string_part)

Performs date arithmetic. Returns the elapsed time between two date strings in a supported format, as an integer whose unit is *part*.

- **column1:** Any column expression.
- **column2:** Any column expression.
- **string_part:** The unit of the result, available values are: millennium, century, decade, year, quarter, month, week, day, hour, minute, second, and millisecond.

DATE_PART_MILLIS(column1, string_part [, tz])

Returns date part as an integer. The date expression is a number representing UNIX milliseconds, and part is one of the following date part strings.

- **column1:** Any column expression.
- **string_part:** The component of the date to extract. Available values are: millennium, century, decade, year, quarter, month, week, day, hour, minute, second, millisecond, day_of_year, day_of_week, iso_week, iso_year, iso_dow, timezone, timezone_hour, and timezone_minute.
- **tz:** The timezone to convert the local time to. Default to the system timezone if not specified. If an incorrect time zone is provided, the null is returned.

DATE_PART_STR(column1, string_part)

Returns date part as an integer. The date expression is a string in a supported format, and part is one of the supported date part strings.

- **column1:** Any column expression.
- **string_part:** The unit of the result, available values are: millennium, century, decade, year, quarter, month, week, day, hour, minute, second, millisecond, day_of_year, day_of_week, iso_week, iso_year, iso_dow, timezone, timezone_hour, and timezone_minute.

DATE_TRUNC_MILLIS(column1, string_part)

Returns UNIX time stamp that has been truncated so that the given date part string is the least significant.

- **column1:** Any column expression.
- **string_part:** The least significant date part, available values are: millennium,

century, decade, year, quarter, month, week, day, hour, minute, second, and millisecond.

DATE_TRUNC_STR(column1, string_part)

Returns ISO 8601 time stamp that has been truncated so that the given date part string is the least significant.

- **column1:** Any column expression.
- **string_part:** The least significant date part, available values are: millennium, century, decade, year, quarter, month, week, day, hour, minute, second, and millisecond.

MILLIS(column1)

Returns date that has been converted in a supported format to UNIX milliseconds.

- **column1:** Any column expression.

STR_TO_MILLIS(column1)

Returns date that has been converted in a supported format to UNIX milliseconds.

- **column1:** Any column expression.

MILLIS_TO_STR(column [, string_fmt])

Returns the string in the supported format to which the UNIX milliseconds has been converted.

- **column1:** Any column expression.
- **string_fmt:** The datetime format to return the system clock in.

MILLIS_TO_UTC(column [, string_fmt])

Returns the UTC string to which the UNIX time stamp has been converted in the supported format.

- **column1**: Any column expression.
- **string_fmt**: The datetime format to return the system clock in.

MILLIS_TO_TZ(column, string_tzname [, string_fmt])

Converts the UNIX time stamp to a string in the named time zone, and returns the string.

- **column1**: Any column expression.
- **string_tzname**: The time zone name.
- **string_fmt**: The datetime format to return the system clock in.

NOW_MILLIS()

Returns statement time stamp as UNIX milliseconds; does not vary during a query.

NOW_STR([string_fmt])

Returns statement time stamp as a string in a supported format; does not vary during a query.

- **string_fmt**: The datetime format to return the timestamp in.

STR_TO_UTC(column1)

Converts the ISO 8601 time stamp to UTC.

- **column1**: Any column expression.

STR_TO_ZONE_NAME(column, string_tzname)

Converts the supported time stamp string to the named time zone.

- **column1**: Any column expression.
- **string_tzname**: The time zone name.

BASE64(expression)

Returns base64 encoding of expression.

- **expression**: Any column or literal expression.

ABS(expression)

Returns absolute value of the number.

- **expression**: Any column or literal expression.

ACOS(expression)

Returns arccosine in radians.

- **expression**: Any column or literal expression.

ASIN(expression)

Returns arcsine in radians.

- **expression**: Any column or literal expression.

ATAN(expression)

Returns arctangent in radians.

- **expression**: Any column or literal expression.

ATAN2(expression1, expression2)

Returns arctangent of expression2/expression1.

- **expression1**: Any column or literal expression.
- **expression2**: Any column or literal expression.

CEIL(expression)

Returns smallest integer not less than the number.

- **expression:** Any column or literal expression.

COS(expression)

Returns cosine.

- **expression:** Any column or literal expression.

DEGREES(expression)

Returns radians to degrees.

- **expression:** Any column or literal expression.

E()

Base of natural logarithms.

EXP(expression)

Returns $e^{\text{expression}}$.

- **expression:** Any column or literal expression.

LN(expression)

Returns log base e.

- **expression:** Any column or literal expression.

LOG(expression)

Returns log base 10.

- **expression:** Any column or literal expression.

FLOOR(expression)

Largest integer not greater than the number.

- **expression:** Any column or literal expression.

PI()

Returns PI.

POWER(expression1, expression2)

Returns $\text{expression1}^{\text{expression2}}$.

- **expression1:** Any column or literal expression.
- **expression2:** Any column or literal expression.

RADIANS(expression)

Returns degrees to radians.

- **expression:** Any column or literal expression.

RANDOM([expression])

Returns pseudo-random number with optional seed.

- **expression:** Any column or literal expression.

ROUND(expression [, integer_digits])

Rounds the value to the given number of integer digits to the right of the decimal point (left if digits is negative). Digits is 0 if not given.

- **expression:** Any column or literal expression.

- **integer_digits:** The number of digits to round to.

SIGN(expression)

Valid values: -1, 0, or 1 for negative, zero, or positive numbers respectively.

- **expression:** Any column or literal expression.

SIN(expression)

Returns sine.

- **expression:** Any column or literal expression.

SQRT(expression)

Returns square root.

- **expression:** Any column or literal expression.

TAN(expression)

Returns tangent.

- **expression:** Any column or literal expression.

TRUNC(expression [, integer_digits])

Truncates the number to the given number of integer digits to the right of the decimal point (left if digits is negative). Digits is 0 if not given.

- **expression:** Any column or literal expression.
- **integer_digits:** The number of digits to truncate.

CONTAINS(column, string_substring)

True if the string contains the substring.

- **column:** Any column or literal expression.
- **string_substring:** The substring to search for.

INITCAP(column)

Converts the string so that the first letter of each word is uppercase and every other letter is lowercase.

- **column:** Any column or literal expression.

LENGTH(column)

Returns length of the string value.

- **column:** Any column or literal expression.

LOWER(column)

Returns lowercase of the string value.

- **column:** Any column or literal expression.

LTRIM(column [, string_chars])

Returns string with all leading chars removed. White space by default.

- **column:** Any column or literal expression.
- **string_chars:** The leading characters to remove.

POSITION(column, string_substring)

Returns the first position of the substring within the string, or -1. The position is zero-based, i.e., the first position is 0.

- **column:** Any column or literal expression.
- **string_substring:** The substring to search for.

REPEAT(column, integer_n)

Returns string formed by repeating expression n times.

- **column**: Any column or literal expression.
- **integer_n**: The number of times to repeat column.

REPLACE(column, string_substring, string_replace [, integer_n])

Returns string with all occurrences of substr replaced with repl. If n is given, at most n replacements are performed.

- **column**: The column expression.
- **string_substring**: The regular expression to match.
- **string_replace**: The value to replace the matched pattern.
- **integer_n**: The maximum number of replacements to make.

RTRIM(column [, string_chars])

Returns string with all trailing chars removed. White space by default.

- **column**: Any column or literal expression.
- **string_chars**: The trailing characters to remove.

SPLIT(column [, string_sep])

Splits the string into an array of substrings separated by string_sep. If string_sep is not given, any combination of white space characters is used.

- **column**: Any column or literal expression.
- **string_sep**: The separator to split column on.

SUBSTR(column, integer_position [, integer_length])

Returns substring from the integer position of the given length, or to the end of the string. The position is zero-based, i.e. the first position is 0. If position is negative, it is counted

from the end of the string; -1 is the last position in the string.

- **column:** Any column or literal expression.
- **integer_position:** The starting position.
- **integer_length:** The total length of the substring to retrieve.

TRIM(column [, string_chars])

Returns string with all leading and trailing chars removed. White space by default.

- **column:** Any column or literal expression.
- **string_chars:** The leading and trailing characters to remove.

UPPER(column)

Returns uppercase of the string value.

- **column:** Any column or literal expression.

TOARRAY(column)

Returns array as follows: MISSING is MISSING; NULL is NULL; Arrays are themselves; All other values are wrapped in an array.

- **column:** Any column expression.

TOATOM(column)

Returns array as follows: MISSING is MISSING; NULL is NULL; Arrays of length 1 are the result of TOATOM() on their single element; Objects of length 1 are the result of TOATOM() on their single value; Booleans, numbers, and strings are themselves; All other values are NULL.

- **column:** Any column expression.

TOBOOLEAN(column)

Returns array as follows: MISSING is MISSING; NULL is NULL; False is false; Numbers +0, -0, and NaN are false; Empty strings, arrays, and objects are false; All other values are true.

- **column:** Any column expression.

TONUMBER(column)

Returns array as follows: MISSING is MISSING; NULL is NULL; False is 0; True is 1; Numbers are themselves; Strings that parse as numbers are those numbers; All other values are NULL.

- **column:** Any column expression.

TOOBJECT(column)

Returns array as follows: MISSING is MISSING; NULL is NULL; Objects are themselves; All other values are the empty object.

- **column:** Any column expression.

TOSTRING(column)

Returns array as follows: MISSING is MISSING; NULL is NULL; False is "false"; True is "true"; Numbers are their string representation; Strings are themselves; All other values are NULL.

- **column:** Any column 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 Name, TotalDue INTO
'csv://c:/Customer.txt' FROM 'Customer' WHERE CustomerId = '12345'");
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 'Customer' IN
'csv://filename=c:/Customer.csv;delimiter=tab' FROM 'Customer' WHERE
CustomerId = '12345'");
System.out.println(stat.getUpdateCount()+" rows affected");
```

INSERT Statements

To create new records, use INSERT statements.

INSERT Syntax

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

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

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

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

```
String cmd = "INSERT INTO Customer (TotalDue) VALUES (?)";
PreparedStatement pstmt = connection.prepareStatement
(cmd,Statement.RETURN_GENERATED_KEYS);
pstmt.setString(1, "John");
```



```
int count = pstmt.executeUpdate();
System.out.println(count+" rows were affected");
ResultSet rs = pstmt.getGeneratedKeys();
while(rs.next()){
    System.out.println(rs.getString("Id"));
}
connection.close();
```

UPDATE Statements

To modify existing records, use UPDATE statements.

Update Syntax

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

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

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

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

DELETE Statements

To delete information from a table, use DELETE statements.

DELETE Syntax

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

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

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

```
Connection connection = DriverManager.getConnection
("jdbc:couchbase:User='myusername';Password='mypassword';Server='http://
couchbase40'");
String cmd = "DELETE FROM Customer WHERE Id = ?";
PreparedStatement pstmt = connection.prepareStatement(cmd);
pstmt.setString(1, "1");
int count=pstmt.executeUpdate();
connection.close();
```

EXECUTE Statements

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

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

Stored Procedure Syntax

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

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

```
| ?  
| <literal>
```

Example Statements

Reference stored procedure inputs by name:

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

Execute a parameterized stored procedure statement:

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

PIVOT and UNPIVOT

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

PIVOT

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

PIVOT Syntax

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

UNPIVOT

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

UNPIVOT Syntax

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

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

Data Model

Overview

Depending upon the connection settings being used, the adapter can present several different mappings between Couchbase entities and relational tables and views. For more details on each of these capabilities, refer to the NoSQL portion of this documentation.

- When connecting to the N1QL query service, the adapter models Couchbase buckets as relational tables. In addition, if `TypeDetectionScheme` is set to `DocType` or `Infer`, the adapter will present different document flavors in each bucket as their own tables.
- When connecting to the Analytics service, the adapter models Couchbase datasets as relational views.
- When connecting with either service, the adapter can expose arrays of data as child tables or views.

Please see the [Automatic Schema Discovery](#) section for more details on how flavor and child tables are exposed. In addition, the `NewChildJoinsMode` connection property is

recommended for workflows that make heavy use of child tables. The documentation for that connection property details the improvements it makes to the adapter data model.

Dataverses, Scopes and Collections

Couchbase has different ways of grouping buckets and datasets depending on the [CouchbaseService](#) and version of Couchbase you are connecting to:

- Couchbase organizes Analytics datasets into groups called dataverses. By default the adapter exposes datasets from all dataverses using compound names like **Default.users** as described in [DataverseSeparator](#). It is important to remember that these compound names must be quoted when used in queries, for example *SELECT * FROM [Default.users]*
- You may also set the [Dataverse](#) property to limit the the adapter to exposing a single dataverse. This disables compound names so view names will not include the dataset.
- When connecting to Couchbase 7 and above, the adapter will use the scope, collection and bucket/dataset name to build table and view names. For example, a table with a name like **crm.accounts.customers** exposes the **customers** collection under the **accounts** scope of the **crm** bucket. These must be quoted the same as other compound names when used in queries, for example *SELECT * FROM [crm.accounts.customers]*

Live Metadata

All of the schemas provided by the adapter are dynamically retrieved from Couchbase, so any changes in the buckets or fields within Couchbase will be reflected in the adapter the next time you connect. You may also issue a reset query to refresh schemas without having to close the connection:

```
RESET SCHEMA CACHE
```

Stored Procedures

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

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

Couchbase Adapter Stored Procedures

Name	Description
AddDocument	Upsert entire JSON documents to Couchbase as-is.
CreateBucket	Creates a new bucket in CouchBase.
CreateCollection	Creates a collection under an existing scope
CreateSchema	Creates a schema definition of a table in Couchbase. Results may change depending of the value of FlattenObjects, FlattenArrays, and TypeDetectionScheme.
CreateScope	Creates a scope under an existing bucket
CreateUserTable	An internal operation used when GenerateSchemaFiles=OnCreate
DeleteBucket	Deletes a bucket (and all its collections and scopes, where supported)
DeleteCollection	Deletes a collection (Couchbase 7 and up)
DeleteScope	Deletes a scope and all its collections (Couchbase 7 and up)
FlushBucket	Removes all documents from a bucket in Couchbase.
ListIndices	Lists all indices available in Couchbase
ManageIndices	Creates/Drops an index in a target bucket in Couchbase.

AddDocument

Upsert entire JSON documents to Couchbase as-is.

Input

Name	Type	Required	Description
BucketName	<i>String</i>	<i>True</i>	The bucket to insert the document into.
SourceTable	<i>String</i>	<i>False</i>	The name of the temp table containing ID and Document columns. Required if no ID is specified.
ID	<i>String</i>	<i>False</i>	The primary key to insert the document under. Required if no SourceTable is specified.
Document	<i>String</i>	<i>False</i>	The JSON text of the document to insert. Required if not SourceTable is specified.

Result Set Columns

Name	Type	Description
RowsAffected	<i>String</i>	The number of rows successfully updated

CreateBucket

Creates a new bucket in CouchBase.

Creating Buckets

Buckets using @AuthType 'none' can be created by specifying only the @Name, @AuthType, @BucketType, and @RamQuotaMB. The @ProxyPort option may also be required, depending upon what version of Couchbase you are connecting to.

```
EXECUTE CreateBucket
  @Name = 'Players',
  @AuthType = 'NONE',
  @BucketType = 'COUCHBASE',
  @RamQuotaMB = 100,
  @ProxyPort = 1234
```

When creating a bucket with @AuthType 'sasl', the @ProxyPort must not be provided, and the @SaslPassword is optional:

```
EXECUTE CreateBucket
  @Name = 'Players',
  @AuthType = 'SASL',
  @BucketType = 'COUCHBASE',
  @RamQuotaMB = 100
```

All other parameters can be used regardless of what @AuthType you provide.

Input

Name	Type	Required	Description
Name	<i>String</i>	<i>True</i>	The name of the bucket to create.
AuthType	<i>String</i>	<i>True</i>	The type of authentication to use can be sasl or none.
BucketType	<i>String</i>	<i>True</i>	The type of the bucket, can be memcached or couchbase.
EvictionPolicy	<i>String</i>	<i>False</i>	What to evict from the cache if the bucket is full, can be fullEviction or valueOnly
FlushEnabled	<i>String</i>	<i>False</i>	Enables or disables flush all support, can be 0 or 1.
ParallelDBAndViewCompaction	<i>String</i>	<i>False</i>	Enables simultaneous

			compactions of the database and the views, can be true or false.
ProxyPort	String	False	The proxy port, must be unused, required if authorization is not SASL.
RamQuotaMB	String	True	The amount of RAM to allocate to the bucket, in megabytes.
ReplicaIndex	String	False	Enables or disables replicate indexes, can be 1 or 0.
ReplicaNumber	String	False	A number between 0 and 3, specifies number of replicas.
SaslPassword	String	False	SASL password, may be provided if the authentication type is SASL.
ThreadsNumber	String	False	A number between 2 and 8, specifies number of concurrent readers/writers.
CompressionMode	String	False	Either Off (no compression), Passive (documents inserted compressed stay compressed) or Active (server can compress any document). On Couchbase Enterprise, Passive is the default.
ConflictResolutionType	String	False	How the server will resolve conflicts between cluster nodes. Either lww (timestamp-based resolution) or seqno (revision ID-based resolution). Defaults to seqno on Couchbase Enterprise.

Result Set Columns

Name	Type	Description
Success	<i>String</i>	Whether or not the bucket was successfully created.

CreateCollection

Creates a collection under an existing scope

Input

Name	Type	Required	Description
Bucket	<i>String</i>	<i>True</i>	The name of the bucket containing the collection.
Scope	<i>String</i>	<i>True</i>	The name of the scope containing the collection.
Name	<i>String</i>	<i>True</i>	The name of the collection to create.

Result Set Columns

Name	Type	Description
Success	<i>Bool</i>	Whether or not the collection was successfully created.

CreateSchema

Creates a schema definition of a table in Couchbase. Results may change depending of the value of FlattenObjects, FlattenArrays, and TypeDetectionScheme.

Input

Name	Type	Required	Accepts Output Streams	Description
TableName	<i>String</i>	<i>True</i>	<i>False</i>	The name of the table.
FileName	<i>String</i>	<i>False</i>	<i>False</i>	The full file path and name of the schema to generate. Ex : 'C:\\Users\\User\\Desktop\\Couchbase\\sheet.rsd'
Overwrite	<i>String</i>	<i>False</i>	<i>False</i>	Will delete any existing schema file for this table.
FileStream	<i>String</i>	<i>False</i>	<i>True</i>	Stream to write the schema to. Only used if FileName is not provided.

Result Set Columns

Name	Type	Description
Result	<i>String</i>	Whether or not the schema was successfully built.
FileData	<i>String</i>	The content of the schema encoded as base64. Only returned if the FileName and FileStream are not provided.

CreateScope

Creates a scope under an existing bucket

Input

Name	Type	Required	Description
Bucket	<i>String</i>	<i>True</i>	The name of the bucket containing the scope.
Name	<i>String</i>	<i>True</i>	The name of the scope to create.

Result Set Columns

Name	Type	Description
Success	<i>Bool</i>	Whether or not the scope was successfully created.

CreateUserTable

An internal operation used when GenerateSchemaFiles=OnCreate

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

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

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

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

Input

Name	Type	Required	Description
CreateNotExist	<i>String</i>	<i>False</i>	Whether an existing table is an error or not
TableName	<i>String</i>	<i>False</i>	The name of the table to create
ColumnNames#	<i>String</i>	<i>False</i>	For each column, its name
ColumnDataTypes#	<i>String</i>	<i>False</i>	For each column, its type
ColumnSizes#	<i>String</i>	<i>False</i>	For each column, its size (ignored)
ColumnScales#	<i>String</i>	<i>False</i>	For each column, its scale (ignored)
ColumnIsNulls#	<i>String</i>	<i>False</i>	For each column, whether it allows NULLs (ignored)
ColumnDefaults#	<i>String</i>	<i>False</i>	For each column, its default value (ignored)
Location	<i>String</i>	<i>False</i>	Where the schema file is generated

Result Set Columns

Name	Type	Description
AffectedTables	<i>String</i>	The number of tables created, either 0 or 1

DeleteBucket

Deletes a bucket (and all its collections and scopes, where supported)

Input

Name	Type	Required	Description
Name	<i>String</i>	<i>True</i>	The name of the bucket to delete.

Result Set Columns

Name	Type	Description
Success	<i>Bool</i>	Whether or not the bucket was successfully deleted.

DeleteCollection

Deletes a collection (Couchbase 7 and up)

Input

Name	Type	Required	Description
Bucket	<i>String</i>	<i>True</i>	The name of the bucket containing the collection.
Scope	<i>String</i>	<i>True</i>	The name of the scope containing the collection.
Name	<i>String</i>	<i>True</i>	The name of the collection to delete.

Result Set Columns

Name	Type	Description
Success	<i>Bool</i>	Whether or not the collection was successfully deleted.

DeleteScope

Deletes a scope and all its collections (Couchbase 7 and up)

Input

Name	Type	Required	Description
Bucket	<i>String</i>	<i>True</i>	The name of the bucket containing the scope.
Name	<i>String</i>	<i>True</i>	The name of the scope to delete.

Result Set Columns

Name	Type	Description
Success	<i>Bool</i>	Whether or not the scope was successfully deleted.

FlushBucket

Removes all documents from a bucket in Couchbase.

Input

Name	Type	Required	Description
Name	<i>String</i>	<i>True</i>	The name of the bucket to delete. Flush must be enabled on this bucket.

Result Set Columns

Name	Type	Description
Success	<i>Bool</i>	Whether or not the bucket was successfully flushed.

ListIndices

Lists all indices available in Couchbase

Result Set Columns

Name	Type	Description
Id	<i>String</i>	The unique index ID
Datastore_id	<i>String</i>	The server hosting the indexed bucket
Namespace_id	<i>String</i>	The pool hosting the indexed bucket
Bucket_id	<i>String</i>	The bucket the index applies to if the index applies to a collection (Couchbase 7 and up). NULL otherwise.
Scope_id	<i>String</i>	The scope the index applies to if the index applies to a collection (Couchbase 7 and up). NULL otherwise.
Keyspace_id	<i>String</i>	The collection the index applies to, if the index applies to a collection (Couchbase 7 and up). The bucket the index applies to otherwise.
Index_key	<i>String</i>	A list of keys participating in the index
Condition	<i>String</i>	The N1QL filter that the index applies to
Is_primary	<i>String</i>	Whether the index is on the primary key
Name	<i>String</i>	The name of the index
State	<i>String</i>	Whether the index is available
Using	<i>String</i>	Whether the index is backed by GSI or a view

ManageIndices

Creates/Drops an index in a target bucket in Couchbase.

Building Indices

An anonymous primary index can be created with these parameters:

```
EXECUTE ManageIndices
  @BucketName = 'Players'
  @Action = 'CREATE'
  @IsPrimary = 'true'
  @IndexType = 'VIEW'
```

This is the same as executing this N1QL:

```
CREATE PRIMARY INDEX ON `Players` USING VIEW
```

A named primary index can be created by specifying an @Name, in addition to the parameters listed above:

```
EXECUTE ManageIndices
  @BucketName = 'Players'
  @Action = 'CREATE'
  @IsPrimary = 'true'
  @Name = 'Players_primary'
  @IndexType = 'VIEW'
```

A secondary index can be created by setting @IsPrimary to false and providing at least one expression.

```
EXECUTE ManageIndices
  @BucketName = 'Players',
  @Action = 'CREATE',
  @IsPrimary = 'false',
  @Name = 'Players_playtime_score',
  @Expressions = '["score", "playtime"]'
```

This is the same as running the following N1QL:

```
CREATE INDEX `Players_playtime_score` ON `Players`(score, playtime)
USING GSI;
```

Multiple nodes and filters can also be provided to generate more complex indices. They must be provided as JSON lists:

```
EXECUTE ManageIndices
  @BucketName = 'Players',
  @Name = 'TopPlayers',
  @Expressions = '["score", "playtime"]',
  @Filter = '["topscore > 1000", "playtime > 600"]',
  @Nodes = '["127.0.0.1:8091", "192.168.0.100:8091"]'
```

This is the same as running the following N1QL:

```
CREATE INDEX `TopPlayers` ON `Players`(score, playtime) WHERE topscore >
1000 AND playtime > 600 USING GSI WITH { "nodes": ["127.0.0.1:8091",
"192.168.0.100:8091"]};
```

Input

Name	Type	Required	Description
BucketName	String	True	The target bucket to create or drop the the index from.
ScopeName	String	False	The target scope to create or drop the index from (Couchbase 7 and up)
CollectionName	String	False	The target collection to create or drop the index from (Couchbase 7 and up)
Action	String	True	Specifies which action to perform on the index, can be Create or Drop.
Expressions	String	False	A list of expressions or functions, encoded as JSON, that the index will be based off of. At least one is required if IsPrimary is set to false

			and the action is Create.
Name	<i>String</i>	<i>False</i>	The name of the index to create or drop, required if IsPrimary is set to false.
IsPrimary	<i>String</i>	<i>False</i>	Specifies whether the index should be a primary index. The default value is <i>true</i> .
Filters	<i>String</i>	<i>False</i>	A list of filters, encoded as JSON, to apply on the index.
IndexType	<i>String</i>	<i>False</i>	The type of index to create, can be GSI or View, only used if the action is Create. The default value is <i>GSI</i> .
ViewName	<i>String</i>	<i>False</i>	Deprecated, included for compatibility only. Does nothing.
Nodes	<i>String</i>	<i>False</i>	A list, encoded as JSON, of nodes to contain the index, must contain the port. Only used if the action is Create.
NumReplica	<i>String</i>	<i>False</i>	How many replicas to create among the index nodes in the cluster.

Result Set Columns

Name	Type	Description
Success	<i>String</i>	Whether or not the index was successfully created or dropped.

Connection String Options

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

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

Authentication

Property	Description
AuthScheme	The type of authentication to use when connecting to Couchbase.
User	The Couchbase user account used to authenticate.
Password	The password used to authenticate the user.
CredentialsFile	Use this property if you need to provide credentials for multiple users or buckets. This file takes priority over other forms of authentication.
Server	The address of the Couchbase server or servers to which you are connecting.
CouchbaseService	Determines the Couchbase service to connect to. Default is N1QL. Available options are N1QL and Analytics.
ConnectionMode	Determines how to connect to the Couchbase server. Must be either Direct or Cloud.
DNSServer	Determines what DNS server to use when retrieving Couchbase Capella information.
N1QLPort	The port for connecting to the Couchbase N1QL Endpoint.
AnalyticsPort	The port for connecting to the Couchbase Analytics Endpoint.
WebConsolePort	The port for connecting to the Couchbase Web Console.

SSL

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

Firewall

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

Proxy

Property	Description

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

Logging

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

Schema

Property	Description
Location	A path to the directory that contains the schema files defining tables, views, and stored procedures.
Dataverse	Which Analytics dataverse to scan when discovering tables.

TypeDetectionScheme	Determines how the provider builds tables and columns from the buckets found in Couchbase.
InferNumSampleValues	The maximum number of values for every field to scan before determining its data type. Applies to Automatic Schema Discovery when TypeDetectionScheme is set to INFER.
InferSampleSize	The maximum number of documents to scan for the columns available in the bucket. Applies to Automatic Schema Discovery when TypeDetectionScheme is set to INFER.
InferSimilarityMetric	Specifies the similarity degree where different schemas will be considered to be the same flavor. Applies to Automatic Schema Discovery when TypeDetectionScheme is set to INFER.
FlexibleSchemas	Whether the provider allows queries to use columns that it has not discovered.
ExposeTTL	Specifies whether document TTL information should be exposed.
NumericStrings	Whether to allow string values to be treated as numbers.
IgnoreChildAggregates	Whether the provider exposes aggregate columns that are also available as child tables. Ignored if TableSupport is not set to Full.
TableSupport	How much effort the provider will put into discovering tables on the Couchbase server.
NewChildJoinsMode	Determines the kind of child table model the provider exposes.

Miscellaneous

Property	Description
AllowJSONParameters	Allows raw JSON to be used in parameters when QueryPassthrough is enabled.

ChildSeparator	The character or characters used to denote child tables.
CreateTableRamQuota	The default RAM quota, in megabytes, to use when inserting buckets via the CREATE TABLE syntax.
DataverseSeparator	The character or characters used to denote Analytics dataverses and scopes/collections.
FlattenArrays	The number of elements to expose as columns from nested arrays. Ignored if IgnoreChildAggregates is enabled.
FlattenObjects	Set FlattenObjects to true to flatten object properties into columns of their own. Otherwise, objects nested in arrays are returned as strings of JSON.
FlavorSeparator	The character or characters used to denote flavors.
GenerateSchemaFiles	Indicates the user preference as to when schemas should be generated and saved.
InsertNullValues	Determines whether an INSERT should include fields that have NULL values.
MaxRows	Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.
Other	These hidden properties are used only in specific use cases.
Pagesize	The maximum number of results to return per page from Couchbase.
PeriodsSeparator	The character or characters used to denote hierarchy.
QueryExecutionTimeout	This sets the server-side timeout for the query, which governs how long Couchbase will execute the query before returning a timeout error.
Readonly	You can use this property to enforce read-only access to Couchbase from the provider.

RowScanDepth	The maximum number of rows to scan to look for the columns available in a table.
StrictComparison	Adjusts how precisely to translate filters on SQL input queries into Couchbase queries. This can be set to a comma-separated list of values, where each value can be one of: date, number, boolean, or string.
Timeout	The value in seconds until the timeout error is thrown, canceling the operation.
TransactionDurability	Specifies how a document must be stored for a transaction to succeed. Specifies whether to use N1QL transactions when executing queries.
TransactionTimeout	This sets the amount of time a transaction may execute before it is timed out by Couchbase.
UpdateNullValues	Determines whether an UPDATE writes NULL values as NULL, or removes them.
UseCollectionsForDDL	Whether to assume that CREATE TABLE statements use collections instead of flavors. Only takes effect when connecting to Couchbase v7+ and GenerateSchemaFiles is set to OnCreate.
UserDefinedViews	A filepath pointing to the JSON configuration file containing your custom views.
UseTransactions	Specifies whether to use N1QL transactions when executing queries.
ValidateJSONParameters	Allows the provider to validate that string parameters are valid JSON before sending the query to Couchbase.

Authentication

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

Property	Description
AuthScheme	The type of authentication to use when connecting to Couchbase.
User	The Couchbase user account used to authenticate.
Password	The password used to authenticate the user.
CredentialsFile	Use this property if you need to provide credentials for multiple users or buckets. This file takes priority over other forms of authentication.
Server	The address of the Couchbase server or servers to which you are connecting.
CouchbaseService	Determines the Couchbase service to connect to. Default is N1QL. Available options are N1QL and Analytics.
ConnectionMode	Determines how to connect to the Couchbase server. Must be either Direct or Cloud.
DNSServer	Determines what DNS server to use when retrieving Couchbase Capella information.
N1QLPort	The port for connecting to the Couchbase N1QL Endpoint.
AnalyticsPort	The port for connecting to the Couchbase Analytics Endpoint.
WebConsolePort	The port for connecting to the Couchbase Web Console.

AuthScheme

The type of authentication to use when connecting to Couchbase.

Possible Values

Auto, Basic, CredentialsFile, SSLCertificate

Data Type

string

Default Value

"Auto"

Remarks

- Auto: This option is deprecated and included only for compatibility.
- Basic: Uses HTTP Basic authentication with [User](#) and [Password](#).
- CredentialsFile: Uses a credentials file. This will require that the [CredentialsFile](#) property be set.
- SSLCertificate: Uses SSL client certificate authentication. Requires that [UseSSL](#) be enabled and that [SSLClientCert](#) and [SSLClientCertType](#) be set.

Note that only Basic authentication is supported when using the "Cloud" [ConnectionMode](#).

User

The Couchbase user account used to authenticate.

Data Type

string

Default Value

""

Remarks

Together with [Password](#), this field is used to authenticate against the Couchbase server.

Password

The password used to authenticate the user.

Data Type

string

Default Value

""

Remarks

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

CredentialsFile

Use this property if you need to provide credentials for multiple users or buckets. This file takes priority over other forms of authentication.

Data Type

string

Default Value

""

Remarks

Use this property if you need to provide credentials for multiple users or buckets. This takes priority over other forms of authentication.

Set CredentialsFile to the path to a file that has the same markup as below:

```
[{"user": "YourUserName1", "pass": "YourPassword1"},  
 {"user": "YourUserName2", "pass": "YourPassword2"}]
```

Server

The address of the Couchbase server or servers to which you are connecting.

Data Type

string

Default Value

""

Remarks

This value can be set to a hostname or an IP address, like "couchbase-server.com" or "1.2.3.4". It can also be set to an HTTP or HTTPS URL, such as "https://couchbase-server.com" or "http://1.2.3.4". If [ConnectionMode](#) is set to Cloud then this should be the hostname of the Couchbase Cloud instance as reported in the control panel.

If the URL form is used, then setting this option will also set the [UseSSL](#) option: if the URL scheme is "https://", then [UseSSL](#) will be set to true, and a URL with "http://" will set [UseSSL](#) to false.

A port value cannot be used as part of this option, so values like "http://couchbase-server.com:8093" are not allowed. Please use [WebConsolePort](#), [N1QLPort](#) and [AnalyticsPort](#).

This value can also accept multiple servers in the above format separated by commas, such as "1.2.3.4, couchbase-server.com". This will allow the adapter to recover the connection in case some of the servers listed are inaccessible.

Note that while the adapter will try to recover the connection as a whole, it may lose individual operations. For example, while a long-running query will fail if the server becomes inaccessible while that query is running, that query can be retried on the same connection and the adapter will execute it on the next active server.

CouchbaseService

Determines the Couchbase service to connect to. Default is N1QL. Available options are N1QL and Analytics.

Possible Values

N1QL, Analytics

Data Type

string

Default Value

"N1QL"

Remarks

Determines the Couchbase service to connect to. Default is N1QL. Available options are N1QL and Analytics

ConnectionMode

Determines how to connect to the Couchbase server. Must be either Direct or Cloud.

Possible Values

Direct, Cloud

Data Type

string

Default Value

"Direct"

Remarks

By default the adapter connects to Couchbase directly using the address given in the [Server](#) option. The [Server](#) must be running the appropriate [CouchbaseService](#) to accept the connection. This will work in most on-premise or basic cloud deployments.

This should be set to Cloud when connecting to Couchbase Capella or a custom deployment that uses service records. These records will allow the adapter to determine the exact Couchbase servers that provide the appropriate [CouchbaseService](#). You must also set the [DNSServer](#) property so that the adapter is able to fetch these service records.

Note that enabling Cloud mode will override these connection properties with the values discovered by contacting the cluster:

- Server
- N1QLPort
- AnalyticsPort

DNSServer

Determines what DNS server to use when retrieving Couchbase Capella information.

Data Type

string

Default Value

""

Remarks

In most cases any public DNS server can be provided here such as the ones provided by OpenDNS, Cloudflare or Google.

If these are not accessible then you will need to use the DNS server configured by your network administrator.

N1QLPort

The port for connecting to the Couchbase N1QL Endpoint.

Data Type

string

Default Value

""

Remarks

This defaults to 8093 when not using SSL, and 18093 when using SSL. See [UseSSL](#).

This port is used for submitting queries when [CouchbaseService](#) is set to N1QL. Any requests to manage indices will also go through this port.

AnalyticsPort

The port for connecting to the Couchbase Analytics Endpoint.

Data Type

string

Default Value

""

Remarks

This defaults to 8095 when not using SSL, and 18095 when using SSL. See [UseSSL](#).

This port is used for submitting queries when [CouchbaseService](#) is set to Analytics.

WebConsolePort

The port for connecting to the Couchbase Web Console.

Data Type

string

Default Value

""

Remarks

This defaults to 8091 when not using SSL, and 18091 when using SSL. See [UseSSL](#).

This port is used for API operations like managing buckets.

SSL

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

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

SSLClientCert

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

Data Type

string

Default Value

""

Remarks

The name of the certificate store for the client certificate.

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

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

Designations of certificate stores are platform-dependent.

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

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

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

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

SSLClientCertType

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

Possible Values

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

Data Type

string

Default Value

"USER"

Remarks

This property can take one of the following values:

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

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

SSLClientCertPassword

The password for the TLS/SSL client certificate.

Data Type

string

Default Value

""

Remarks

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

SSLClientCertSubject

The subject of the TLS/SSL client certificate.

Data Type

string

Default Value

"*"

Remarks

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

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

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

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

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

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

UseSSL

Whether to negotiate TLS/SSL when connecting to the Couchbase server.

Data Type

bool

Default Value

false

Remarks

When this is set to true, the defaults for the following options change:

Property	Plaintext Default	SSL Default
AnalyticsPort	8095	18095
N1QLPort	8093	18093
WebConsolePort	8091	18091

This option should be enabled when connecting to Couchbase Capella because all Capella deployments use SSL by default.

SSLServerCert

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

Data Type

string

Default Value

""

Remarks

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

This property can take the following forms:

Description	Example
A full PEM Certificate (example shortened for brevity)	-----BEGIN CERTIFICATE----- MIICHTCCAe4CAQAwDQYJKoZIhvd.....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)	34e929226ae0819f2ec14b4a3d904f801c
The SHA1 Thumbprint (hex values can also be either space or colon separated)	bb150d

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

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

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

Firewall

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

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

FirewallType

The protocol used by a proxy-based firewall.

Possible Values

NONE, TUNNEL, SOCKS4, SOCKS5

Data Type

string

Default Value

"NONE"

Remarks

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

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

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

FirewallServer

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

Data Type

string

Default Value

""

Remarks

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

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

FirewallPort

The TCP port for a proxy-based firewall.

Data Type

int

Default Value

0

Remarks

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

FirewallUser

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

Data Type

string

Default Value

""

Remarks

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

FirewallPassword

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

Data Type

string

Default Value

""

Remarks

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

Proxy

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

Property	Description
ProxyAutoDetect	This indicates whether to use the system proxy settings or not. This takes precedence over other proxy settings, so you'll need to set ProxyAutoDetect to FALSE in order use custom proxy settings.
ProxyServer	The hostname or IP address of a proxy to route HTTP traffic through.
ProxyPort	The TCP port the ProxyServer proxy is running on.
ProxyAuthScheme	The authentication type to use to authenticate to the ProxyServer proxy.
ProxyUser	A user name to be used to authenticate to the ProxyServer proxy.

ProxyPassword	A password to be used to authenticate to the ProxyServer proxy.
ProxySSLType	The SSL type to use when connecting to the ProxyServer proxy.
ProxyExceptions	A semicolon separated list of destination hostnames or IPs that are exempt from connecting through the ProxyServer .

ProxyAutoDetect

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

Data Type

bool

Default Value

true

Remarks

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

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

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

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

ProxyServer

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

Data Type

string

Default Value

""

Remarks

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

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

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

ProxyPort

The TCP port the ProxyServer proxy is running on.

Data Type

int

Default Value

80

Remarks

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

ProxyAuthScheme

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

Possible Values

BASIC, DIGEST, NONE, NEGOTIATE, NTLM, PROPRIETARY

Data Type

string

Default Value

"BASIC"

Remarks

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

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

The authentication type can be one of the following:

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

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

ProxyUser

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

Data Type

string

Default Value

""

Remarks

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

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

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

ProxyPassword

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

Data Type

string

Default Value

""

Remarks

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

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

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

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

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

ProxySSLType

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

Possible Values

AUTO, ALWAYS, NEVER, TUNNEL

Data Type

string

Default Value

"AUTO"

Remarks

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

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

ProxyExceptions

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

Data Type

string

Default Value

""

Remarks

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

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

Logging

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

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

LogModules

Core modules to be included in the log file.

Data Type

string

Default Value

""

Remarks

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

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

Schema

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

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

Dataverse	Which Analytics dataverse to scan when discovering tables.
TypeDetectionScheme	Determines how the provider builds tables and columns from the buckets found in Couchbase.
InferNumSampleValues	The maximum number of values for every field to scan before determining its data type. Applies to Automatic Schema Discovery when TypeDetectionScheme is set to INFER.
InferSampleSize	The maximum number of documents to scan for the columns available in the bucket. Applies to Automatic Schema Discovery when TypeDetectionScheme is set to INFER.
InferSimilarityMetric	Specifies the similarity degree where different schemas will be considered to be the same flavor. Applies to Automatic Schema Discovery when TypeDetectionScheme is set to INFER.
FlexibleSchemas	Whether the provider allows queries to use columns that it has not discovered.
ExposeTTL	Specifies whether document TTL information should be exposed.
NumericStrings	Whether to allow string values to be treated as numbers.
IgnoreChildAggregates	Whether the provider exposes aggregate columns that are also available as child tables. Ignored if TableSupport is not set to Full.
TableSupport	How much effort the provider will put into discovering tables on the Couchbase server.
NewChildJoinsMode	Determines the kind of child table model the provider exposes.

Location

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

Data Type

string

Default Value

"%APPDATA%\\CData\\Couchbase 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\\Couchbase 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

Dataverse

Which Analytics dataverse to scan when discovering tables.

Data Type

string

Default Value

""

Remarks

This property is empty by default, which means that all dataverses will be scanned and table names will be generated as described in [DataverseSeparator](#).

If you assign this property to a non-blank value, then the adapter will scan only the corresponding dataverse (for example, setting this to "Default" scans the Default dataverse). Since only one dataverse is being scanned, table names will not be prefixed with the dataverse name. It is recommended to set this property to "Default" if you are coming from a previous version of the adapter and need backwards compatibility.

If you are connecting to Couchbase 7.0 or later, this option will be treated as a compound name containing both a dataset and a scope. For example, if you have previously created collections like these:

```
CREATE ANALYTICS SCOPE websites.exempldotcom
CREATE ANALYTICS COLLECTION websites.exempldotcom.traffic ON
examplecom_traffic_bucket
CREATE ANALYTICS COLLECTION websites.exempldotcom.ads ON examplecom_
ads_bucket
```

You would set this option to "websites.exempldotcom".

TypeDetectionScheme

Determines how the provider builds tables and columns from the buckets found in Couchbase.

Data Type

string

Default Value

"DocType"

Remarks

A comma-separated list of the following options:

DocType	This discovers tables by checking at each bucket and looking for
---------	--

different values of the "docType" field in the documents. For example, if the bucket beer-sample contains documents with "docType" = 'brewery' and "docType" = 'beer', this will generate three tables: beer-sample (containing all documents), beer-sample.brewery (containing just breweries) and beer-sample.beer (containing just beers).

Like RowScan, this will scan a sample of the documents in each flavor and determine the data type for each field. [RowScanDepth](#) determines how many documents are scanned from each flavor.

DocType=fieldName	Like DocType, but this scans based off of a field called "fieldName" rather than "docType". "fieldName" must match the field name in Couchbase exactly, including case.
Infer	This uses the N1QL INFER statement to determine what tables and columns exist. This does more flexible flavor detection than DocType, but is only available for Couchbase Enterprise.
RowScan	This reads a sample of documents from a bucket, and heuristically determines the data type. RowScanDepth determines how many documents are scanned. It does not do any flavor detection.
None	This is like RowScan, but will always return columns that have string types instead of the detected type.

InferNumSampleValues

The maximum number of values for every field to scan before determining its data type. Applies to Automatic Schema Discovery when TypeDetectionScheme is set to INFER.

Data Type

string

Default Value

"10"

Remarks

The maximum number of values to scan from every field of the sampled documents before determining the field's data type. This property enables additional configuration of [Automatic Schema Discovery](#) when you are using the Couchbase Infer command -- [TypeDetectionScheme](#) must also be set to Infer to use this property.

InferSampleSize

The maximum number of documents to scan for the columns available in the bucket. Applies to Automatic Schema Discovery when TypeDetectionScheme is set to INFER.

Data Type

string

Default Value

"100"

Remarks

The maximum number of documents to scan for the columns available in the bucket. The Infer command will return column metadata by scanning a random sample of documents of the size specified here.

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

This property enables additional configuration of [Automatic Schema Discovery](#) when you are using the Couchbase Infer command -- [TypeDetectionScheme](#) must also be set to Infer to use this property.

InferSimilarityMetric

Specifies the similarity degree where different schemas will be considered to be the same flavor. Applies to Automatic Schema Discovery when TypeDetectionScheme is set to INFER.

Data Type

string

Default Value

"0.7"

Remarks

This property specifies how similar two schemas must be to be considered to be the same flavor. As an example, consider the following rows:

```
Row 1: ColA, ColB, ColC, ColD
Row 2: ColA, ColB, ColE, ColF
Row 3: ColB, ColF, ColX, ColY
```

You can configure the columns returned for each flavor with different [InferSimilarityMetric](#) values, as in the following examples:

- If you set [InferSimilarityMetric](#) to 1, the adapter will return no flavors.
- If you set [InferSimilarityMetric](#) to 0.5, the adapter will return 2 flavors, Row1 and Row2 making up one, and Row3 making up another.
- If you set [InferSimilarityMetric](#) to 0.25, the adapter will return a single flavor containing all rows.

You can then query document flavors using dot notation, as in the following statement:

```
SELECT * FROM "Items.Technology"
```

This property enables additional configuration of [Automatic Schema Discovery](#) when you are using the Couchbase Infer command -- [TypeDetectionScheme](#) must also be set to Infer to use this property.

FlexibleSchemas

Whether the provider allows queries to use columns that it has not discovered.

Data Type

bool

Default Value

false

Remarks

By default adapter will only allow queries to use columns that it has found during the metadata discovery process (see [TypeDetectionScheme](#) for details). This means that the adapter has the full information for each column it presents, but it also means that fields set on only a few documents may not be exposed. Disabling this option means that the adapter will allow you to write a query with any columns you want. If you use columns in a query that have not been discovered the adapter will assume that they are simple strings.

For example, the adapter uses column type information to automatically convert dates for comparison since Couchbase cannot natively compare dates directly. If the adapter detects that *datecol* is a date field, it can apply the STR_TO_MILLIS conversion automatically:

```
/* SQL */
WHERE datecol < '2020-06-12';
/* N1QL */
WHERE STR_TO_MILLIS(datecol) < STR_TO_MILLIS('2020-06-12');
```

When using undiscovered columns the adapter cannot make this type of conversion for you. You must apply any needed conversions manually to ensure that operations behave the way you want them to.

ExposeTTL

Specifies whether document TTL information should be exposed.

Data Type

bool

Default Value

false

Remarks

By default the adapter does not expose TTL values or consider document TTLs when performing DML operations. Enabling this option exposes TTL values in two ways:

- All tables get a new column called Document.Expiration which contains the TTL value for each document. This column is an integer and returns whatever TTL value is stored in Couchbase directly. This column is read-write on bucket tables and read-only on child tables.
- INSERT and UPDATE will use this field to set TTL values, or to preserve them (for update) when none is provided. Setting the field to either 0 or NULL will remove the TTL from any affected documents.

Note that enabling this features requires that your server be version 6.5.1 or later and that your [CouchbaseService](#) is set to N1QL. If either of these is not the case the adapter will not connect.

NumericStrings

Whether to allow string values to be treated as numbers.

Data Type

bool

Default Value

true

Remarks

By default this property is enabled and the adapter will treat string values as numeric if they all the values it samples during schema detection are numeric. This can cause type errors later on if the field contains non-numeric values in other documents. If this property is disabled then numeric strings are left as strings although other string-based data types like timestamps will still be detected.

For example, the "code" field in the below bucket would be affected by this setting. By default it would be considered an integer but if this property were enabled it would be treated as a string.

```
{ "code": "123", "message": "Please restart your computer" }
{ "code": "456", "message": "Urgent update must be applied" }
```

IgnoreChildAggregates

Whether the provider exposes aggregate columns that are also available as child tables. Ignored if TableSupport is not set to Full.

Data Type

bool

Default Value

false

Remarks

The adapter will expose array fields within a bucket as a separate child table, such as in the Games_scores example described in [Automatic Schema Discovery](#). By default the adapter will also expose these array fields as JSON aggregates on the base table. For example, either of these queries would return information on game scores:

```
/* Return each score as an individual row */
SELECT value FROM Games_scores;
/* Return all scores for each Game as a JSON string */
SELECT scores FROM Games;
```

Since these aggregates are exposed on the base table, they will be generated even when the information they contain is redundant. For example, when performing this join the scores aggregate on Games is populated as well as the value column on Games_scores. Internally this causes two copies of the scores data to be transferred from Couchbase.

```
/* Retrieves score data twice, once for Games.scores and once for Games_
scores.value */
```

```
SELECT * FROM Games INNER JOIN Games_scores ON Games.[Document.Id] =
Games_scores.[Document.Id]
```

This option can be used to prevent the aggregate field from being exposed when the same information is also available from a child table. In the games example, setting this option to true means that the Games table would only expose a primary key column. The only way to retrieve information about scores would be the child table, so score data would only be read once from Couchbase.

```
/* Only exposes Document.Id, not scores */
SELECT * FROM Games;
/* Only retrieves score data once for Games_scores.value */
SELECT * FROM Games INNER JOIN Games_scores ON Games.[Document.Id] =
Games_scores.[Document.Id]
```

Note that this option overrides [FlattenArrays](#), since all data from flattened arrays is also available as child tables. If this option is set then no array flattening is performed, even if [FlattenArrays](#) is set to a value over 0.

TableSupport

How much effort the provider will put into discovering tables on the Couchbase server.

Possible Values

Full, Basic, None

Data Type

string

Default Value

"Full"

Remarks

The available options are:

Full	The adapter will discover the available buckets, and look inside of each of those buckets for child tables. This provides the most flexible way to access nested data, but requires that each bucket on your server have primary indexes.
Basic	The adapter will discover the available buckets, but will not look inside of them for child tables. This is recommended for cases where you either want to reduce the time that schema detection takes, or if your buckets do not have primary indexes.
None	The adapter will only use the schema files found in the Location directory, and will not discover buckets on the server. This option should only be used after you have already created schema files. Using this option without schema files will result in no tables being available.

NewChildJoinsMode

Determines the kind of child table model the provider exposes.

Data Type

string

Default Value

"false"

Remarks

By default the adapter exposes a backwards-compatible data model that is not fully relational. In this mode non-child tables have a primary key called **Document.Id**, but child tables do not have a primary key. Instead they have a column called **Document.Id** which has the same value as the **Document.Id** of the parent row that contains the child row.

For example, a parent table **invoices** containing invoice records may look like this:

Document.Id	customer
1	Adam

2	Beatrice
3	Charlie

And its child **invoices_lineitems** containing line items may look like this:

Document.Id	item
1	laptop
1	keyboard
2	stapler
3	whiteboard
3	markers

This model has several limitations:

- Complex JOIN results may be incorrect. In most cases the adapter can translate a JOIN like *SELECT * FROM invoices INNER JOIN invoices_lineitems ON invoices.[Document.Id] = invoices_lineitems.[Document.Id]* into an UNNEST. But if the JOIN is too complex then both sides are executed separately which can produce incorrect results.
- DML operations on nested child tables are impossible because there is no way to specify what row of the middle child to use. For example, you cannot change rows in a table like **invoices_lineitems_discounts** because there is no way to specify the lineitem that contains the discount you are updating.
- Some environments like SSIS may not be able to operate on child tables at all because they do not have primary keys.

The NewChildJoins data model is fully relational. In this mode non-child tables have the same **Document.Id** as before, but child tables are extended to have both a foreign key and a primary key. The foreign key is called **Document.Parent** and it refers to the **Document.Id** of the row in the parent table that contains the child row. The primary key is called **Document.Id** and it contains a path which uniquely refers to that child row.

For example, the same tables as above would look like this in the NewChildJoins model. **invoices** would be the same:

Document.Id	customer
1	Adam
2	Beatrice
3	Charlie

However, **invoices_lineitems** would have both a primary and foreign key. The primary key contains the ID of the parent row as well as the child row's position in the parent.

Document.Id	Document.Parent	item
1\$1	1	laptop
1\$2	1	keyboard
2\$1	2	stapler
3\$1	3	whiteboard
3\$2	3	markers

This fixes the limitations of the old data model:

- Complex JOIN results are always consistent because they link foreign keys to primary keys. *SELECT * FROM invoices INNER JOIN invoices_lineitems ON invoices.[Document.Id] = invoices_lineitems.[Document.Parent]*
- DML operations on nested child tables are allowed because the **Document.Id** contains all the required information to pick out specific rows, regardless of the table's depth.
- Environments which depend on primary keys can use these tables and generate JOIN queries since the relationships between **Document.Id** and **Document.Parent**

columns are included in the adapter metadata.

Miscellaneous

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

Property	Description
AllowJSONParameters	Allows raw JSON to be used in parameters when QueryPassthrough is enabled.
ChildSeparator	The character or characters used to denote child tables.
CreateTableRamQuota	The default RAM quota, in megabytes, to use when inserting buckets via the CREATE TABLE syntax.
DataverseSeparator	The character or characters used to denote Analytics dataverses and scopes/collections.
FlattenArrays	The number of elements to expose as columns from nested arrays. Ignored if IgnoreChildAggregates is enabled.
FlattenObjects	Set FlattenObjects to true to flatten object properties into columns of their own. Otherwise, objects nested in arrays are returned as strings of JSON.
FlavorSeparator	The character or characters used to denote flavors.
GenerateSchemaFiles	Indicates the user preference as to when schemas should be generated and saved.
InsertNullValues	Determines whether an INSERT should include fields that have NULL values.
MaxRows	Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.

Other	These hidden properties are used only in specific use cases.
Pagesize	The maximum number of results to return per page from Couchbase.
PeriodsSeparator	The character or characters used to denote hierarchy.
QueryExecutionTimeout	This sets the server-side timeout for the query, which governs how long Couchbase will execute the query before returning a timeout error.
Readonly	You can use this property to enforce read-only access to Couchbase from the provider.
RowScanDepth	The maximum number of rows to scan to look for the columns available in a table.
StrictComparison	Adjusts how precisely to translate filters on SQL input queries into Couchbase queries. This can be set to a comma-separated list of values, where each value can be one of: date, number, boolean, or string.
Timeout	The value in seconds until the timeout error is thrown, canceling the operation.
TransactionDurability	Specifies how a document must be stored for a transaction to succeed. Specifies whether to use N1QL transactions when executing queries.
TransactionTimeout	This sets the amount of time a transaction may execute before it is timed out by Couchbase.
UpdateNullValues	Determines whether an UPDATE writes NULL values as NULL, or removes them.
UseCollectionsForDDL	Whether to assume that CREATE TABLE statements use collections instead of flavors. Only takes effect when connecting to Couchbase v7+ and GenerateSchemaFiles is set to OnCreate.
UserDefinedViews	A filepath pointing to the JSON configuration file containing

	your custom views.
UseTransactions	Specifies whether to use N1QL transactions when executing queries.
ValidateJSONParameters	Allows the provider to validate that string parameters are valid JSON before sending the query to Couchbase.

AllowJSONParameters

Allows raw JSON to be used in parameters when QueryPassthrough is enabled.

Data Type

bool

Default Value

false

Remarks

This option affects how string parameters are handled when using direct N1QL and SQL++ queries through [QueryPassthrough](#). For example, consider this query:

```
INSERT INTO `bucket` (KEY, VALUE) VALUES ("1", @x)
```

By default, this option is disabled and string parameters are quoted and escaped into JSON strings. That means that any value can be safely used as a string parameter, but it also means that parameters cannot be used as raw JSON documents:

```
/*
 * If @x is set to: test value " contains quote
 *
 * Result is a valid query
 */
INSERT INTO `bucket` (KEY, VALUE) VALUES ("1", "test value \" contains
quote")
/*
```

```

* If @x is set to: {"a": ["valid", "JSON", "value"]}
*
* Result contains string instead of JSON document
*/
INSERT INTO `bucket` (KEY, VALUE) VALUES ("1", "{\"a\": [\"valid\",
\"JSON\", \"value\"]})

```

When this option is enabled, string parameters are assumed to be valid JSON. This means that raw JSON documents can be used as parameters, but it also means that all simple strings must be escaped:

```

/*
* If @x is set to: test value " contains quote
*
* Result is an invalid query
*/
INSERT INTO `bucket` (KEY, VALUE) VALUES ("1", test value " contains
quote)
/*
* If @x is set to: {"a": ["valid", "JSON", "value"]}
*
* Result is a JSON document
*/
INSERT INTO `bucket` (KEY, VALUE) VALUES ("1", {"a": ["valid", "JSON",
"value"]})

```

Please refer to [ValidateJSONParameters](#) for more details on how parameters are validated when this option is enabled.

ChildSeparator

The character or characters used to denote child tables.

Data Type

string

Default Value

"_"

Remarks

When creating a child table for an array underneath a bucket, the adapter will generate the name of the child table by concatenating the name of the base table, along with this separator and each path element.

For example, if this document were in the bucket "customers", then the child table for the addresses field would be called "customers_addresses".

```
{
  "addresses": [
    {"street": "123 Main St"},
    {"street": "424 Pleasant Ct"},
    {"street": "719 Blue Way"}
  ]
}
```

CreateTableRamQuota

The default RAM quota, in megabytes, to use when inserting buckets via the CREATE TABLE syntax.

Data Type

string

Default Value

"250"

Remarks

The default RAM quota, in megabytes, to use when inserting buckets via the CREATE TABLE syntax.

DataverseSeparator

The character or characters used to denote Analytics dataverses and scopes/collections.

Data Type

string

Default Value

". "

Remarks

When using the Analytics service, the adapter will scan all datasets from all available dataverses. To avoid potential name conflicts, it will include the dataverse name and the dataset name in the generated table name.

By default this is set to ". ", so that if there is a dataset called "users" on the "Default" dataverse, then the table generated will be "Default.users".

This property is also used when generating table names for collections (on both N1QL and Analytics) on Couchbase 7 and later. For example, a bucket called "users" that has two collections called "active" and "inactive" under the "status" scope would be detected as the tables "users.status.active" and "users.status.inactive".

FlattenArrays

The number of elements to expose as columns from nested arrays. Ignored if IgnoreChildAggregates is enabled.

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

FlattenObjects

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

Data Type

bool

Default Value

true

Remarks

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

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

```
address : {
  "street" : "123 Main St.",
  "city"   : "Nowhere",
  "state"  : "NY",
```

```
"zip"      : "12345"
}
```

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

Column Name	Column Value
address.street	123 Main St.
address.city	Nowhere
address.state	NY
address.zip	12345

FlavorSeparator

The character or characters used to denote flavors.

Data Type

string

Default Value

". "

Remarks

When the adapter detects a flavored table, using either a DocType or Infer [TypeDetectionScheme](#), it names flavored tables by concatenating the underlying bucket name, this separator, and the value of the bucket's primary flavor.

For example, if the adapter detects the flavor "docType = 'beer'" on the "beer-sample" bucket, then it will generate the table "beer-sample.beer" which contains only documents in "beer-sample" which have the "beer" doctype.

GenerateSchemaFiles

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

Possible Values

Never, OnUse, OnStart, OnCreate

Data Type

string

Default Value

"Never"

Remarks

GenerateSchemaFiles enables you to save 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.
- OnCreate: A schema file will be generated by when running a CREATE TABLE SQL query.

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

Generate Schemas with SQL

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

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

Generate Schemas on Connection

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

Alternatives to Static Schemas

If your data structures are volatile, consider setting `GenerateSchemaFiles` to **Never** and using dynamic schemas. See [Automatic Schema Discovery](#) for more information about dynamic schemas.

Editing Schemas

Schema files have a simple format that makes them easy to modify. See [Custom Schema Definitions](#) for more information.

InsertNullValues

Determines whether an INSERT should include fields that have NULL values.

Data Type

bool

Default Value

true

Remarks

By default the adapter uses NULL values provided in an INSERT statement and inserts them as JSON null values.

If this option is disabled, SQL NULL values are ignored during an INSERT. In the case of array columns ([FlattenArrays](#) must be set to retrieve these), this means that array indices are shifted over to compensate for the values that have been removed.

MaxRows

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

Data Type

int

Default Value

-1

Remarks

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

Other

These hidden properties are used only in specific use cases.

Data Type

string

Default Value

""

Remarks

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

Specify multiple properties in a semicolon-separated list.

Integration and Formatting

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

Pagesize

The maximum number of results to return per page from Couchbase.

Data Type

int

Default Value

1000

Remarks

The Pagesize property affects the maximum number of results to return per page from Couchbase. Setting a higher value may result in better performance at the cost of additional memory allocated per page consumed.

PeriodsSeparator

The character or characters used to denote hierarchy.

Data Type

string

Default Value

"."

Remarks

When flattening objects and arrays, the adapter will use this value to separate different levels of objects and arrays. For example, if your Couchbase server returns a document like this (and [FlattenObjects](#) is enabled), then the adapter will return the columns "geo.latitude" and "geo.longitude" if the periods separator is set to ".".

```
{
  "geo": {
    "latitude": 35.9132,
    "longitude": -79.0558
  }
}
```

QueryExecutionTimeout

This sets the server-side timeout for the query, which governs how long Couchbase will execute the query before returning a timeout error.

Data Type

string

Default Value

"-1"

Remarks

The default is -1, which disables the timeout. When enabling the timeout, the value must include both an amount and a unit, which can be one of: "ns" (nanoseconds), "us" (microseconds), "ms" (milliseconds), "s" (seconds), "m" (minutes) or "h" (hours). For example, "5m" and "300s" both set timeouts of 5 minutes.

There is a server-side timeout as well called the "index scan timeout", which will override this one if it is lower. By default the index scan timeout is 2 minutes, but it can be changed by setting the "indexer.settings.scan_timeout" property on your Couchbase server.

Readonly

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

Data Type

bool

Default Value

false

Remarks

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

RowScanDepth

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

Data Type

int

Default Value

100

Remarks

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

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

StrictComparison

Adjusts how precisely to translate filters on SQL input queries into Couchbase queries. This can be set to a comma-separated list of values, where each value can be one of: date, number, boolean, or string.

Data Type

string

Default Value

""

Remarks

This option is empty by default, which means that WHERE clauses sent to Couchbase will include extra functions that convert values so that more comparisons work.

For example, leaving the "string" setting out of the list causes arrays to be converted, so that they can be compared with strings:

```
SELECT * FROM Bucket WHERE MyArrayColumn = '[1,2,3]'
```

If set to a value, queries including the relevant types of comparisons will be translated literally. This makes better use of Couchbase's indexes, but means that the types of comparisons must be in a format Couchbase can compare directly.

For example, if "date" is provided as one of the options, then dates must match the format they are stored as in Couchbase since they will not be converted automatically:

```
SELECT * FROM Bucket WHERE MyDateColumn = '2018-10-31T10:00:00';
```

Timeout

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

Data Type

int

Default Value

60

Remarks

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

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

TransactionDurability

Specifies how a document must be stored for a transaction to succeed. Specifies whether to use N1QL transactions when executing queries.

Possible Values

None, Majority, MajorityAndPersistActive, PersistToMajority

Data Type

string

Default Value

"Majority"

Remarks

If [UseTransactions](#) is enabled, then this option can be set to determine when Couchbase will allow writes in transactions to commit. The Couchbase documentation on Durability and Transactions contains the full details, below is a high-level summary.

This option controls requirements on both **quorum** and **persistence**. The quorum may either require no bucket replicas to receive the document (None), or a majority of replicas to have the document (all others). The persistence level requires either that the document be stored in the replica memory (Majority) or on the replica disk (MajorityAndPersistActive, PersistToMajority).

None is only useful if the bucket you are using is not configured for replicas. The other options can be used depending on the required performance and durability tradeoffs.

Persisting to more replicas is slower but provides greater resilience against a node crashing.

TransactionTimeout

This sets the amount of time a transaction may execute before it is timed out by Couchbase.

Data Type

string

Default Value

""

Remarks

If transactions are enabled, then the adapter will default to the server's default transaction timeout setting.

When enabling the timeout, the value must include both an amount and a unit, which can be one of: "ns" (nanoseconds), "us" (microseconds), "ms" (milliseconds), "s" (seconds), "m" (minutes) or "h" (hours). For example, "5m" and "300s" both set timeouts of 5 minutes.

There are also cluster-level and node-level transaction timeouts which override this one if they are smaller. For example, if the node-level timeout is set to a minute then setting this option to "5m" will have no effect.

UpdateNullValues

Determines whether an UPDATE writes NULL values as NULL, or removes them.

Data Type

bool

Default Value

true

Remarks

By default the adapter will use NULL values provided in an UPDATE statement and set the field in Couchbase to NULL.

If this option is disabled SQL NULL values in an UPDATE will cause the adapter to mark the field as MISSING. This removes the field from the object containing it, or if the field is contained in an array (per [FlattenArrays](#)) then that element is set to NULL.

This option should be used with care as the adapter may not detect that the field exists if it is removed from enough documents within a bucket.

UseCollectionsForDDL

Whether to assume that CREATE TABLE statements use collections instead of flavors. Only takes effect when connecting to Couchbase v7+ and GenerateSchemaFiles is set to OnCreate.

Data Type

bool

Default Value

false

Remarks

Normally the adapter will assume that compound table names referenced in a CREATE TABLE statement are flavors. For compatibility, this is still the default with Couchbase v7+ even though flavors are not recommended there.

```
CREATE TABLE [myBucket.myFlavor](
  [Document.Id] VARCHAR PRIMARY KEY,
  docType VARCHAR,
  sometext VARCHAR,
  somenum INT
)
```

Enable this option to assume that CREATE TABLE statements refer to collection instead. In that scenario this query will create the bucket and scope if necessary, before creating the collection and setting a primary index:

```
CREATE TABLE [myBucket.myScope.myCollection](
  [Document.Id] VARCHAR PRIMARY KEY,
  sometext VARCHAR,
  somenum INT
)
```

UserDefinedViews

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

Data Type

string

Default Value

""

Remarks

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

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

This User Defined View configuration file is formatted as follows:

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

For example:

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

```
}
}
```

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

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

UseTransactions

Specifies whether to use N1QL transactions when executing queries.

Possible Values

Never, Always, Explicit

Data Type

string

Default Value

"Never"

Remarks

By default the adapter does not use transactions for compatibility with older versions of Couchbase. All of the other options require a connection to Couchbase 7 or above. The N1QL service must also be enabled using [CouchbaseService](#).

Setting this to **Always** means that all queries will use transactions. An explicit transaction may be created on the connection and queries will use that transaction while it is active. If there is no explicit transaction then queries will use implicit transactions instead.

Setting this to **Explicit** enables support for explicit transactions only. Explicit transactions may be created but if one is not currently active, then statements will not create an implicit transaction.

ValidateJSONParameters

Allows the provider to validate that string parameters are valid JSON before sending the query to Couchbase.

Data Type

bool

Default Value

true

Remarks

When [AllowJSONParameters](#) and [QueryPassthrough](#) are enabled, the query parameters given to the adapter will be treated as raw JSON documents instead of arbitrary string values. This option controls what happens when invalid JSON is given to the adapter in this mode.

When this option is enabled, the adapter will check that all string parameters can be parsed as valid JSON. If any cannot be, an error will be raised and the query will not be run.

When this option is disabled, no check is performed and all string parameter values are substituted into the query directly. This makes executing prepared statements faster, but less safe since invalid N1QL or SQL++ may be sent to the Couchbase.

TIBCO Product Documentation and Support Services

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

How to Access TIBCO Documentation

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

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

Product-Specific Documentation

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

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

TDV Adapter Guides

TDV Data Source Toolkit Guide (Formerly Extensibility Guide)

- **References**

TDV Reference Guide

TDV Application Programming Interface Guide

- **Other**

TDV Business Directory Guide

TDV Discovery Guide

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

How to Contact TIBCO Support

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

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

Release Version Support

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

https://docs.tibco.com/pub/tdv/general/LTS/tdv_LTS_releases.htm.

How to Join TIBCO Community

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

Legal and Third-Party Notices

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

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

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

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

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

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

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

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

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

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

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

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

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