



Schema Editor User Guide

User Guide

Version 4.4.0

April 2023

Copyright

This document was released in April 2023.

Copyright ©2014-2023 Magnitude Software, Inc., an insightsoftware company. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission from Magnitude, Inc.

The information in this document is subject to change without notice. Magnitude, Inc. strives to keep this information accurate but does not warrant that this document is error-free.

Any Magnitude product described herein is licensed exclusively subject to the conditions set forth in your Magnitude license agreement.

Simba, the Simba logo, SimbaEngine, and Simba Technologies are registered trademarks of Simba Technologies Inc. in Canada, the United States and/or other countries. All other trademarks and/or servicemarks are the property of their respective owners.

All other company and product names mentioned herein are used for identification purposes only and may be trademarks or registered trademarks of their respective owners.

Information about the third-party products is contained in a third-party-licenses.txt file that is packaged with the software.

Contact Us

Magnitude Software, Inc.

www.magnitude.com

About This Guide

Purpose

The *Schema Editor User Guide* explains how to run the Schema Editor on all supported platforms, and how to use it to create and modify schema definitions for NoSQL data stores.

Audience

The guide is intended for end users of the Schema Editor who want to create or modify a schema definition for a connection to a NoSQL data store.

Knowledge Prerequisites

To use the Schema Editor, the following knowledge is helpful:

- An understanding of the data structures in the data store
- Familiarity with the data store for which you are modifying schemas
- Familiarity with the platform on which you are using the Schema Editor
- Exposure to SQL

Document Conventions

Italics are used when referring to book and document titles.

Bold is used in procedures for graphical user interface elements that a user clicks and text that a user types.

`Monospace font` indicates commands, source code or contents of text files.

Note:

A text box with a pencil icon indicates a short note appended to a paragraph.

Important:

A text box with an exclamation mark indicates an important comment related to the preceding paragraph.

Table of Contents

About the Schema Editor	5
Using the Schema Editor for ODBC Connections	6
System Requirements	6
Starting the Schema Editor for ODBC Connections	7
Using the Schema Editor for JDBC Connections	9
System Requirements	9
Starting the Schema Editor for JDBC Connections	10
Sampling Data	12
Sampling Data for Couchbase	12
Sampling Data for DocumentDB	14
Sampling Data for DynamoDB	17
Sampling Data for MongoDB	18
Modifying a Schema Definition	21
Working in the Design View	21
Opening a Schema Definition for Editing	22
Hiding and Unhiding Tables	23
Renaming Tables	24
Editing Column Properties	24
Column Properties	25
Adding and Deleting Columns	30
Creating a Columnar View of Array Elements	30
Example of Virtual Tables vs. Columnar Views	31
Previewing Data	34
Saving a Schema Definition	35
Connecting to a Data Store	36
Configuring Logging	37
Third-Party Trademarks	39

About the Schema Editor

NoSQL data stores are able to store "denormalized" data, which does not follow the rules of data typing and structure that apply to traditional relational data. Denormalized data does not have a schema, and can include complex data types such as nested arrays or arrays of differently-typed elements. Because traditional ODBC and JDBC toolsets do not support these data structures, the data needs to be mapped to a relational form.

The Schema Editor is a Java application that enables you to create and modify schema definitions for NoSQL data stores. A schema definition serves as a database layer that maps NoSQL data to ODBC- and JDBC-compatible formats so that you can use standard BI tools to work with the data.

You can use the Schema Editor in conjunction with a Simba connector to create schema definitions. The connector generates the initial schema definition by sampling the data to detect its structure, and then assigning an appropriate data type to each column and generating virtual tables to represent complex data types. You can then use the options in the Schema Editor to manually modify the schema definition and make sure that your application can work with the data.

All Simba connectors that support schema definitions include the Schema Editor as part of the connector installation.

Note:

The AIX version of the Simba MongoDB ODBC Connector does not include the Schema Editor, because the Schema Editor is not compatible with AIX at this time.

Using the Schema Editor for ODBC Connections

The Schema Editor application is packaged differently for ODBC and JDBC connectors. Depending on the platform that you are running and whether you are using a version of the application that is packaged with an ODBC connector or a JDBC connector, different system requirements apply, and different procedures and files are available for starting the application.

The following sections describe how to run the ODBC version of the Schema Editor. For information about the JDBC version, see [Using the Schema Editor for JDBC Connections](#) on page 9.

System Requirements

The ODBC version of the Schema Editor supports ODBC connections only. However, the schema definitions themselves are compatible with both the ODBC and JDBC versions of the Schema Editor, and can be used by both ODBC and JDBC connectors regardless of the Schema Editor that was used to create the definition.

Important:

The Schema Editor does not support Couchbase schema definitions that were created using version 1.1.2 or earlier of the ODBC connector.

Windows

The Schema Editor supports the following versions of Windows (32- and 64-bit editions are supported):

- Windows 10, or 8.1 SP1
- Windows Server 2008 or later

Linux

The Schema Editor supports the following Linux distributions (32- and 64-bit editions are supported):

- Red Hat® Enterprise Linux® (RHEL) 7
- CentOS 7
- SUSE Linux Enterprise Server (SLES) 12 or 15
- Debian 8 or 9
- Ubuntu 18.04 or 20.04

To use the Schema Editor on Linux, you must have the following software installed:

- GTK+ 2.18, 2.20, or 2.24
- One of the following ODBC driver managers:
 - iODBC 3.52.9 or later
 - unixODBC 2.2.14 or later

Note:

If the LD_LIBRARY_PATH environment variable on your machine includes paths for both iODBC and unixODBC, unixODBC takes precedence.

macOS

The Schema Editor supports macOS version 10.14 and 10.15.

To use the Schema Editor on macOS, you must have one of the following ODBC driver managers installed:

- iODBC 3.52.9 or later
- unixODBC 2.2.14 or later

Note:

If the DYLD_LIBRARY_PATH environment variable on your machine includes paths for both iODBC and unixODBC, iODBC takes precedence.

Starting the Schema Editor for ODBC Connections

Depending on which platform you are running, different procedures and files are available for starting the application.

To start the Schema Editor on Windows:

➤ Choose one:

- From the program group of your Simba connector, click the **SchemaEditor** application shortcut.
- Or, open the ODBC Data Source Administrator where you created the DSN for your connection, select the DSN, click **Configure**, and then click **Schema Editor**.

Note:

When you start the Schema Editor from the DSN configuration of a connector, the application automatically uses the connection information specified in your DSN.

To start the Schema Editor on Linux:

1. Navigate to the `[DRIVER_DIR]/Tools/[Bitness]/SchemaEditor` folder, where `[DRIVER_DIR]` is the root directory of the connector and `[Bitness]` is the bitness of your connector installation.
2. Run the `SchemaEditor` application.

To start the Schema Editor on macOS:

1. Navigate to the `[DRIVER_DIR]/Tools/SchemaEditor` folder, where `[DRIVER_DIR]` is the root directory of the connector.
2. Run the `SchemaEditor` application.

The Schema Editor opens on the start page. From there, you can choose to create a new schema definition by sampling data, or modify an existing schema definition stored in a JSON file or a NoSQL data store.

Using the Schema Editor for JDBC Connections

The Schema Editor application is packaged differently for ODBC and JDBC connectors. Depending on the platform that you are running and whether you are using a version of the application that is packaged with an ODBC connector or a JDBC connector, different system requirements apply, and different procedures and files are available for starting the application.

The following sections describe how to run the JDBC version of the Schema Editor. For information about the ODBC version, see [Using the Schema Editor for ODBC Connections](#) on page 6.

System Requirements

The Schema Editor requires Java Runtime Environment version 8, Update 66 (JRE 8u66) or later. The Schema Editor can be used with a Java 8 Runtime Environment that contains the JavaFX libraries.

The JDBC version of the Schema Editor supports JDBC connections only. However, the schema definitions themselves are compatible with both the ODBC and JDBC versions of the Schema Editor, and can be used by both ODBC and JDBC connectors regardless of the Schema Editor that was used to create the definition.

Windows

The Schema Editor supports the following versions of Windows (32- and 64-bit editions are supported):

- Windows Vista, 7, 8, or 10
- Windows Server 2008 or later

Linux

The Schema Editor supports the following Linux distributions (32- and 64-bit editions are supported):

- Red Hat® Enterprise Linux® (RHEL) 5, 6, or 7
- CentOS 5, 6, or 7
- SUSE Linux Enterprise Server (SLES) 11 or 12
- Debian 7 or 8
- Ubuntu 12.04, 14.04, or 16.04

To use the Schema Editor on Linux, you must have GTK+ 2.18, 2.20, or 2.24 installed.

macOS

The Schema Editor supports macOS version 10.9, 10.10, and 10.11.

Starting the Schema Editor for JDBC Connections

Use the `SchemaEditor.jar` file included in your JDBC connector package to start the Schema Editor. The application must be run using JRE 8u66 or later.

To verify the JRE version:

- Open a Command Prompt or Terminal window and then run the following command:

```
java -version
```

This command returns information about the JRE version that your machine runs by default. For example, the following message indicates that your machine runs JRE 8u66:

```
java version "1.8.0_66"  
Java(TM) SE Runtime Environment (build 1.8.0_66-b18)
```

If the command fails to run or an earlier version of Java is returned, then you must specify the full path to the JRE. For example:

```
"C:\Program Files\Java\jre1.8.0_66\bin\java.exe"  
-version
```

To start the Schema Editor:

1. Open a Command Prompt or Terminal window and then navigate to the root directory of the connector.
2. Run one of the following commands, where `[JRE_Path]` is the full path to the `bin` subfolder in the JRE directory:

- On Windows:

```
[JRE_Path]\java -jar SchemaEditor.jar
```

- Or, on Linux or macOS:

```
[JRE_Path]/java -jar SchemaEditor.jar
```

i Note:

If your machine uses JRE 8u66 or later by default, then it is not necessary to include *[JRE_Path]* in the command.

The Schema Editor opens on the start page. From there, you can choose to create a new schema definition by sampling data, or modify an existing schema definition stored in a JSON file or a NoSQL data store.

Sampling Data

You can use the options in the Sample View to specify how the connector samples data, and then generate a schema definition. The connector samples the data in the NoSQL data store in order to detect its structure and determine the data mappings that best support the data.

Depending on the data store that you are working with, different sampling options and workflows may be available. For detailed information about how to sample the data from your data store, see the following:

- [Sampling Data for Couchbase](#) on page 12
- [Sampling Data for DocumentDB](#) on page 14
- [Sampling Data for DynamoDB](#) on page 17
- [Sampling Data for MongoDB](#) on page 18

Sampling Data for Couchbase

You can use the options in the Sample View to specify how the connector samples data from Couchbase to generate a schema definition. The connector samples the data in order to detect its structure and determine the data mappings that best support the data.

Important:

Before generating a schema definition, make sure that primary indexes have been created for all of the buckets in your Couchbase database.

To sample data for Couchbase:

1. Choose one:
 - From the start page, click **Create New**, provide your connection information, and then click **Connect**. For detailed information about how to specify connection information, [Connecting to a Data Store](#) on page 36.
The Sample dialog box opens.
 - Or, from the Design View, click the **Sample View** tab. If you are not already connected to a data store, the Schema Editor prompts you to provide your connection information. For detailed information about how to specify connection information, [Connecting to a Data Store](#) on page 36.
The Schema Editor displays the Sample View.

2. In the **Sampling Count** field, type the maximum number of records that the connector can sample to generate the schema definition. To sample every record in the database, set this option to **0**.

Note:

Typically, sampling a large number of records results in a schema definition that is more accurate and better able to represent all the data in the database. However, the sampling process might take longer than expected when many records are sampled, especially if the database contains complex, nested data structures.

3. In the bottom pane, specify the collections that the connector samples records from by selecting the corresponding check boxes in the **Selected** column. You can select every collection in the database by selecting the check box in the **Selected** column header.

Note:

You can group and sort collections by clicking a column header. For example, to group collections based on the catalogs they belong to and then sort those groupings by catalog name in ascending order, click the **Catalog** column header. To sort the list in descending order, click the header again. To disable sorting, click the header a third time.

4. In each field in the **Type Name Filter** column, type the name of the attribute that the bucket (or collection) uses to specify document types.

When generating a schema definition, the connector creates a base table for each different document type. For example, if you specify `type` as the type name filter for a bucket, and that bucket contains documents that have the `type` values `retail` and `internal`, then the connector creates two tables named "retail" and "internal" in the schema definition.

Important:

Although specifying a type name filter is optional, it is highly recommended that you do so. If you do not provide a type name filter, then the connector uses the bucket name as the type for all the documents in the bucket, and the resulting schema definition is likely to contain errors.

In addition, make sure that you specify the type attribute for every data bucket in the Couchbase Server instance or cluster to which you are connecting. Otherwise, you might encounter issues when working with document types that are not part of the schema definition.

5. To generate the schema, click **Sample**.

The connector samples the data as specified and generates a schema definition, which opens in the Design View in the Schema Editor. If you return to the Sample View, you will see that the check boxes in the Sampled column are selected for all the columns that were included in the sampling process.

Sampling Data for DocumentDB

You can use the options in the Sample View to specify how the connector samples DocumentDB data in order to generate a schema definition. The connector samples the data in order to detect its structure and determine the data mappings that best support the data.

When sampling your DocumentDB data to create a schema definition, the Schema Editor automatically identifies columns, assigns data types, and creates virtual tables for complex data types. To further refine your schema, you can use the following methods:

- **Collection Mapping (default):** The connector samples the data source and creates a single table schema.
- **Table Delimiters:** You can specify one or more attributes from the data, and the connector creates separate tables for each unique value of those selected attributes. Optionally, you can choose specific values for each attribute, and tables are only created for that subset of values. With this option, any unspecified values are excluded from the schema.

Note:

Collection Mapping and Table Delimiters are mutually exclusive. You can only sample a collection using one of these methods.

- **Views:** In addition to the two above methods, you can specify one or more views and the connector represents each as a separate table in the schema. A view is defined using the query language of the data source. For example:

```
SELECT c.Name
FROM AddressBook a
JOIN c IN a.children
WHERE a.id = '123'
ORDER BY a.address.city ASC
```

Important:

If you are using the 32-bit Schema Editor to work with deeply-nested structures or thousands of attributes, you may encounter a Java heap space error. In this case you may need to specify larger initial and maximum values for the memory allocation pool used by the Schema Editor. To do this, launch the Schema Editor from the command line using the following command:

```
java -jar SchemaEditor.jar -Xms[initial]m -Xmxmaximumm
```

where:

- *[initial]* is the initial size, in megabytes, of the memory pool
- *[maximum]* is the maximum size, in megabytes, of the memory pool

For example, to specify an initial memory pool of 512 megabytes and a maximum memory pool of 2048 megabytes, use the following command:

```
java -jar SchemaEditor.jar -Xms512m -Xmx2048m
```



To sample data using Collection Mapping:

1. Launch the Schema Editor.
2. Select your DSN from the list, or enter an appropriate connection string.
3. Click **Connect**.
4. Under Create A New Schema Definition, click **Create New**.
Schema Editor shows a list of databases and collections from the data source.

5. Select the collection or collections you wish to sample from the list.
6. Click **Sample**.



To sample data using Table Delimiters:

1. Launch the Schema Editor.
2. Select your DSN from the list, or enter an appropriate connection string.
3. Click **Connect**.
4. Under Create A New Schema Definition, click **Create New**. Schema Editor shows a list of databases and collections from the data source.
5. Next to the collection you wish to sample, click **Edit** under the Mapping Method column.
6. Set the Mapping Method to **Table Delimiters**.
7. In the **Attributes** field, enter the Attribute you wish to use as a delimiter and press **Enter**. Repeat this process to add all relevant attributes.

You can delete attributes by selecting them and clicking **Delete** , or **Delete All**  to remove all attributes.

Note:

If you do not specify values for the selected attribute or attributes, the Schema Editor creates tables for all unique values of the specified attributes, requiring it to survey the entire data source. For larger data sources this can take a significant amount of time.

8. In the **Values** field, enter the unique value you want used in your schema and press **Enter**. Repeat this process to add all relevant values. Any values that are not specified, and data related to them, will be excluded from the schema.
You can delete values by selecting them and clicking **Delete** , or **Delete All**  to remove all values.
9. Click **OK** to return to the Sample dialog box.
10. To save the information entered here so it can be reused with another data source with the same structure, click **Save**.
11. Click **Sample**.

To sample data using View definitions:

1. Launch the Schema Editor.
2. Select your DSN from the list, or enter an appropriate connection string.

3. Click **Connect**.
4. Under Create A New Schema Definition, click **Create New**.
Schema Editor shows a list of databases and collections from the data source.
5. Select the collection or collections you wish to sample from the list. Set up any Table Delimiters you want to use according to the previous procedure.
6. To add a new view definition, do the following:
 - a. Under the **View Definitions** column for the appropriate collection, click **Add**.
 - b. Click **New**.
 - c. Type a unique name for the View Definition, then click **OK**.
 - d. In the **Edit View** field, type a query that retrieves the information you want to view.
 - e. To add an additional view, click **Add**.
To remove views from the list, use **Delete**.
 - f. To return to the Sample dialog box, click **OK**.
7. Click **Sample**.

The connector samples the data as specified and generates a schema definition, which opens in the Design View in the Schema Editor.

Sampling Data for DynamoDB

You can use the options in the Sample View to specify how the connector samples data from DynamoDB to generate a schema definition. The connector samples the data in order to detect its structure and determine the data mappings that best support the data.

To sample data for DynamoDB:

1. Choose one:
 - From the start page, click **Create New**, provide your connection information, and then click **Connect**. For detailed information about how to specify connection information, [Connecting to a Data Store](#) on page 36.

The Sample dialog box opens.
 - Or, from the Design View, click the **Sample View** tab. If you are not already connected to a data store, the Schema Editor prompts you to provide your connection information. For detailed information about how to specify connection information, [Connecting to a Data Store](#) on page 36.

The Schema Editor displays the Sample View.

2. From the **Sampling Method** drop-down list, select the direction in which the connector reads data during sampling. For example, if you select **Forward**, the connector samples data starting with the first record in the data store, then samples the next record, and so on.
3. In the **Sampling Count** field, type the maximum number of records that the connector can sample to generate the schema definition. To sample every record in the database, set this option to **0**.

Note:

Typically, sampling a large number of records results in a schema definition that is more accurate and better able to represent all the data in the database. However, the sampling process might take longer than expected when many records are sampled, especially if the database contains complex, nested data structures.

4. In the **Sampling Interval** field, type the interval at which the connector samples a record when scanning through the data store. For example, if you set this option to **2**, then the connector samples every second record in the data store.
5. In the bottom pane, specify the collections that the connector samples records from by selecting the corresponding check boxes in the **Selected** column. You can select every collection in the database by selecting the check box in the **Selected** column header.

Note:

You can group and sort collections by clicking a column header. For example, to group collections based on the catalogs they belong to and then sort those groupings by catalog name in ascending order, click the **Catalog** column header. To sort the list in descending order, click the header again. To disable sorting, click the header a third time.

6. To generate the schema, click **Sample**.

The connector samples the data as specified and generates a schema definition, which opens in the Design View in the Schema Editor. If you return to the Sample View, you will see that the check boxes in the Sampled column are selected for all the columns that were included in the sampling process.

Sampling Data for MongoDB

You can use the options in the Sample View to specify how the connector samples data from MongoDB to generate a schema definition. The connector samples the data

in order to detect its structure and determine the data mappings that best support the data.

To sample data for MongoDB:

1. Choose one:

- From the start page, click **Create New**, provide your connection information, and then click **Connect**. For detailed information about how to specify connection information, [Connecting to a Data Store](#) on page 36.

The Sample dialog box opens.

- Or, from the Design View, click the **Sample View** tab. If you are not already connected to a data store, the Schema Editor prompts you to provide your connection information. For detailed information about how to specify connection information, [Connecting to a Data Store](#) on page 36.

The Schema Editor displays the Sample View.

2. From the **Sampling Method** drop-down list, select the sampling method to use. You can sample record sequentially by selecting the **Forward** (starting from the first record), or **Backward** (starting from the last record) options, or at random by selecting the **Random** option.

Note:

The random sampling method is only supported by MongoDB Server 3.2 or higher.

3. In the **Sampling Count** field, type the maximum number of records that the connector can sample to generate the schema definition. To sample every record in the database, set this option to **0**.

Note:

Typically, sampling a large number of records results in a schema definition that is more accurate and better able to represent all the data in the database. However, the sampling process might take longer than expected when many records are sampled, especially if the database contains complex, nested data structures.

4. In the **Sampling Interval** field, type the interval at which the connector samples a record when scanning through the data store. For example, if you set this option to **2**, then the connector samples every second record in the data store. This option is ignored if the Sampling Method is set to Random.

5. In the bottom pane, specify the collections that the connector samples records from by selecting the corresponding check boxes in the **Selected** column. You can select every collection in the database by selecting the check box in the **Selected** column header.

Note:

You can group and sort collections by clicking a column header. For example, to group collections based on the catalogs they belong to and then sort those groupings by catalog name in ascending order, click the **Catalog** column header. To sort the list in descending order, click the header again. To disable sorting, click the header a third time.

6. To filter the data from a collection so that only certain documents are included in the sampling process, type a JSON filter in the corresponding field in the **JSON Filter** column. The connector samples data only from documents in the collection that meet the filter conditions.

Note:

- The JSON filter is used only during the sampling process. It does not affect the data that is returned when you preview data in the Design View.
- For information about the syntax of the **JSON Filter** value, see "db.collection.find()" in the MongoDB Manual: <http://docs.mongodb.org/manual/reference/method/db.collection.find/#db.collection.find>. The JSON filter is the argument for the "query" parameter.
- The value that you type in the JSON Filter field is comparable to a WHERE clause in SQL. For example, to select a row that has the `_id` value T123 from a table named Customers, you would use the SQL statement `SELECT * FROM Customers WHERE _id = "T123"`. In the Schema Editor, to sample data only from documents that contain the `_id` value T123 from a collection named Customers, you would select the collection named Customers and then type `{"_id": { "$oid" : "123 " }}` in the **JSON Filter** field.

7. To generate the schema, click **Sample**.

The connector samples the data as specified and generates a schema definition, which opens in the Design View in the Schema Editor. If you return to the Sample View, you will see that the check boxes in the Sampled column are selected for all the columns that were included in the sampling process.

Modifying a Schema Definition

In some cases, you might need to manually modify a schema definition to make sure that your data displays as desired in ODBC and JDBC applications. Use the options in the Design View to modify your schema definitions as needed.

Note:

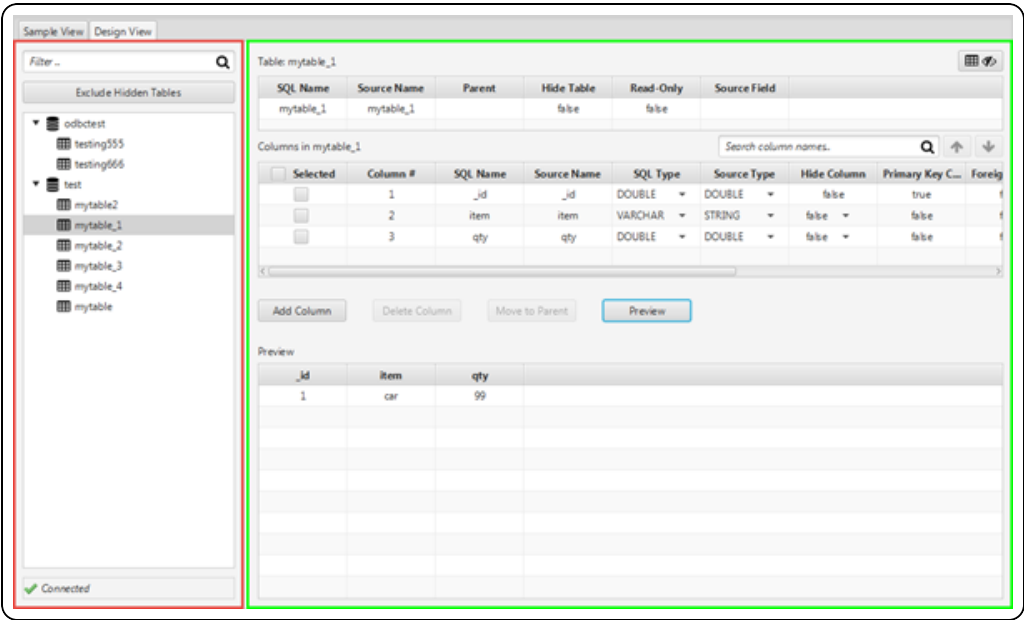
Catalog and schema properties cannot be modified.

For detailed information about how you can modify your schema definition, see the following sections:

- [Opening a Schema Definition for Editing](#) on page 22
- [Hiding and Unhiding Tables](#) on page 23
- [Renaming Tables](#) on page 24
- [Editing Column Properties](#) on page 24
- [Adding and Deleting Columns](#) on page 30
- [Creating a Columnar View of Array Elements](#) on page 30

Working in the Design View






The Design View lets you inspect and modify the schema definition that is currently open in the application. The following image shows the Design View, with the left pane outlined in red and the main pane outlined in green:



Left Pane

The left pane contains a tree structure representing the relational database structure that the NoSQL data is mapped to. You can browse through the tree structure to inspect the database structure created by the schema definition, or select an object to view its properties in the main pane.

Each node in the tree is a database object, such as a catalog, table, or virtual table. The icons beside the object names represent the object type and indicate whether the object is hidden from ODBC and JDBC applications:

-  represents catalogs.
-  represents schemas (Couchbase only).
-  represents tables.
-  represents virtual tables, which are used to renormalize complex data.
-  indicates that the table or virtual table is hidden.

To locate a specific object, you can type the object name in the **Filter** field at the top. The left pane excludes any objects with names that do not contain the value in the **Filter** field.

You can also exclude hidden tables and virtual tables from the left pane by clicking **Exclude Hidden Tables** at the top. However, if a hidden object has an unhidden object nested under it, then the hidden object continues to appear in the pane.

Main Pane

The main pane shows the properties of the selected object and its contents. For example, when you select a table, the top area of the pane shows table properties while the middle area shows column properties.

You can use the main pane to edit column properties and preview the results of your schema definition. For more information, see [Editing Column Properties](#) on page 24 and [Previewing Data](#) on page 34.

Opening a Schema Definition for Editing

Note:

Opening a schema definition from the data store is not supported in DocumentDB.

You can modify schema definitions that are saved in a local JSON file or in a data store. Open a schema definition in the Schema Editor and then use the options in the Design View to modify it as needed.

To open a schema definition from a JSON file:

1. Choose one:
 - In the start page, click **Modify Local Schema Definition**.
 - Or, in the Design View or Sample View, from the menu bar at the top, select **File > Open**.
2. Browse to select your JSON file and then click **Open**. A connection dialog for the data source will be displayed.
3. Enter any required data for connecting to your data source and click **Connect**. For more information see [Connecting to a Data Store](#) on page 36.

The schema definition opens in the Design View.

To open a schema definition from a data store:

- In the start page, click **Modify Database Schema Definition**, provide your connection information, and then click **Connect**. For detailed information about how to specify connection information, [Connecting to a Data Store](#) on page 36.

The schema definition opens in the Design View.

Hiding and Unhiding Tables

You can hide tables so that they cannot be viewed or queried in ODBC and JDBC applications.

To hide a table:

- In the Design View, in the left pane, right-click the table that you want to hide and then click **Hide Table**.

In the left pane, the  icon displays beside the table name to indicate that it is hidden.

To unhide a table:

- In the Design View, in the left pane, right-click the table that you want to unhide and then click **Unhide Table**.

The  icon indicating a hidden table no longer displays beside the table name.

Renaming Tables

You can change the table name that displays in ODBC and JDBC applications (the `SQL Name` table property).

To rename a table:

1. In the Design View, in the left pane, right-click the table that you want to rename and then click **Rename Table**.
2. In the Rename Table dialog box, type a new name in the field and then click **OK**.

The table is shown with the specified name in ODBC and JDBC applications.



Editing Column Properties

When you select a table or virtual table from the left pane of the Design View, the main pane displays its properties as well as the properties of the columns that it contains. You can edit column properties to change the way that the columns are handled in ODBC and JDBC applications.

Note:

Some properties are read-only and cannot be edited. This includes properties such as `Column #`, `Primary Key Column`, and `Foreign Key Column`. For more information, see [Column Properties](#) on page 25.

The center area of the main pane displays column properties. The header contains the property names, and each row contains the property values for one column.

To locate the row for a specific column, in the **Search Column Names** field, type the column name, and then click  or  to browse through the search results. The Schema Editor highlights any rows where the column name contains the search term.

The exact column properties that are available in the main pane differ depending on the specific NoSQL data store that you are working with, but typically include the following:

- `SQL Name`: The column name that is used in SQL operations.
- `Source Name`: The column name that is used in the NoSQL data store.
- `SQL Type`: The SQL data type that the column is mapped to for SQL operations.
- `Source Type`: The NoSQL data type that the column is mapped to in the NoSQL data store.

For more information about specific column properties, see [Column Properties](#) on page 25.

To edit column properties:

1. In the Design View, in the left pane, select the table containing the properties that you want to edit.
2. In the main pane, click the cell containing the value that you want to edit and then do one of the following:
 - If the cell is an editable field, then type your new value and press **ENTER**.
 - If the cell is a drop-down list, then select a value from the list.

Column Properties

General

The following table lists and describes the column properties that are available in the Design View for all NoSQL data stores. The properties are listed in alphabetical order. Only writable properties can be edited.

Property Name	Description	Writable
Column #	A unique identifier for the column in the schema definition.	No
Foreign Key Column	A Boolean indicating whether the column contains foreign key values.	No
Hide Column	A Boolean indicating whether the column displays in ODBC and JDBC applications.	Yes
Primary Key Column	A Boolean indicating whether the column contains primary key values.	No
Source Name	The column name that is used in the NoSQL data store.	Yes
Source Type	The NoSQL data type that the column is mapped to in the NoSQL data store.	Yes
SQL Name	The column name that is used in SQL operations.	Yes

Property Name	Description	Writable
SQL Type	The SQL data type that the column is mapped to for SQL operations.	Yes
User Generated Column	<p>A Boolean indicating whether the column was manually created through the Add Column or Move to Parent options in the Schema Editor. User-generated columns can be deleted.</p> <div> <p>Note:</p> <p>The Move to Parent option is not supported for Couchbase and DocumentDB.</p> </div>	No

Couchbase

The following is an additional column property that is available when you work with Couchbase data stores. Writable properties can be edited.

Property Name	Description	Writable
Precision	<p>The precision for fractional seconds in TIME and TIMESTAMP data. This value indicates the number of digits that exist to the right of the decimal in the seconds component of TIME and TIMESTAMP data.</p> <p>This property does not affect data that is not of type TIME or TIMESTAMP.</p>	Yes

DocumentDB

The following table lists and describes additional column properties that are available when you work with DocumentDB data stores. The properties are listed in alphabetical order. Only writable properties can be edited.

Property Name	Description	Writable
SQL Length	The maximum number of characters that the column can contain.	Yes
Precision	The number of digits in numeric data. This property does not affect non-numeric data directly, but it is used if the data is converted to a numeric type.	Yes
Scale	The number of digits to the right of the decimal in numeric data. This property does not affect non-numeric data directly, but it is used if the data is converted to a numeric type.	Yes

DynamoDB

The following table lists and describes additional column properties that are available when you work with DynamoDB data stores. The properties are listed in alphabetical order. Only writable properties can be edited.

Property Name	Description	Writable
Column Type	<p>An identifier for the type of data that the column contains:</p> <ul style="list-style-type: none"> <code>Data</code> indicates that the column contains raw data values. <code>Index</code> indicates that the column contains index values for lists. <code>Set</code> indicates that the column contains data values from a set. 	Yes

Property Name	Description	Writable
Length	<p>The maximum number of characters that the column can contain.</p> <div> <p>Note:</p> <p>If you connect using Simba DynamoDB JDBC Connector version 1.1.3 or later, the <code>StringColumnMaxLength</code> connection property from the connector overrides this <code>Length</code> property for all String columns.</p> <p>For more information about the <code>StringColumnMaxLength</code> property, see the <i>Simba DynamoDB JDBC Connector Installation and Configuration Guide</i>.</p> </div>	Yes
Precision	<p>The number of digits in numeric data.</p> <p>This property does not affect non-numeric data directly, but it is used if the data is converted to a numeric type.</p>	Yes
Scale	<p>The number of digits to the right of the decimal in numeric data.</p> <p>This property does not affect non-numeric data directly, but it is used if the data is converted to a numeric type.</p>	Yes
Write Permission	<p>A value specifying whether the data in the column is writable:</p> <ul style="list-style-type: none"> <code>Read Only</code> indicates that the data in the column cannot be changed. <code>Write</code> indicates that the data in the column can be changed using DML statements. 	Yes

MongoDB

The following table lists and describes additional column properties that are available when you work with MongoDB data stores. The properties are listed in alphabetical order. Only writable properties can be edited.

Property Name	Description	Writable
Column Size	The maximum number of characters that the column can contain.	Yes
Key Type	<p>An identifier for the type of data that the column contains:</p> <ul style="list-style-type: none"> • <code>DATA_COLUMN</code> indicates that the column contains raw data values. • <code>ID_COLUMN</code> indicates that the column contains <code>_id</code> values, which are primary key values used specifically in MongoDB. • <code>INDEX_COLUMN</code> indicates that the column contains index values for arrays. 	No
Nullable	A Boolean indicating whether the column can contain NULL values.	Yes
Precision	<p>The number of digits in numeric data.</p> <p>This property does not affect non-numeric data directly, but it is used if the data is converted to a numeric type.</p>	Yes
Scale	<p>The number of digits to the right of the decimal in numeric data.</p> <p>This property does not affect non-numeric data directly, but it is used if the data is converted to a numeric type.</p>	Yes

Adding and Deleting Columns

Note:

This feature is not currently available for DocumentDB.

In some cases, due to the configuration settings used during the sampling process, the connector might generate a schema definition that does not contain all the desired table columns. To correct the schema definition, you can use the Schema Editor to add or delete columns. However, you can only delete columns that have the `User Generated Column` property set to `true`.

To add a column:

1. In the Design View, in the left pane, select the table that you want to add a column to.
2. In the main pane, click **Add Column**. A row representing the new column is added to the main pane.
3. Define properties for the new column. For detailed instructions, see [Editing Column Properties](#) on page 24.

To delete a column:

1. In the Design View, in the left pane, select the table that you want to delete a column from.
2. In the main pane, specify the column that you want to delete by selecting the corresponding check box in the **Selected** column.
3. Click **Delete Column**.

The selected column is deleted from the schema definition.

Creating a Columnar View of Array Elements

Note:

- This feature is not supported for Couchbase and DocumentDB.
- The Schema Editor supports columnar views for all forms of array data structures. For example, in addition to creating columnar views of arrays from MongoDB, you can also create columnar views of lists, maps, and sets from DynamoDB.

To support arrays, the Schema Editor creates virtual tables. Each virtual table is a child object of the table that contains the array, and each row in a virtual table contains

an array element. As an alternative, you can create a columnar view of the array elements in the parent table. For an example of how a virtual table representation of an array compares to a columnar view, see [Example of Virtual Tables vs. Columnar Views](#) on page 31.

To create a columnar view of array elements:

1. In the Design View, in the left pane, select the virtual table containing the array elements for which you want to create a columnar view.
2. In the main pane, select the virtual table column that contains the array elements.
3. Click **Move to Parent**.
4. When prompted, in the **Number of Array Elements to Show in Parent** field, type the number of array elements for which you want to create a columnar view, and then click **OK**.

In the parent table, the Schema Editor creates columns that contain the array elements.

Example of Virtual Tables vs. Columnar Views

Note:

- This feature is not supported for Couchbase and DocumentDB.
- The following example uses arrays from MongoDB. Columnar views are supported for all forms of array data structures, so the concepts in this example are also applicable to other array data structures such as lists, maps, and sets from DynamoDB.

Because arrays are not natively supported in standard ODBC and JDBC applications, the Schema Editor renormalizes arrays by mapping them to a supported format.

Array data can be renormalized into virtual tables or columnar views. In virtual tables, each row contains one array element. In columnar views, each column contains one array element. These columns are not created in the database. Rather, they are generated at query time based on the data structure specified in the schema definition.

The following example shows a comparison between a virtual table of arrays and a columnar view of the same arrays.

Consider the following array data in a collection named testscores:

```
{
  "grades": [
```

```

    { "course": "BIOL100", "grade": "A", "score": 96 },
    { "course": "CHEM100", "grade": "A", "score": 88 },
    { "course": "PHYS100", "grade": "B", "score": 75 },
    { "course": "MATH100", "grade": "B", "score": 72 },
    { "course": "ENGL100", "grade": "C", "score": 60 }
  ],
  "name": "John Doe",
  "student_id": "11875445"
}
{
  "grades": [
    { "course": "BIOL100", "grade": "A", "score": 86},
    { "course": "CHEM100", "grade": "B", "score": 78},
    { "course": "PHYS100", "grade": "A", "score": 85},
    { "course": "MATH100", "grade": "A", "score": 90}
  ],
  "name": "Jane Doe",
  "student_id": "11812340"
}

```

By default, your Schema Editor connector renormalizes arrays into virtual tables. The following two tables show the base table and virtual table that would be created in the schema definition.

_id	name	student_id
1111	John Doe	11875445
2222	Jane Doe	11812340

_id	testscores_grades_dim1_idx	course	grade	score
1111	0	BIOL100	A	96
1111	1	CHEM100	A	88
1111	2	PHYS100	B	75

_id	testscores_grades_dim1_idx	course	grade	score
1111	3	MATH100	B	72
1111	4	ENGL100	C	60
2222	0	BIO100	A	86
2222	1	CHEM100	B	78
2222	2	PHYS100	A	85
2222	3	MATH100	A	90

To create a columnar view of the score array, you would select the score column, click **Move to Parent**, and then specify **5** as the number of elements to move. The Schema Editor creates columns in the base table containing up to 5 elements from each score array. The following table shows what the base table looks like after the columnar view is created. The virtual table remains unchanged.

_id	name	student_id	score_0	score_1	score_2	score_3	score_4
1111	John Doe	11875445	96	88	75	72	60
2222	Jane Doe	11812340	86	78	85	90	

Previewing Data

The preview feature shows you the relational tables that are created as a result of the data mapping from the schema definition. Preview your data to see how a table or virtual table would appear in ODBC and JDBC applications.

i Note:

When working with Couchbase data, make sure that the connector is configured to execute queries in SQL instead of N1QL. The Schema Editor uses a SQL query to determine which data to preview.

For information about configuring the connector to use SQL or N1QL, see the descriptions of "Query Mode" or "QueryMode" in the *Installation and Configuration Guide* for your Couchbase connector.

To preview data:

1. In the Design View, in the left pane, select the table that you want to preview.
2. In the main pane, click **Preview**.

The bottom area of the main pane displays up to 10 rows showing how the table appears in ODBC and JDBC applications. The configuration of the database and the connector determine which rows are selected to be included in the preview.

Saving a Schema Definition

After creating or modifying a schema definition, you can preserve your changes by saving the schema definition in a local JSON file or in the NoSQL data store that the schema is used for.

After saving your schema definition, you can configure your connector to use the schema definition when connecting to and working with the data store. For detailed information, see the [Installation and Configuration Guide](#) for your connector.

To save a schema definition:

➤ Choose one:

- To save the schema definition in a local JSON file, select **File > Save**, specify a name and location for the file, and then click **Save**.

Important:

If the data store already contains a schema definition, that schema definition is permanently overwritten when you select **Publish Schema Map**.

- Or, to save the schema definition in the data store that you are connected to, select **Connection > Publish Schema Map**.

Note:

This option is not available for DocumentDB.

If you are not connected to a data store, the Schema Editor prompts you to connect to a data store. For information about how to specify connection information, see [Connecting to a Data Store](#) on page 36.

Connecting to a Data Store

If you are not already connected to a data store when you try to sample data or open a schema definition from a data store, the Schema Editor prompts you to provide your connection information and connect to the data store.

To connect to a data store through ODBC:

1. In the connection prompt, do one of the following:
 - Select **DSN List** and then select a DSN from the drop-down list.
 - Or, select **Connection String** and then type an ODBC connection string in the field.

Note:

For more information about connection string syntax, see the Installation and Configuration Guide for your connector.

2. Click **Connect**.

To connect to a data store through JDBC:

1. In the connection prompt, in the **Connection URI** field, type a JDBC connection URL.

Note:

For more information about connection URL syntax, see the Installation and Configuration Guide for your connector.

2. Click **Connect**.

Configuring Logging

To help troubleshoot issues, you can configure the Schema Editor to run with logging enabled.

Important:

Only enable logging long enough to capture an issue. Logging decreases performance and can consume a large quantity of disk space.

Set the **Log Level** option to enable logging and specify the amount of detail included in log files. The following table lists the logging levels available in the Schema Editor, in order from least verbose to most verbose.

Log Level Value	Description
OFF	Disables all logging.
ERROR	Logs error events that occur while the Schema Editor is running.
INFO	Logs general information that describes the progress of the Schema Editor.
TRACE	Logs all Schema Editor activity.

To enable logging:

1. From the menu bar at the top, select **Help > Settings**.
2. From the **Log Level** drop-down list, select the desired level of information to include in log files.
3. To include your connection information in the log, select the **Log Connection String** check box.
4. In the **Log Location** field, specify a full path for the log file. The log file must be saved in `.log` format.
5. Click **OK**.
6. Restart the Schema Editor to make sure that the new settings take effect.

The Schema Editor produces a log file in the directory specified in the Log Location field.

To disable logging:

1. From the menu bar at the top, select **Help > Settings**.
2. From the **Log Level** drop-down list, select **OFF**.
3. Click **OK**.
4. Restart Schema Editor to make sure that the new settings take effect.

Third-Party Trademarks

Debian is a trademark or registered trademark of Software in the Public Interest, Inc. or its subsidiaries in Canada, United States and/or other countries.

Linux is the registered trademark of Linus Torvalds in Canada, United States and/or other countries.

Mac, macOS, Mac OS, and OS X are trademarks or registered trademarks of Apple, Inc. or its subsidiaries in Canada, United States and/or other countries.

Microsoft, Windows, Windows Azure, Azure, Windows Server, Windows Vista, and the Windows start button are trademarks or registered trademarks of Microsoft Corporation or its subsidiaries in Canada, United States and/or other countries.

Red Hat, Red Hat Enterprise Linux, and CentOS are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in Canada, United States and/or other countries.

SUSE is a trademark or registered trademark of SUSE LLC or its subsidiaries in Canada, United States and/or other countries.

Ubuntu is a trademark or registered trademark of Canonical Ltd. or its subsidiaries in Canada, United States and/or other countries.

Amazon DynamoDB, Amazon, and DynamoDB are trademarks or registered trademarks of Amazon Web Services, Inc. or its subsidiaries in Canada, United States and/or other countries.

Couchbase and Couchbase Server are trademarks or registered trademarks of Couchbase or its subsidiaries in Canada, United States and/or other countries.

MongoDB and Mongo are trademarks or registered trademarks of MongoDB, Inc. or its subsidiaries in Canada, the United States and/or other countries.

All other trademarks are trademarks of their respective owners.