Denodo HBase Custom Wrapper

Revision 20190410

Denodo Inc.: 525 University Avenue, Suite 31, Palo Alto, CA 94301. USA www.denodo.com

CONTENTS

1 INTRODUCTION

The HBase Custom Wrapper enables VDP to perform read operations on an HBase database.

HBase is a column-oriented **NoSQL** database management system that runs on top of Hadoop infrastructure, that allows you to store and process large amounts of data on multiple machines.

As usually happens in the NoSQL database world, HBase does not support SQL queries, therefore it is necessary to use a binary client API (HBase Java client API), to access the database system.

Important

For many scenarios in VDP, accessing HBase data through an SQL abstraction layer

such as **Hive or Impala would be preferable**, as this moves the logic related to the transition between a noSQL, column-oriented system to a relational one over to the Hadoop cluster, which is much closer to where the data actually lives and therefore can often do this **more efficiently and with better performance**.

2 FEATURES

2.1 ARCHITECTURE

The HBase architecture has two main components:

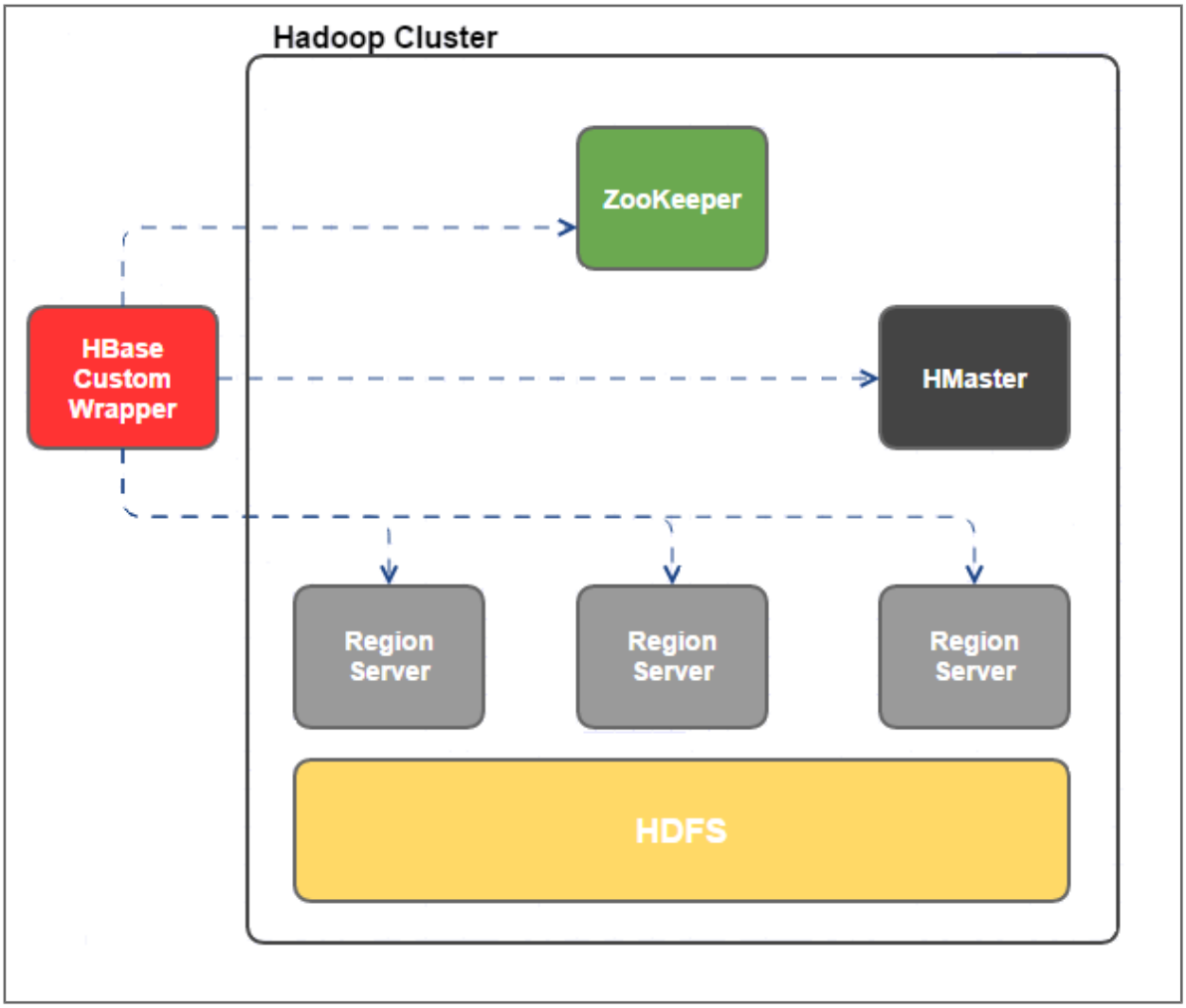
- One **HMaster** node/process that is responsible for coordinating the cluster and executing administrative operations like region allocation and failover, log splitting, and load balancing.

- One or several **HRegionServers** that are responsible for handling a subset of the table's data and I/O requests to the hosting regions, flushing the in-memory data store (MemStore) to HDFS, and splitting and compacting regions.

The wrapper accesses the HBase cluster through the Java API. The ZooKeeper cluster acts as a coordination service for the entire HBase cluster, handling master selection, root region server lookup, node registration, configuration information, hierarchical naming space, and so on.

In terms of connectivity this wrapper needs to be able to access **HMaster** to check if HBase is running. And then it contacts to **ZooKeeper** to learn the location of two special catalog tables named *-ROOT-* and *.META.* within which HBase maintains the current list, state, and location of all regions on the cluster. Once the wrapper has been told where the specific data resides (i.e., in what region), it caches this information and directly access the **HRegionServer** hosting that region.

The next diagram provides a general overview of the system architecture and the interactions between the custom wrapper and HBase:



HBase Custom wrapper and HBase interactions

2.2 DATA MODEL

- **Tables** in HBase are made up of rows and columns.
- **Columns** are grouped into family columns. A column name is made of its column family name and a qualifier `familyName:columnName`.
- **Column families** regroup data of the same nature; they are stored together on the filesystem.
- **Rows** have one row key. They are lexicographically sorted with the lowest order appearing first in a table.
- **Cells** are identified by row, column and version. The cell contents are an uninterpreted array of bytes.

- **Versions** are cells that have the same column and table, every version has associated a timestamp, that it is a long (milliseconds). The HBase version dimension is stored in decreasing order, the latest value is found first. The number of versions that we can fetch when a cell is retrieved is configurable. HBase, by default, returns the latest version.

You can read more about the data model in the next link:
http://hbase.apache.org/book/datamodel.html.

2.3 CAPABILITIES

This wrapper allows only read operations. It can delegate to HBase the following query artifacts and operators:

- Operators: `=`, `<>`, `REGEXP_LIKE`, `LIKE`, `IN`, `IS TRUE`, `IS FALSE`, `IS NULL`, `IS NOT NULL`, `CONTAINS_AND`, `CONTAINS_OR`.
- `AND` operations.
- `OR` operations.
- `StartRow` and `StopRow`: HBase stores rows sorted lexicographically by comparing the key bytes arrays. We can perform a scan to effectively select the start row and the stop row. So, HBase can return all the rows that pass the filter without having to read the whole table. Star row is **inclusive** while stop row is **exclusive**.
- Data types: `Text`, `Long`, `Integer`, `Float`, `Double`, `Boolean`.
- Projection of `row_key` column and column families but not individual columns.

Due to the delegation of the previous operators, the performance of the custom wrapper is better because many operations will be performed by HBase itself, instead of having to rely on VDP in-memory post-filtering.

2.4 LIMITATIONS

There are some limitations which should be taken into account when using this wrapper:

- Read-only. Update and delete operations could be developed in the future, but they are not available as of current versions.
- Row keys are uninterpreted bytes, so we have to be careful if we want to make VDP read data types different to the string one.
- The <, <=, >, => operators cannot be delegated to HBase because its API-driven comparison operations are byte-only based, which makes it unsuitable for comparing numbers.

- The wrapper cannot make the most out of the possibilities of HBase because of the difference in paradigm between this column-oriented NoSQL system and the relational paradigm VDP implements.

3 USAGE

The HBase Custom Wrapper distribution consists of:

- `/conf`: A folder containing a sample `hbase-site.xml` file with properties you might need commented out.
- `/dist`:
  - `denodo-hbase-customwrapper-${version}.jar`. The custom wrapper.
  - `denodo-hbase-customwrapper-${version}-jar-with-dependencies.jar`. The custom wrapper plus its dependencies. This is the wrapper we recommend to use, as it is easier to install in VDP.
- `/doc`: A documentation folder containing this user manual
- `/lib`: All the dependencies required by this wrapper in case you need to use the `denodo-hbase-customwrapper-${version}.jar`

3.1 IMPORTING THE CUSTOM WRAPPER INTO VDP

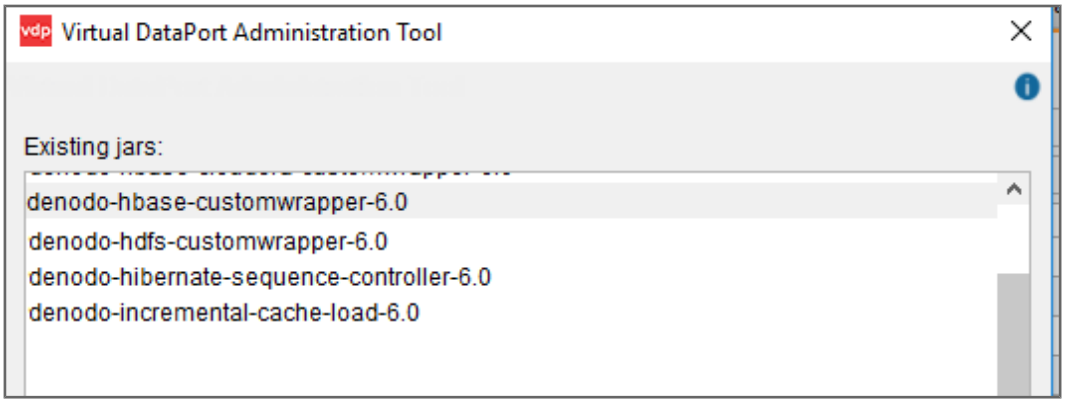
In order to use the HBase Custom Wrapper, first it is necessary to import the jar using the Admin Tool.

From the HBase Custom Wrapper distribution, we will select the `denodo-hbase-customwrapper-${version}-`**`jar-with-dependencies`**`.jar` file and upload it to VDP.

Important

As this wrapper, (the **`jar-with-dependencies`** version), contains the HBase client libraries themselves, increasing the JVM's heap space for VDP Admin Tool is required to avoid a Java heap space when uploading the jar to VDP.

No other jars are required as this one will already contain all the required dependencies.

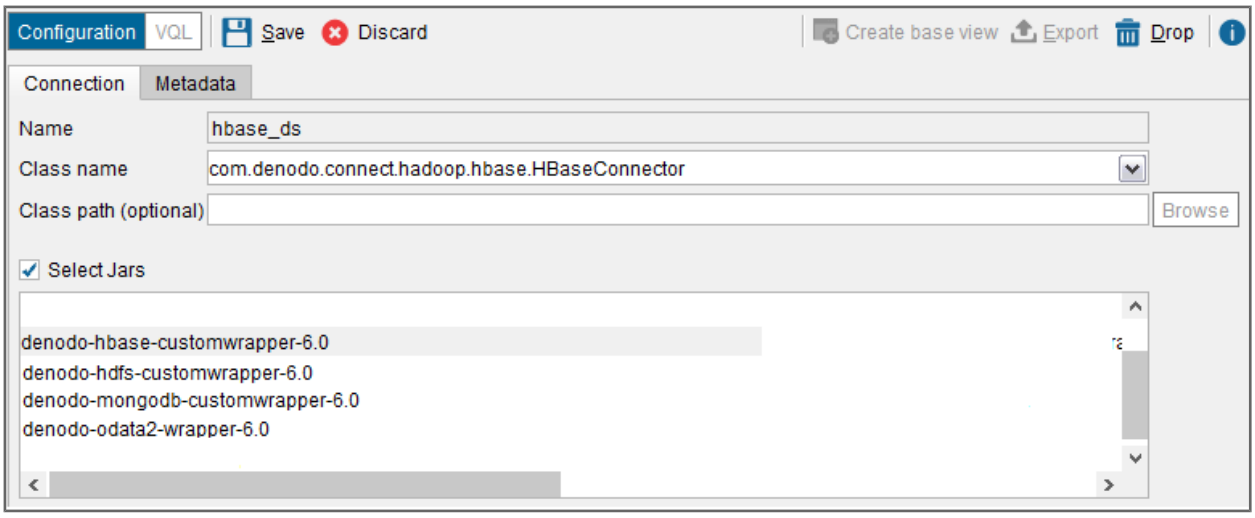


HBase extension in VDP

Creating an HBase Data Source

Once the custom wrapper jar file has been uploaded to VDP using the Admin Tool, we can create new data sources for this custom wrapper --and their corresponding base views-- as usual.

Go to `New → Data Source → Custom`, check '`Select Jars`' and choose the jar file previously uploaded. The custom wrapper is implemented by class `com.denodo.connect.hadoop.hbase.HBaseConnector`, and it will be displayed when selecting the jar in the field `Class name` in this view.



HBase Data Source

3.2 CREATING BASE VIEWS

The base views need the following **mandatory** parameters:

- **`hbaseIP`**: Name or IP address of the host where the Zookeeper server is running.

It can be also a comma-separated list of hosts.

- **`tableName`**: HBase table from which we want to extract the data.
- **`tableMapping`**: A fragment of JSON, giving information about the queried HBase data structure and defining the elements, the column families and columns, that will be projected into the base view.

! Note

If you enter a literal that contains one of the special characters used to indicate interpolation variables `@`, `\`, `^`, `{`, `}` in the **`tableMapping`** parameter, you have to escape these characters with `\`, e.g.:

```
\{
     'title': 'text',
     'body': 'text'
\}
```

An example of a fragment of JSON defining an HBase table:

```
{
      'post': {
          'title': 'text',
          'body': 'text'
      },
      'image': {
          'bodyimage': 'text',
          'header': 'text',
          'active': 'boolean'
      }
  }
```

where `post` and `image` are column families and `title`, `body`, `bodyimage`, `header` and `active` are columns.

There is also a special case that allows retrieving only the `row_key` field of a data set for performance reasons:

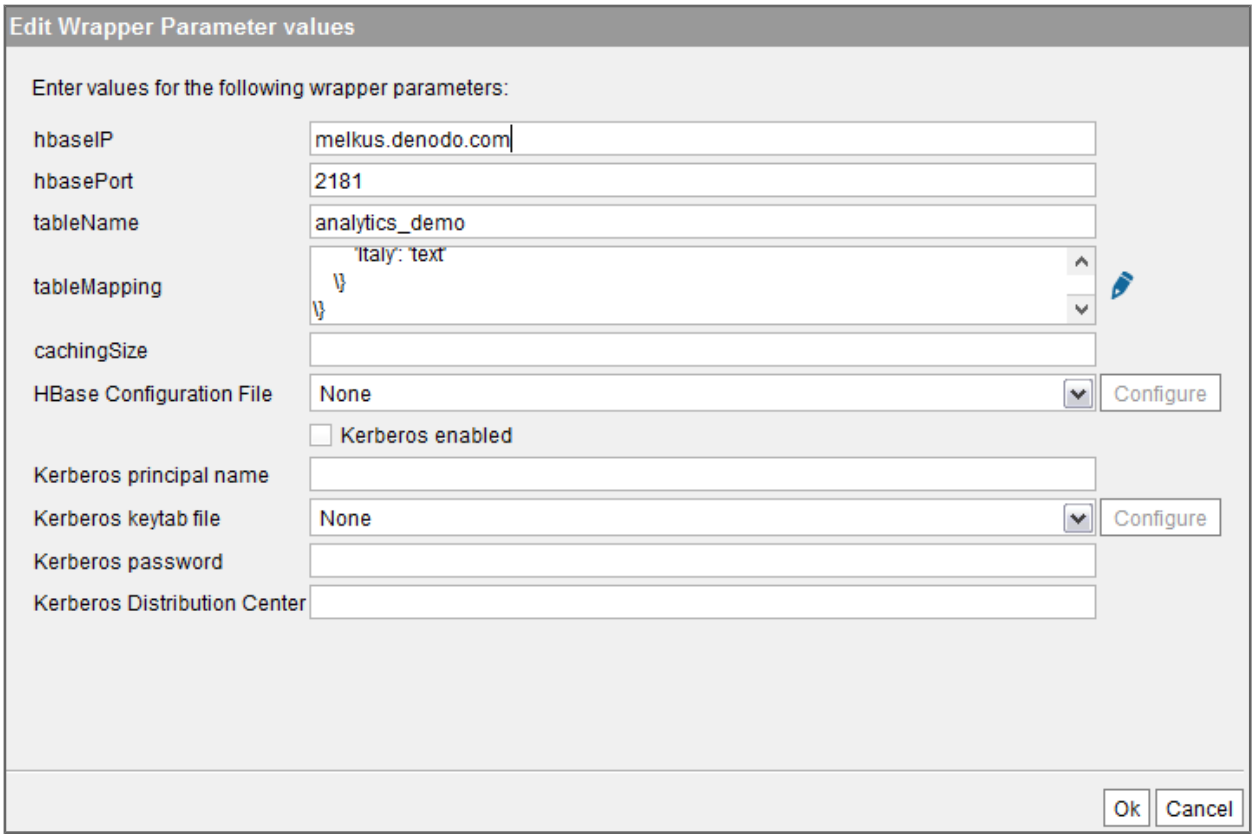
```
{
    'row_key': 'text'
}
```

And two **optional** parameters:

- **`hbasePort`**: ZooKeeper port, default is 2181.
- **`cachingSize`**: the number of rows that a scanner will fetch at once. Higher caching values will enable faster scanners but will use more memory.

This value is important if data in your HBase table is used without any HBase row key based lookups, or when your query looks for wide range scans (wide rowkey lookups).

- **`Hbase Configuration File`**: configuration file that overrides the default HBase parameters (`hbase-site.xml`).



HBase base view edition

3.3 USE CASE

The `blogposts` table has the following structure. Is is represented in JSON as we will use it later for creating the base view in VDP:

```
{
    'post': {
        'title': 'text',
        'body': 'text'
    },
    'image': {
        'bodyimage': 'text',
        'header': 'text',
        'active': 'boolean'
```

```
            }
        }
```

The goal is to retrieve the posts where the title starts with `'Hello World'`.

First we will extract those posts using the HBase Shell and then, we will show how easy it can be with VDP.

3.3.1 Using HBase Shell

Execute this command using the HBase `SingleColumnValueFilter`, that takes a column family, a qualifier, a compare operator and a comparator.

```
hbase(main):001:0> scan 'blogposts', {FILTER =>
 "(SingleColumnValueFilter('post','title',=,'regexstring:Hello World'))"}
```

The following image shows the results of this query:

```
hbase(main):006:0> scan 'blogposts', {FILTER =>
hbase(main):007:1*  "(SingleColumnValueFilter('post','title',=,'regexstring:Hello World'))"}
ROW                           COLUMN+CELL
 post1                        column=image:bodyimage, timestamp=1539190648938, value=image2.jpg
 post1                        column=image:header, timestamp=1539190641937, value=image1.jpg
 post1                        column=post:author, timestamp=1539190625935, value=The Author
 post1                        column=post:body, timestamp=1539190634730, value=This is a blog post
 post1                        column=post:title, timestamp=1539190616009, value=Hello World
1 row(s) in 0.0640 seconds
```

HBase Shell query results

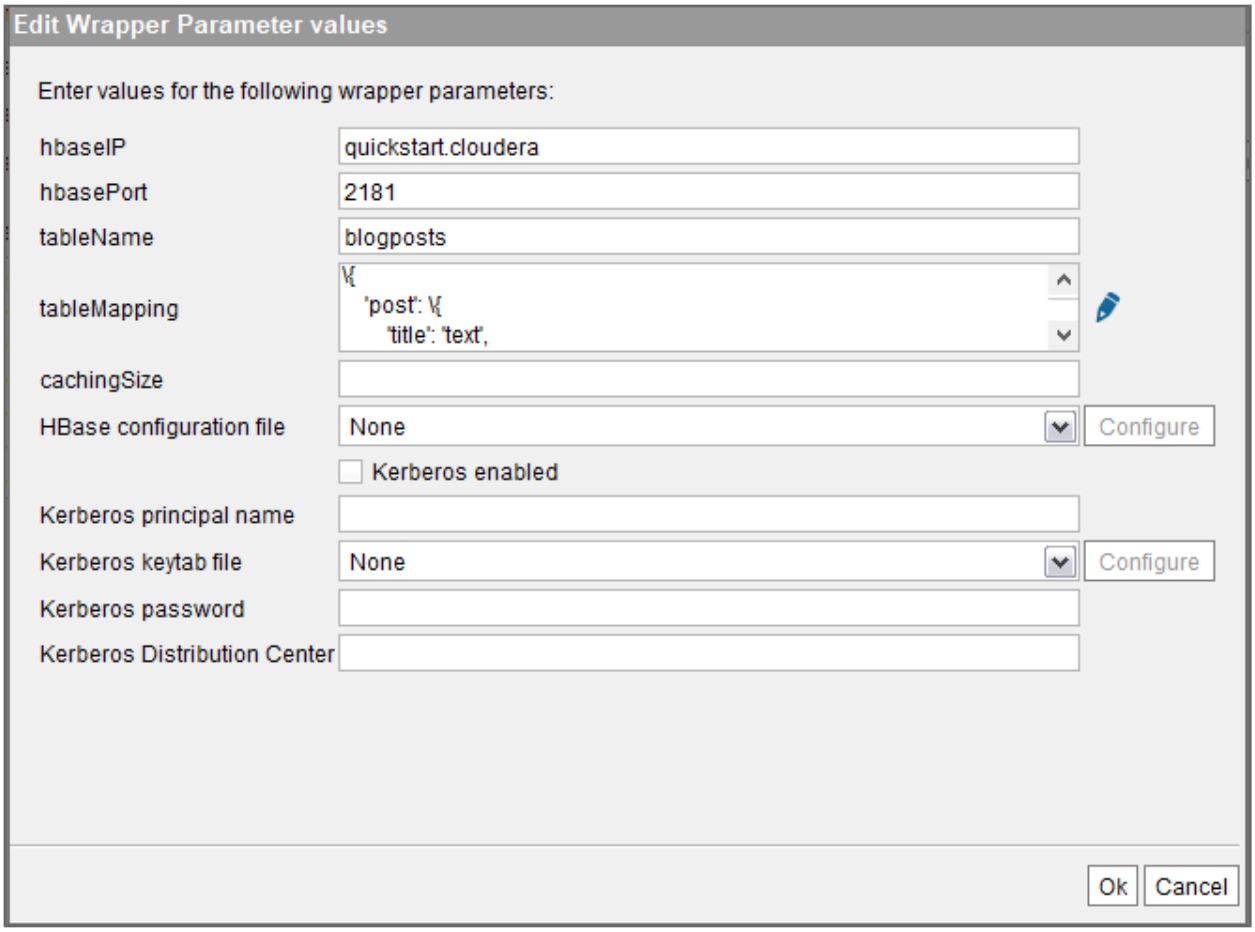
3.3.2 Using VDP

For retrieving the data in VDP you have to create a base view, specifying the table mapping with the following JSON:

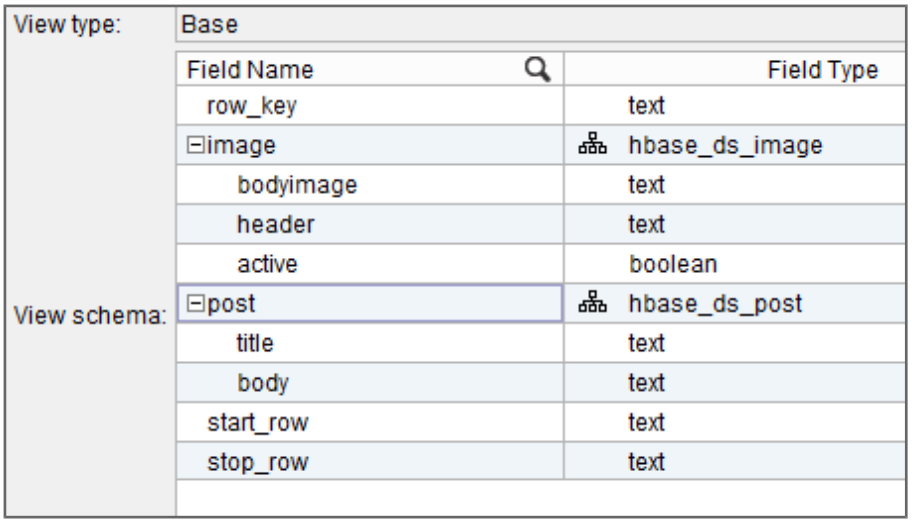
Edit value of 'tableMapping'

```
\{
   'post': \{
      'title': 'text',
      'body': 'text'
   \},
   'image': \{
      'bodyimage': 'text',
      'header': 'text',
      'active': 'boolean'
   \}
\}
```

`blogposts` table mapping



`blogposts` view edition

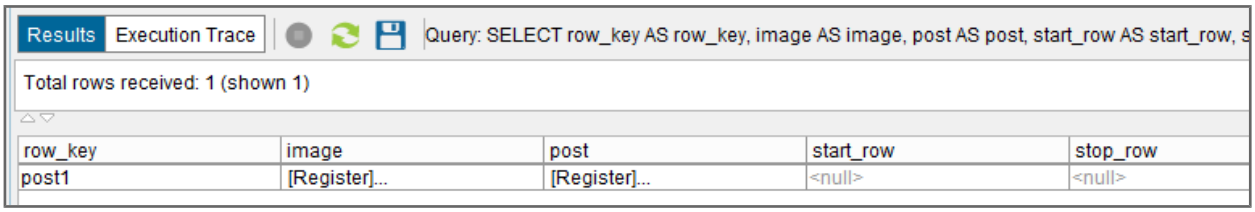


`blogposts` view schema

Once the base view is created, it can be used in VDP just like any other views. Let's see

a VQL sentence equivalent to the above HBase Shell command:

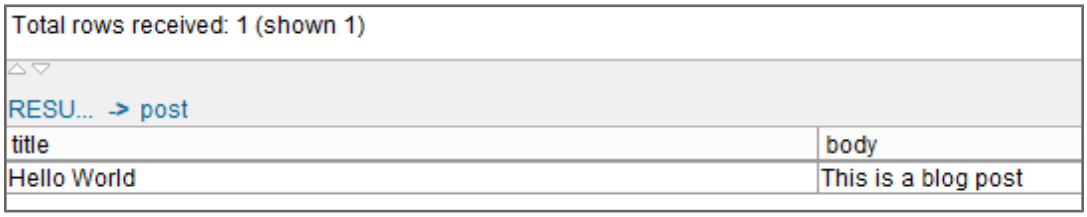
```
SELECT * FROM hbase_ds WHERE (post).title like 'Hello World%'
```



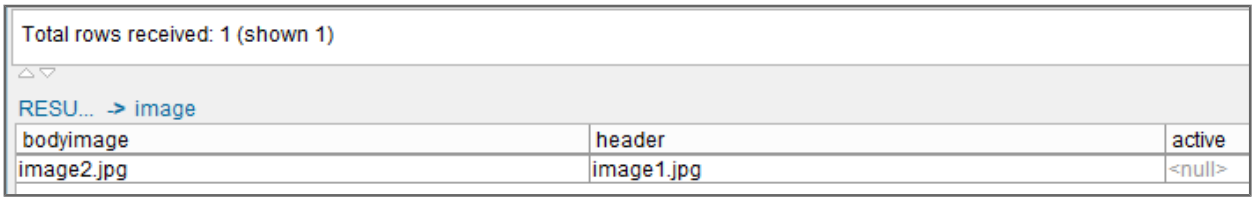
Results with title starting with 'Hello World'

From the `row_key` values, we can check that the same results are obtained as if the HBase Shell had been used instead.

The previous image does not allow to see the array itself, it would require flattening, so here it is for the first returned tuple:



`blogposts post` register



`blogposts image` register

4 SECURE CLUSTER WITH KERBEROS

The configuration required for accessing a Hadoop cluster with Kerberos enabled is the same as the one needed to access HBase and, additionally, the user must supply the Kerberos credentials.

The Kerberos parameters are:

- **`Kerberos enabled`**: Check it when accessing a Hadoop cluster with Kerberos enabled. Required.

- **`Kerberos principal name`**: Kerberos v5 Principal name to access HBase, e.g. `primary/instance@realm`. Optional.

> **! Note**
>
> If you enter a literal that contains one of the special characters used to indicate interpolation variables `@`, `\`, `^`, `{`, `}`, you have to escape these characters with `\`.
>
> E.g if the Kerberos principal name contains `@` you have to enter `\@`.

- **`Kerberos keytab file`**: Keytab file containing the key of the Kerberos principal. Optional.
- **`Kerberos password`**: Password associated with the principal. Optional.
- **`Kerberos Distribution Center`**: Kerberos Key Distribution Center. Optional.

The HBase Custom Wrapper provides **three ways** for accessing a kerberized Hadoop cluster:

1. The client has a valid Kerberos ticket in the **ticket cache** obtained, for example, using the `kinit` command in the Kerberos Client.
   In this case only the `Kerberos enabled` parameter should be checked. The HBase wrapper would use the Kerberos ticket to authenticate itself against the Hadoop cluster.
2. The client does not have a valid Kerberos ticket in the ticket cache. In this case you should provide the `Kerberos principal name` parameter and
   2.1. `Kerberos keytab file` parameter or
   2.2. `Kerberos password` parameter

In all these **three scenarios** the **`krb5.conf` file should be present in the file system**. Below there is an example of the Kerberos configuration file:

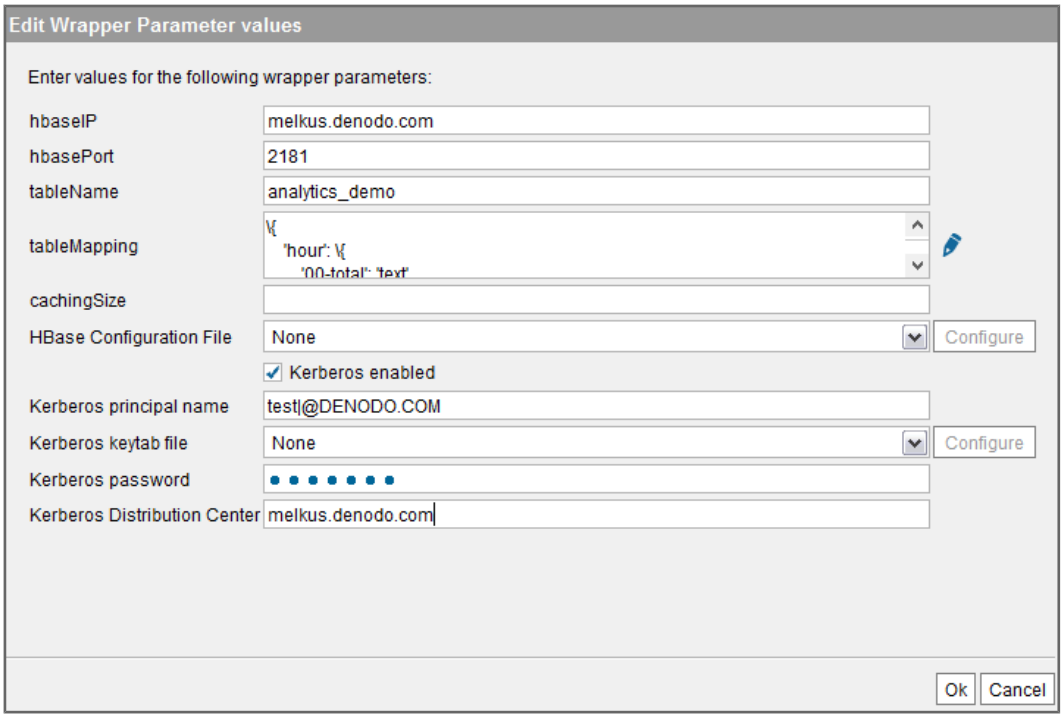
```
[libdefaults]
  renew_lifetime = 7d
  forwardable = true
  default_realm = EXAMPLE.COM
  ticket_lifetime = 24h
  dns_lookup_realm = false
  dns_lookup_kdc = false


[domain_realm]
```

```
  sandbox.hortonworks.com = EXAMPLE.COM
  cloudera = CLOUDERA

[realms]
  EXAMPLE.COM = {
    admin_server = sandbox.hortonworks.com
    kdc = sandbox.hortonworks.com
  }

 CLOUDERA = {
  kdc = quickstart.cloudera
  admin_server = quickstart.cloudera
  max_renewable_life = 7d 0h 0m 0s
  default_principal_flags = +renewable
 }

[logging]
  default = FILE:/var/log/krb5kdc.log
  admin_server = FILE:/var/log/kadmind.log
  kdc = FILE:/var/log/krb5kdc.log
```

The algorithm to locate the `krb5.conf` file is the following:

- If the system property `java.security.krb5.conf` is set, its value is assumed to specify the path and file name.
- If that system property value is not set, then the configuration file is looked for in the directory
  - `<java-home>\lib\security` (Windows)
  - `<java-home>/lib/security` (Solaris and Linux)
- If the file is still not found, then an attempt is made to locate it as follows:
  - `/etc/krb5/krb5.conf` (Solaris)
  - `c:\winnt\krb5.ini` (Windows)
  - `/etc/krb5.conf` (Linux)

There is an **exception**. If you are planning to create HBase views that use the **same Key Distribution Center and the same realm** the `Kerberos Distribution Center` parameter can be provided instead of having the krb5.conf file in the file system:



View edition

5 TROUBLESHOOTING

Symptom

Error message: "`The node /hbase (/hbase-unsecure or /hbase-secure) is not in ZooKeeper. It should have been written by the master. Check the value configured in 'zookeeper.znode.parent'. There could be a mismatch with the one configured in the master.`"

or

Error message: `Received exception with message 'Error accessing HBase: java.lang.NullPointerException`

This exception could mean that there is an error in the authentication, though this might not be the only possible cause. Check the value configured in '`zookeeper.znode.parent`'. There could be a mismatch with the one configured in the HMaster.

Resolution

Edit the configuration file `hbase-site.xml` and change the property `zookeeper.znode.parent` from its value `/hbase` to `/hbase-unsecure` or to `/hbase-secure` when Kerberos is enabled.

The file `hbase-site.xml` could be added as a parameter `HBase configuration file` in the custom wrapper configuration.

Symptom

Error message: "`com.google.protobuf.ServiceException: java.io.IOException: Call to <your domain/your IP:your port> failed on local exception: java.io.EOFException`".

To see the real error message enable Hadoop **debugging** by adding these settings to VDP `log4j2.xml`:

```
<Logger name="org.apache.hadoop" level="debug" />
```

And you will see the following message in VDP log file: "`org.apache.hadoop.ipc.RpcClient - IPC Client (...) exception { exception_class_name: "org.apache.hadoop.security.AccessControlException" stack_trace: "Authentication is required" do_not_retry: false } (...)`"

Resolution

You are trying to connect to a Kerberos-enabled Hadoop cluster. You should configure the custom wrapper accordingly. See **Secure cluster with Kerberos** for configuring Kerberos on this custom wrapper.

Symptom

Error message: "`com.google.protobuf.ServiceException: java.io.IOException: Couldn't setup connection for <Kerberos principal> to <HBase principal>`".

To see the real error message enable Hadoop **debugging** by adding these settings to VDP `log4j2.xml`:

```
<Logger name="org.apache.hadoop" level="debug" />
```

And you will see the following message in VDP log file: `org.apache.hadoop.ipc.RpcClient - (...) Server not found in Kerberos database (7) - UNKNOWN_SERVER)]`

Resolution

Check that `nslookup` is returning the fully qualified hostname of the KDC. If not, modify the `/etc/hosts` of the client machine for the KDC entry to be of the form "`IP address fully.qualified.hostname alias`".

Symptom

Error message: "`org.apache.hadoop.hbase.security.AccessDeniedException: Insufficient permissions for user '<user>' for scanner open on table <table>`"

Resolution

After securing an HBase cluster, if user has not been given appropriate privileges, HBase access will fail with an error "`Insufficient permissions for user`". It is an expected behavior.

Once the cluster has been secured, a user has to authenticate itself to Kerberos by doing a `kinit`. By default, `hbase` is a superuser who was full access and can be used to grant privileges to other users. So you can use `hbase` keytab file to perform a `kinit`. Then, you can login to the HBase Shell and grant privileges to other users.

```
$ sudo -u hbase kinit -kt hbase.keytab
 hbase/cluster.denodo.com@DENODO.COM

$ sudo -u hbase hbase shell
18/10/18 09:15:23 INFO Configuration.deprecation: hadoop.native.lib
is deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.0-cdh5.13.0, rUnknown, Wed Oct 4 11:16:18 PDT 2017

hbase(main):001:0> grant 'test','R'
0 row(s) in 5.4110 seconds
```

6 APPENDICES

6.1 HOW TO USE THE HADOOP VENDOR'S CLIENT LIBRARIES

In some cases, it is advisable to use the libraries of the Hadoop vendor you are connecting to (Cloudera, Hortonworks, ...), instead of the Apache Hadoop libraries distributed in this custom wrapper.

In order to use the Hadoop vendor libraries there is **no need to import the HBase Custom Wrapper** as an extension as it is explained in the **Importing the custom wrapper into VDP** section.

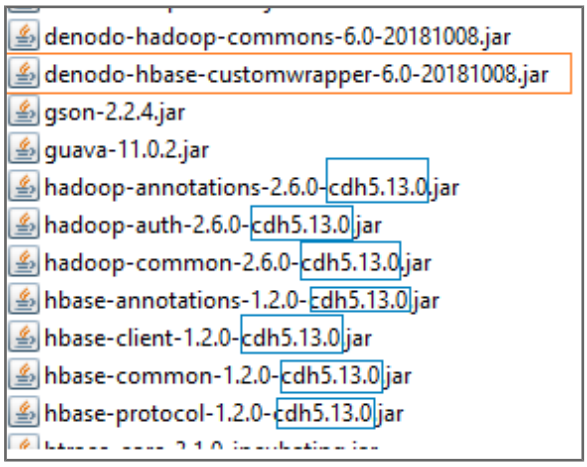
You have to create the custom data sources using the **'`Classpath`' parameter** instead of the '`Select Jars`' option.

Click `Browse` to select the directory containing the required dependencies for this custom wrapper, that is:

- The `denodo-hbase-customwrapper-${version}.jar` file of the `dist` directory of this custom wrapper distribution (highlighted in orange in the image below).
- The contents of the `lib` directory of this custom wrapper distribution, replacing the Apache Hadoop libraries with the vendor specific ones (highlighted in blue in the image below, the suffix indicating that they are Cloudera jars).

Here you can find the libraries for Cloudera and Hortonworks Hadoop distributions:

- Cloudera repository: https://repository.cloudera.com/artifactory/cloudera-repos/org/apache/hbase/hbase/
- Hortonworks repository: http://repo.hortonworks.com/content/repositories/releases/org/apache/hbase/hbase/



C:\Work\hbase-lib directory

HBase Data Source

! Note

When clicking **Browse**, you will browse the file system of the host where the Server is running and not where the Administration Tool is running.

6.2 HOW TO CONNECT TO MAPR DATABASE BINARY TABLES

From MapR documentation: "*MapR Database is an enterprise-grade, high-performance, NoSQL database management system. You can use it for real-time, operational analytics capabilities.*"

As the HBase API provides access to MapR Database binary tables, you can use the HBase Custom Wrapper to connect to MapR Database **binary tables**. This section explains how to do that.

6.2.1 Install MapR Client

To connect to the MapR cluster you need to install the MapR Client on your client machine (where the VDP server is running):

- Verify that the operating system on the machine where you plan to install the MapR Client is supported, see MapR Client Support Matrix.
- Obtain the MapR packages at https://package.mapr.com/releases/ and complete the installation steps explained in https://mapr.com/docs/home/AdvancedInstallation/SettingUptheClient-install-mapr-client.html. These steps are highly dependent on the operating system.

Set `$MAPR_HOME` environment variable to the directory where MapR client was installed. If `MAPR_HOME` environment variable is not defined `/opt/mapr` is the default path.

6.2.2 Copy `mapr-clusters.conf` file

Copy `mapr-clusters.conf` from the MapR cluster to the `$MAPR_HOME/conf` folder in the VDP machine.

```
demo.mapr.com secure=true maprdemo:7222
```

6.2.3 Generate MapR ticket (secure clusters only)

Every user who wants to access a secure cluster must have a MapR ticket (`maprticket_<username>`) in the temporary directory (the default location).

Use the `$MAPR_HOME/maprlogin` command line tool to generate one:

```
C:\opt\mapr\bin>maprlogin.bat password -user mapr

[Password for user 'mapr' at cluster 'demo.mapr.com': ]
MapR credentials of user 'mapr' for cluster 'demo.mapr.com' are written
to 'C:\Users\<username>\AppData\Local\Temp/maprticket_<username>'
```

! Note

If you get an error like

```
java.security.InvalidAlgorithmParameterException: the trustAnchors
parameter must be non-empty when executing maprlogin
```

you need to specify a truststore before executing the `maprlogin` command.

For this, you can copy the `/opt/mapr/ssl_truststore` from MapR Cluster to `$MAPR_HOME/conf` directory in the local machine.

6.2.4 Add JVM option

Add the parameter `-Dmapr.library.flatclass` to the VDP Server JVM options.

VDP Server JVM options

Otherwise, VDP will throw the exception `java.lang.UnsatisfiedLinkError` from `JNISecurity.SetParsingDone()` while executing the HBase Custom Wrapper.

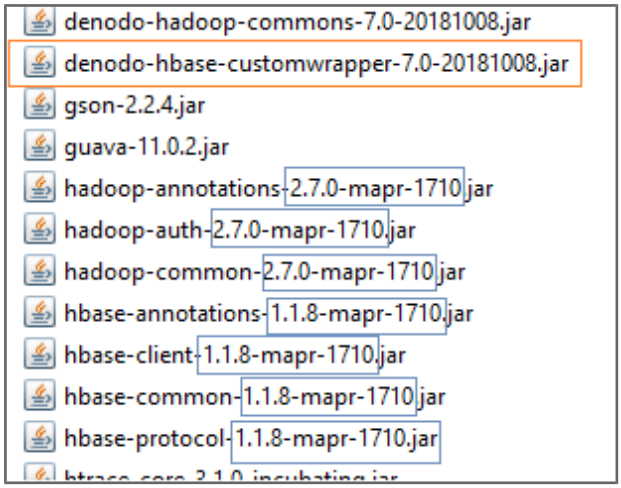
6.2.5 Create custom data source

In order to use the MapR vendor libraries you should **not import the HBase Custom Wrapper** into Denodo.

You have to create the custom data source using the **'Classpath' parameter** instead of the 'Select Jars' option. Click Browse to select the directory containing the required dependencies for this custom wrapper:

- The denodo-hbase-customwrapper-${version}.jar file of the dist directory of this custom wrapper distribution (highlighted in orange in the image below).

- The contents of the lib directory of this custom wrapper distribution, replacing the Apache Hadoop libraries with the **MapR** ones (highlighted in blue in the image below, the suffix indicates that they are MapR jars).



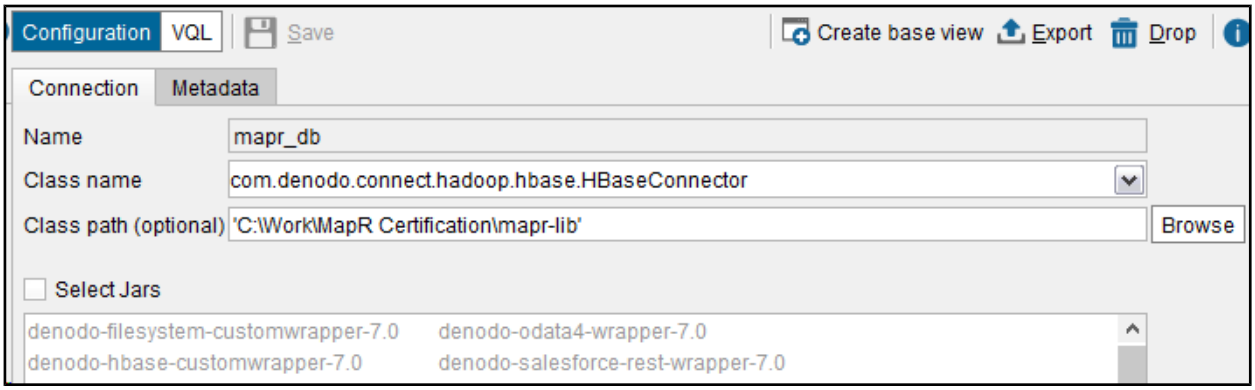
MapR data source dependencies

The MapR Maven repository is located at http://repository.mapr.com/maven/. The name of the JAR files that you must use contains the version of Hadoop, HBase, Zookeeper and the version of MapR that you are using.

The dependencies required for the MapR data source are:

- hadoop-xxx-<hadoop_version>-<mapr_version>
- hbase-xxx-<hadoop_version>-<mapr_version>
- maprfs-<mapr_version>
  As MapRClient native library is bundled in maprfs-<MAPR_VERSION> jar you should use the maprfs jar that comes with the Mapr Client, previously installed, as the library is dependent on the operating system.

- mapr-hbase-<mapr_version>
- zookeeper-<zookeeper_version>-<mapr_version>
- json-<version>
- the other dependencies of the lib directory of this custom wrapper distribution



! Important

MapR native library is included in these Custom Wrapper dependencies and can be loaded only once.

Therefore, if you plan to access to other MapR sources with Denodo, like:

- MapR FileSystem with HDFS Custom Wrapper
- MapR Event Store with Kafka Custom Wrapper
- Drill with JDBC Wrapper.

you have to use the same classpath to configure all the custom wrappers and the JDBC driver; see `'C:\Work\MapR Certification\mapr-lib'` in the image above.

With this configuration Denodo can reuse the same classloader and load the native library only once.

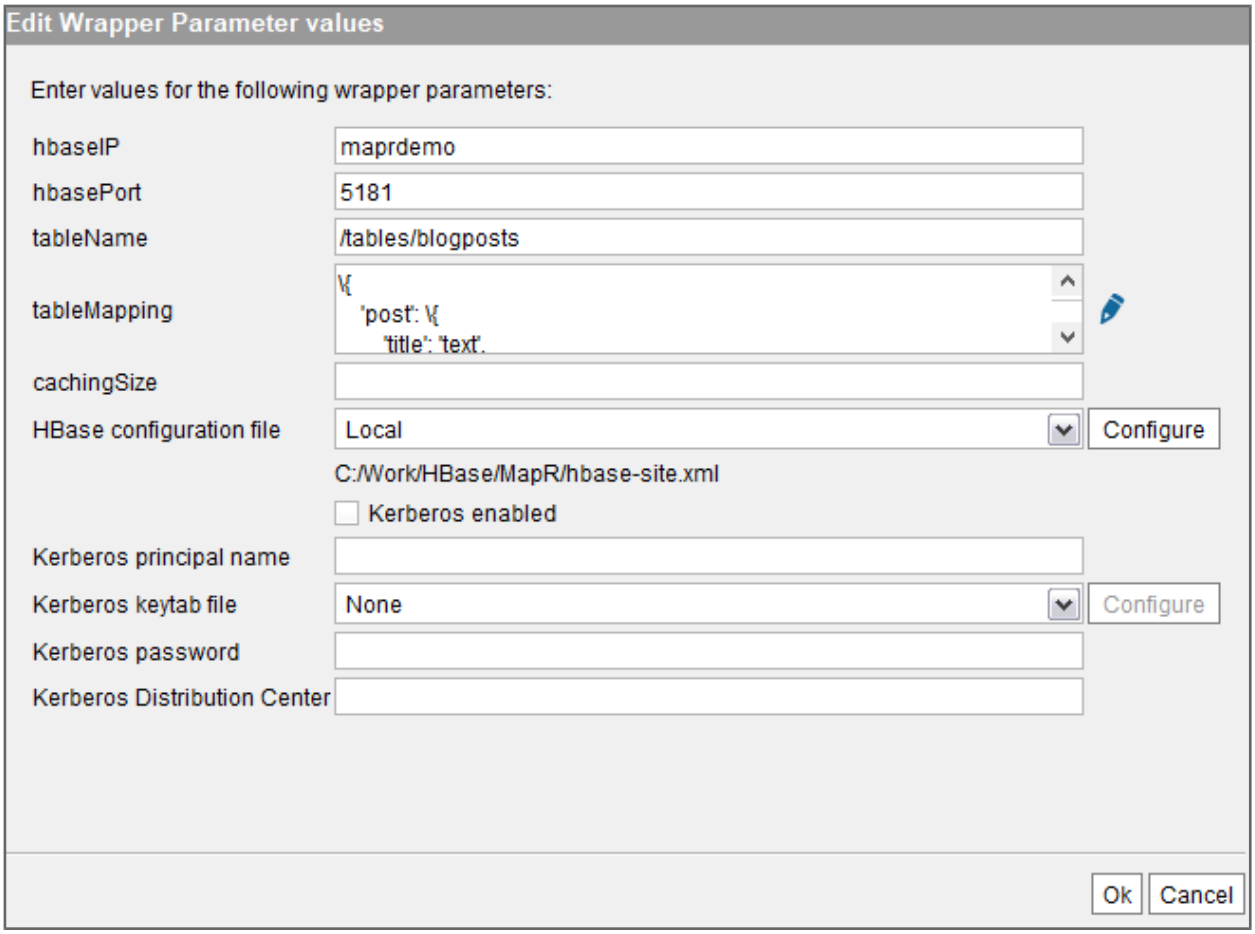
6.2.6 Configure base view

Configure the HBase wrapper parameters as usual, with these three main differences:

- the **`hbasePort`** should be **5181**, as in MapR is the port where Zookeeper is listening for client API calls.
- the **`tableName`** should be the **table path**, instead the table name, e.g. `/user/user01/customer`
- the **`HBase configuration file`** is **required**, the `hbase-site.xml` file from the MapR cluster (`/opt/mapr/hbase/hbase-<version>/conf/hbase-site.xml`) should be used.

```
<configuration>

  <property>
    <name>hbase.rootdir</name>
    <value>maprfs:///hbase</value>
```

```
  </property>

  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>

  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>maprdemo</value>
  </property>

  <property>
    <name>hbase.zookeeper.property.clientPort</name>
    <value>5181</value>
  </property>

  <property>
    <name>dfs.support.append</name>
    <value>true</value>
  </property>

  <property>
    <name>hbase.fsutil.maprfs.impl</name>
    <value>org.apache.hadoop.hbase.util.FSMapRUtils</value>
  </property>

  <property>
    <name>hbase.regionserver.handler.count</name>
    <value>30</value>
  </property>

  <property>
    <name>fs.mapr.threads</name>
    <value>64</value>
  </property>

  <property>
    <name>mapr.hbase.default.db</name>
    <value>maprdb</value>
  </property>
</configuration>
```



MapR base view edition