Denodo MongoDB Custom Wrapper

Revision 20180116

Denodo North America & APAC: 525 University Avenue, Suite 31, Palo Alto, CA 94301. USA
Denodo Iberia & Latino América: Montalbán 5, 28014 Madrid, Spain
Denodo EMEA: 21st Floor, Portland House, Bressenden Place, London SW1E 5RS. UK
Denodo DACH: Karlstraße 10, 80333 München. Germany
www.denodo.com

CONTENTS

1 INTRODUCTION

`mongodb-customwrapper` is a Virtual DataPort custom wrapper for querying MongoDB collections.

It bridges (to some extent) the gap between NoSQL and relational databases by establishing a predefined schema for output, thus enabling SQL queries on MongoDB.

2 WHAT IS MONGODB?

MongoDB is one of the most prominent so-called NoSQL databases, although it would be more accurate to say that is a non-relational database, or as defined on their website: "A document-oriented, scalable and high performance database", developed by 10gen.

In MongoDB we can forget about the table: we have **collections** which are groups of **documents** that do not have to share any common schema. So documents are to MongoDB something very similar to what tuples represent to relational databases, only without the schema restrictions associated with the latter.

Another important feature is the way in which MongoDB stores documents: **BSON** (Binary JSON). The binary part is hidden from us when we work with MongoDB, and thus we see and treat these data in JSON format, so our documents are in fact JSON objects, and its **fields** are the equivalent to the columns of the relational world. But as we do not need the documents in a collection to share a common schema, each JSON document can have just the fields it needs, and all documents in a collection do not have to represent entities of the same nature.

And you can also forget about both the SQL language and the JDBC API. MongoDB does not use SQL but its own query API, and consequently offers a specialized driver for Java that acts as a client and interacts with the database.

A **database** consists of one or more collections, the documents in those collections, and an optional set of security credentials for controlling access.

3 REQUIREMENTS

This Custom wrapper requires:

- When used with Denodo 5.5: At least update 20170529
- When used with Denodo 6.0: At least update 20170515
- MongoDB version 2.4 or newer.

4 ARCHITECTURE AND FEATURES

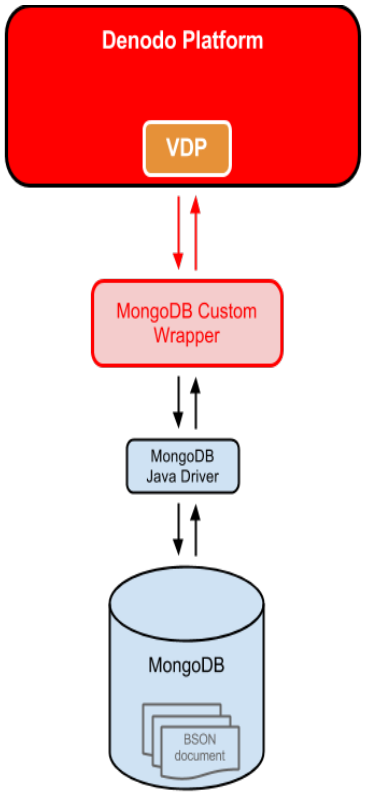
`mongodb-customwrapper` allows us to create base views on MongoDB collections and execute SQL queries on those collections.

This wrapper uses the official MongoDB Driver for Java.

This wrapper uses this driver to get a `MongoClient` object that represents a pool of connections to a MongoDB database. Optionally it can authenticate by user and password.

After making the connection, the custom wrapper can create base views to read collections and documents as if they were tables and rows. Each base view will be created on a specific collection.

Once the base views are created, VDP will be able to perform queries on MongoDB using the VQL Shell, create derived views, etc.



MongoDB Custom Wrapper Architecture

This is a brief summary of the wrapper's current features:

- User/password authentication is supported. User name and password can be specified as input values when building the base view.

- Base view result schema is defined as a comma-separated list of field names.
    - Extra fields in returned documents are ignored.
    - Lacking fields in returned documents are returned as `null`.
    - Types can be specified from constants in `java.sql.Types`, which correspond to SQL standard types.
        - Default is VARCHAR.
        - If documents in the same collection specify different types for fields with the same name, using VARCHAR (`text` in VDP) for that column will automatically perform the required conversions to show data from those documents.
        - Note that the SQL equivalent to MongoDB's "Date" type is SQL's TIMESTAMP, as MongoDB Date values contain both date and time.
        - MongoBD's internal "BSON Timestamp" data type is supported as SQL TIMESTAMP too, but with some limitations (see section at the end).
- Alternatively the schema could be defined using an introspection query. **All documents** (remember, "schemaless") retrieved by this query are analyzed to reveal their fields and build the base view schema. For this reason the query should retrieve a significant sample of the collection we are interested in.
    - Since common fields may hold different types of data the resulting structure is the highest common denominator between all the fields with the same name. In case of incompatible fields --such as `integer` and `subdocuments`-- VDP interprets them as of type `text`. But notice that **MongoDB is strict about types** and you must query for data using the correct type, so this fields would not be searchable.

- Create/update operations using simple fields are supported.
- Delete operations are supported.
- Field projections are delegated to MongoDB.
- Some query conditions are delegated to MongoDB.
    - AND and OR conditions.
    - Operators: =, <>, <, >, <=, >=, `like`, `IS NULL`, `IS NOT NULL`.
        - `IS NULL` operator gets the documents where the specified field has the null value or does not exist.

- `ORDER BY` clause is delegated to MongoDB.

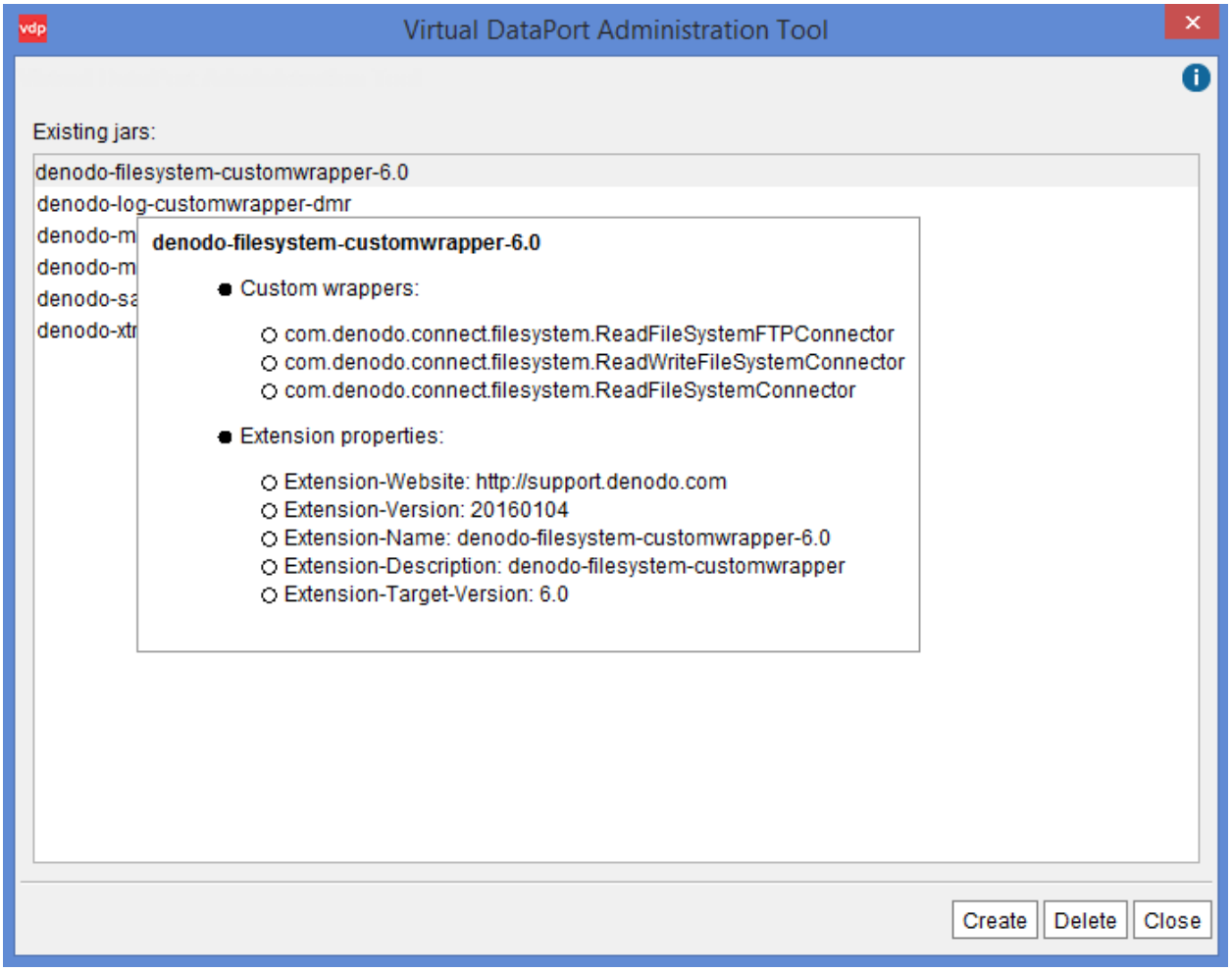
5 USAGE

5.1 IMPORTING THE CUSTOM WRAPPER INTO VDP

In order to use the MongoDB Custom Wrapper in VDP, we must configure the Admin Tool to import the extension.

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

No other jars are required as this one will already contain all the required dependencies, including the MongoDB Java driver classes.

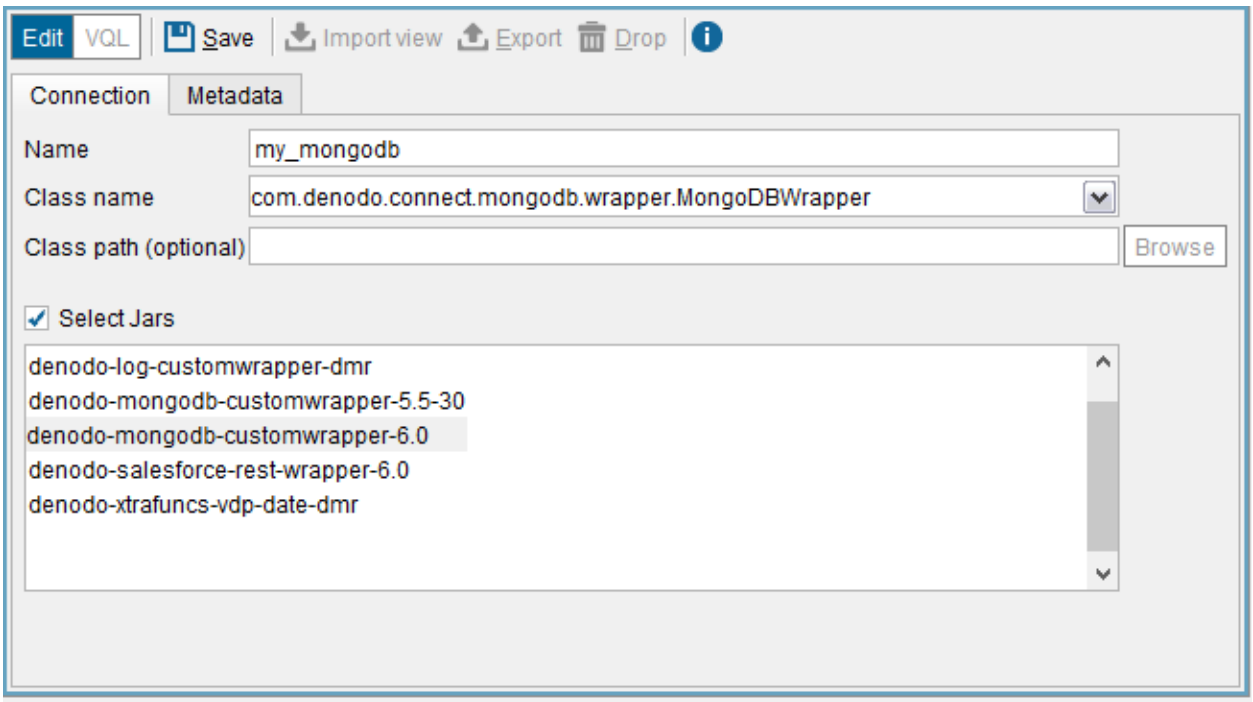


MongoDB Extension in VDP

5.2 CREATING A MONGODB 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` and specify the wrapper's class name `com.denodo.connect.mongodb.wrapper.MongoDBWrapper`. Also check 'Select Jars' and and select the jar file of the custom wrapper.



MongoDB Data Source

5.3 CREATING A BASE VIEW

Once the custom wrapper has been registered, we will be asked by VDP to create a base view for it.

Base views created from the `MongoDBWrapper` need to fill in the `Database` parameter or the `Connection string` parameter, both of them are mutually exclusive. Moreover `Collection` is mandatory

- `Database`: Database name.
- `Connection String`: this parameter allows more options in the connection between the driver and mongodb. This is the pattern: `[mongodb://]host1[:port1][,host2[:port2],...[,hostN[:portN]]]/database[?options]`. The prefix mongodb:// is optional, the database is mandatory and user and password are not written in this field, but in the User and Password parameters
  You can see all the options in the documentation of mongodb, in the following link about Connection String URI Format.
  If you introduce this parameter, host, port and database should be empty.

- `Collection`: Collection name that we will import as a table in VDP.

Four **optional** parameters:

- `Host`: Name of the computer or IP address where MongoDB is running, default is 127.0.0.1. Only with Database parameter.
- `Port`: Port number to connect to MongoDB, default is 27017. Only with Database parameter.
- `User/Password`: Username and password to connect to MongoDB, if the authentication is enabled.

There are also two parameters that are **mutually exclusive**:

- `Fields`: The fields we would like to import as columns in VDP. We must keep the syntax `field1[:type1][,field2[:type2],...]`. Type, when specified, should be one of the constants in `java.sql.Types` (note these are SQL standard types). See a specific section below to learn more about the allowed syntax.
- `Introspection query`: Documents retrieved by this query will be analyzed to reveal their fields and build the view schema. An empty query selects all documents in the collectio, so, if the collection has a lot of documents, the creation of the base view can take a long time. In addition, it is recommendable to check the types of the new view inferred from the introspection. This query requires **MongoDB syntax**.

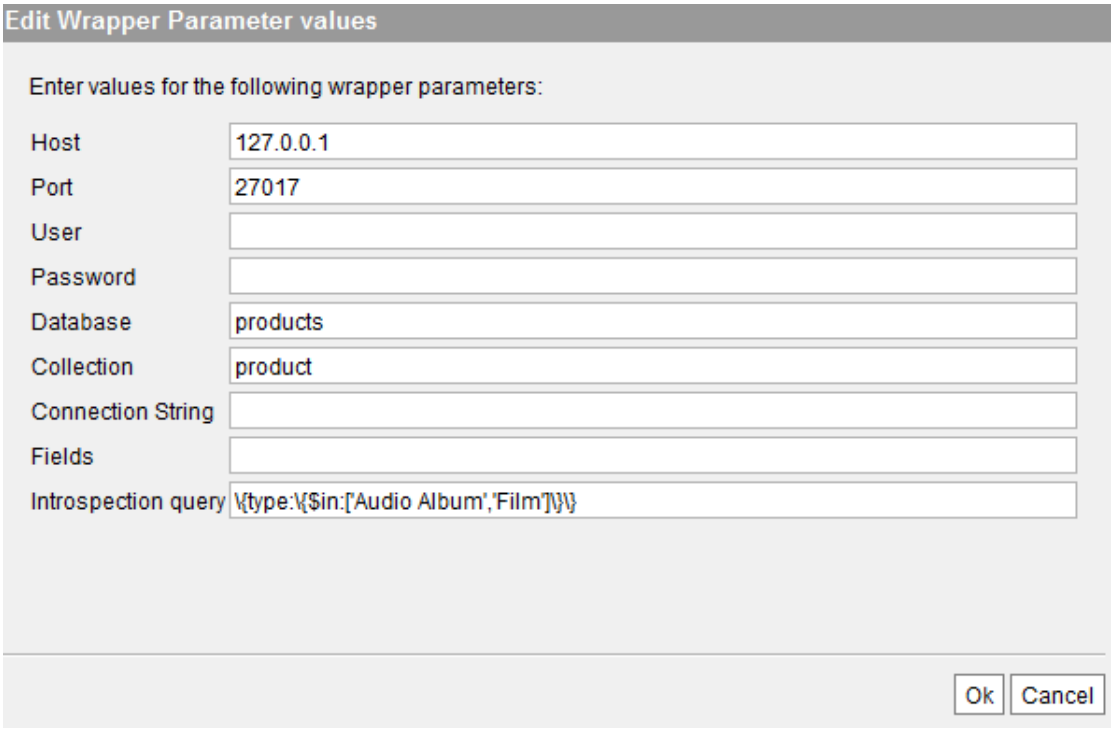
5.4 CREATE BASE VIEW EXAMPLE

In the following example we want to import a product catalog collection of an E-Commerce site database.

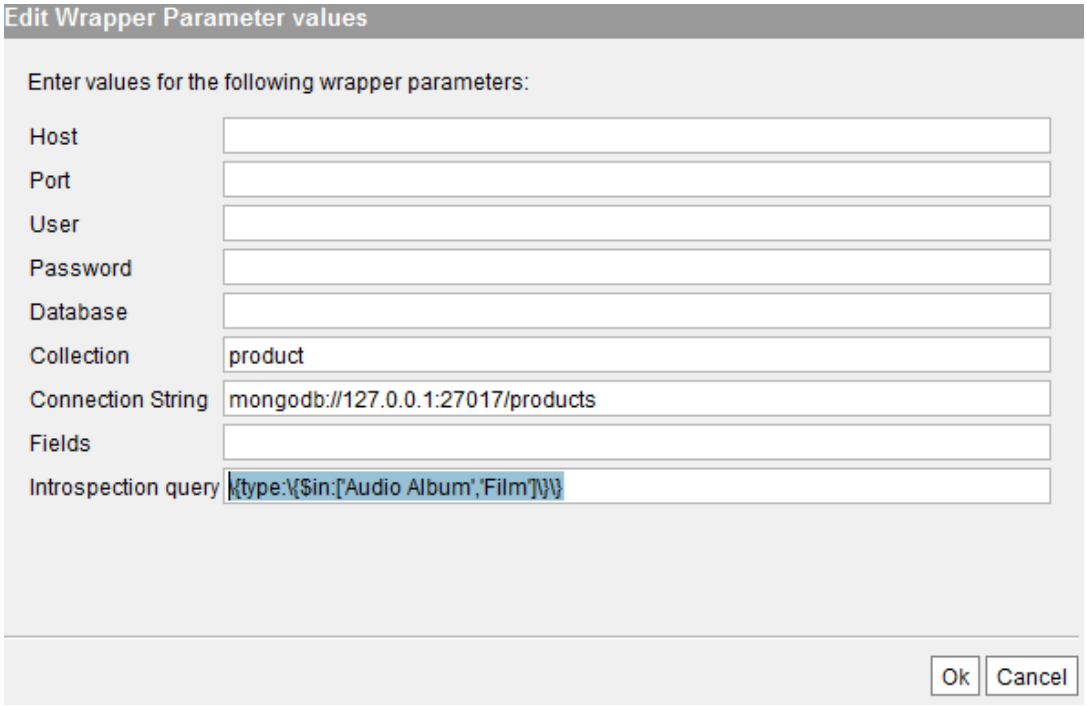
At the beginning of the document, the schema must contain general product information, to facilitate searches of the entire catalog. Then, a details subdocument that contains fields that vary between product types.

As we are only interested in albums and films products we use the following introspection query:

```
{ type: { $in: [ 'Audio Album', 'Film' ] } }
```

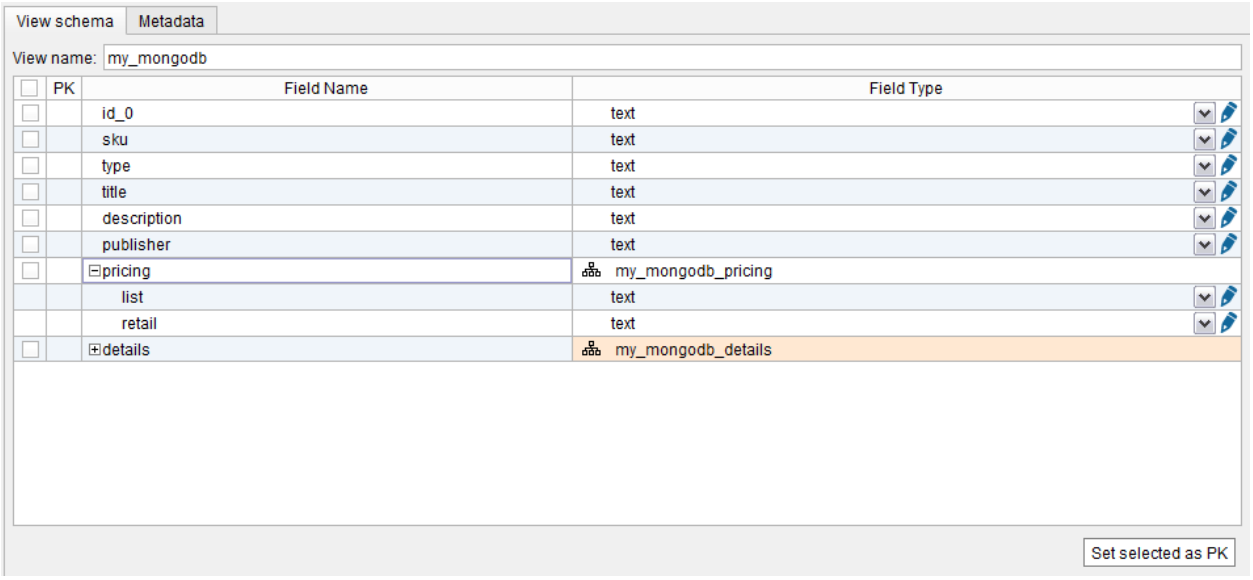


MongoDB Base View edtion using Host, Port and Database parameters

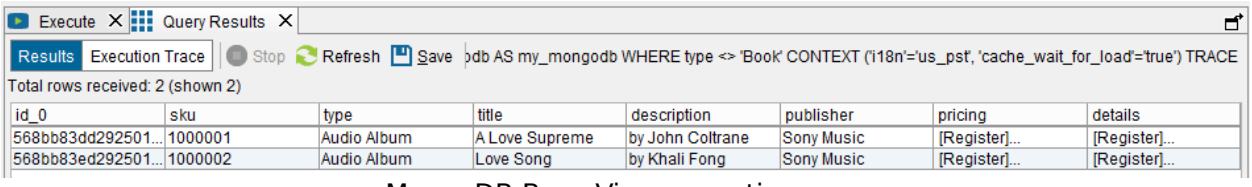


MongoDB Base View edtion using Connection String parameter

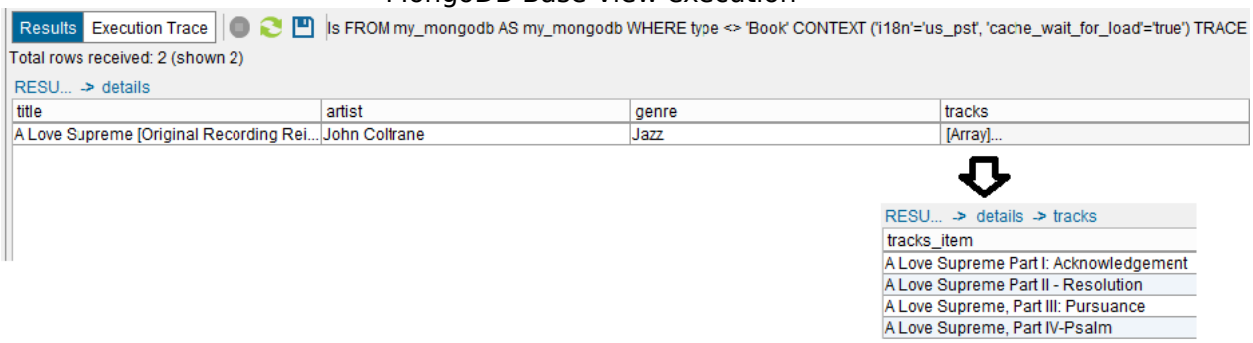
The resulting schema for this product catalog collection can be seen in the image below. It contains common product information like `title`, `type`, `pricing` and the `details` subdocument.



MongoDB Base View



MongoDB Base View execution



Array and record details of MongoDB Base View execution

5.5 CREATE BASE VIEW USING FIELDS

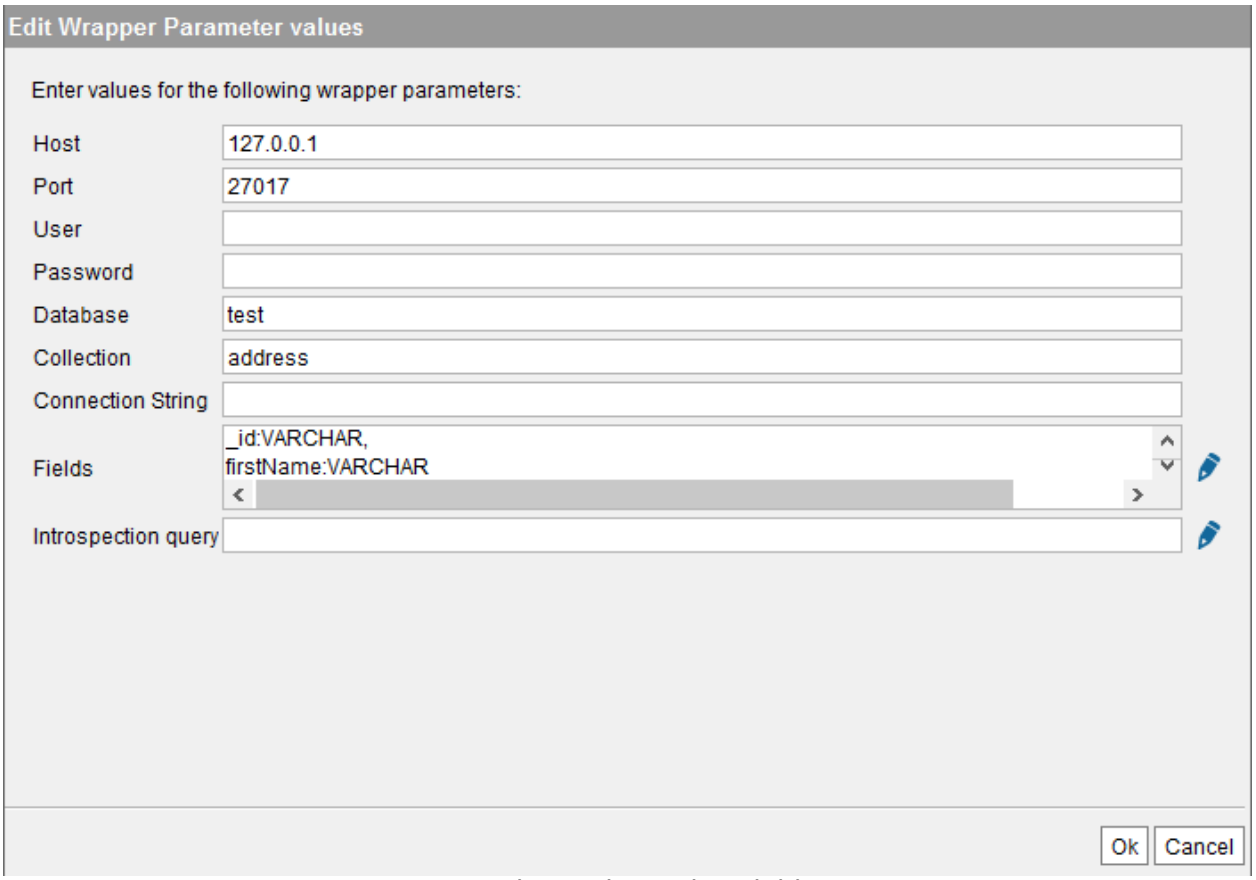
The use of the Fields parameter for specifying the view's schema explicitly is recommend for base view creation when there are complex fields involved coming from mongoDB and there is a possibility that these data are not always returned in the same order or with the same type using an introspection query.

See an example of explicit schema specification (note braces for complex objects should be escaped):

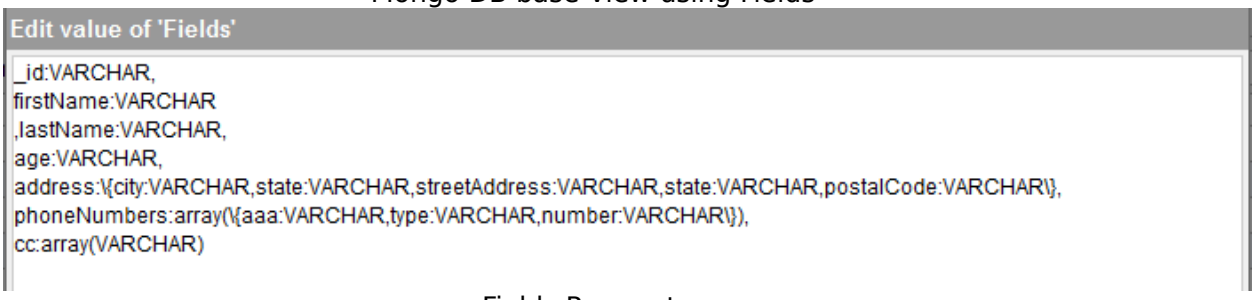
```
_id : VARCHAR,
firstName : VARCHAR,
lastName : VARCHAR,
age : VARCHAR,
address: \{
    city : VARCHAR,
    state : VARCHAR,
    streetAddress : VARCHAR,
    state : VARCHAR,
    postalCode : VARCHAR
    \},
phoneNumbers: array(\{
    aaa : VARCHAR, type : VARCHAR,
    number : VARCHAR
    \}),
cc:array(VARCHAR)
```

See how complex structures are specified using braces (`{...}`) and arrays using the `array` keyword and specifying the contents of the array between parentheses.

Also field names can be surrounded by single or double quotes (e.g. `"phone number"`) if they contain white spaces or non-alphanumeric chars.



Mongo DB base View using Fields

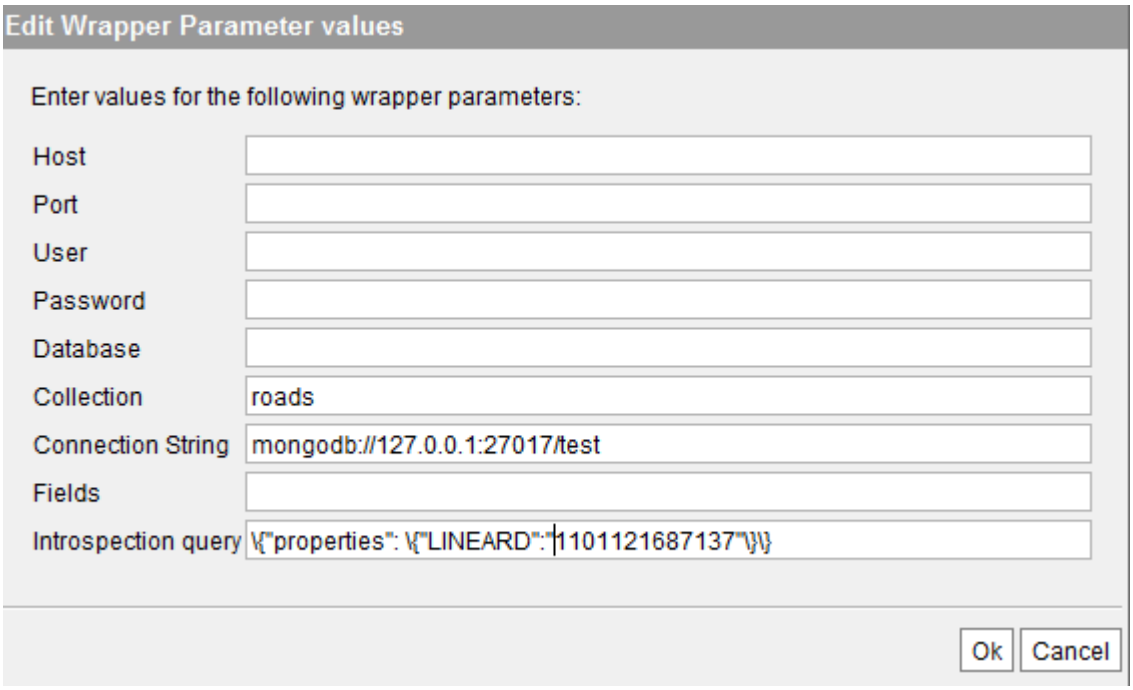


Fields Parameter

5.6 CREATE BASE VIEW USING INTROSPECTION QUERY

If you want to not define the schema, you should use the `Introspection query` parameter. For this you should use a query over the collection, following the syntax for the mongodb method db.collection.find(). You should escape the braces.
A example of introspection query is :

```
\{   "properties":   \{"LINEARID":   "1101121687137",   "observed":   "",
"COUNTYFP":  "051",  "RTTYP":  "M",  "FULLNAME":  "Airport  Rd",  "MTFCC":
"S1400", "STATEFP": "35"\}\}
```



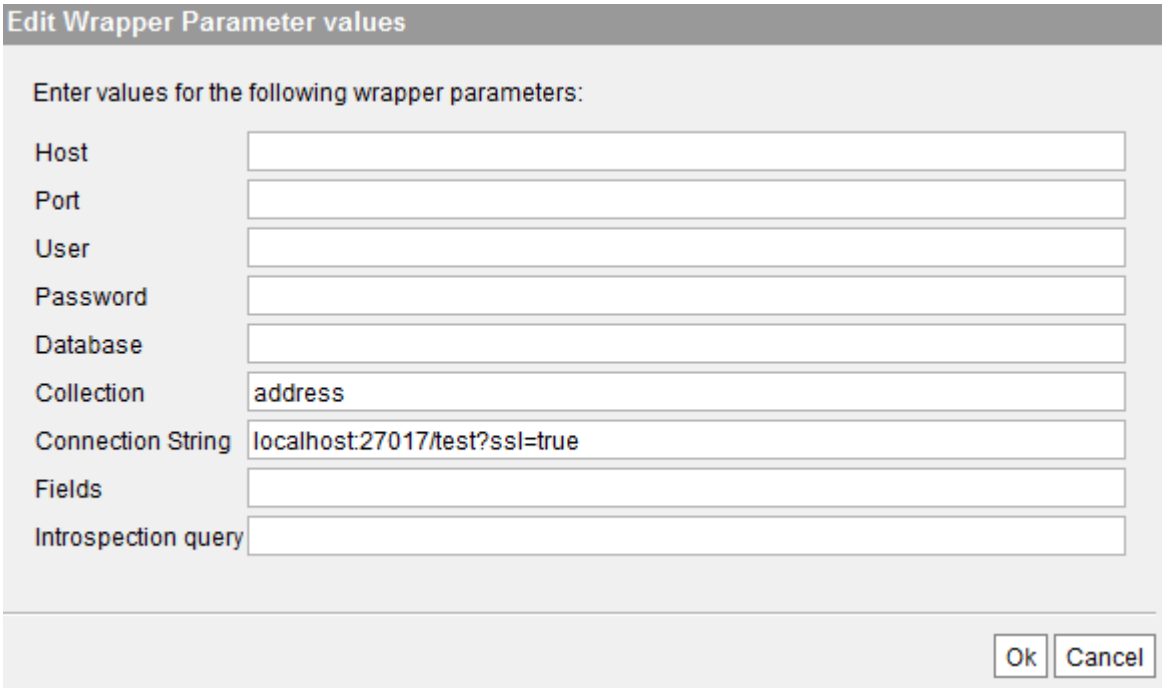
MongoDB Base View edtion using Introspection query parameter

5.7 CREATE BASE VIEW WITH SSL

The MongoDB API for Java Driver allows the connection to a Mongodb Server that is set up with SSL. To create a Base view using SSL, have to add the parameter `ssl` with value `True` in the `Connection String`. SSL is one of the *Connection Options* of the *Connection String URI*.

In addition you have to import the certificate(used by MongoDB server) into the *TrustStore* of the Java Runtime Environment(JRE) of the platform. The section “Importing the Certificates of Data Sources (SSL Connections)” of the Installation Guide explains how to do this.

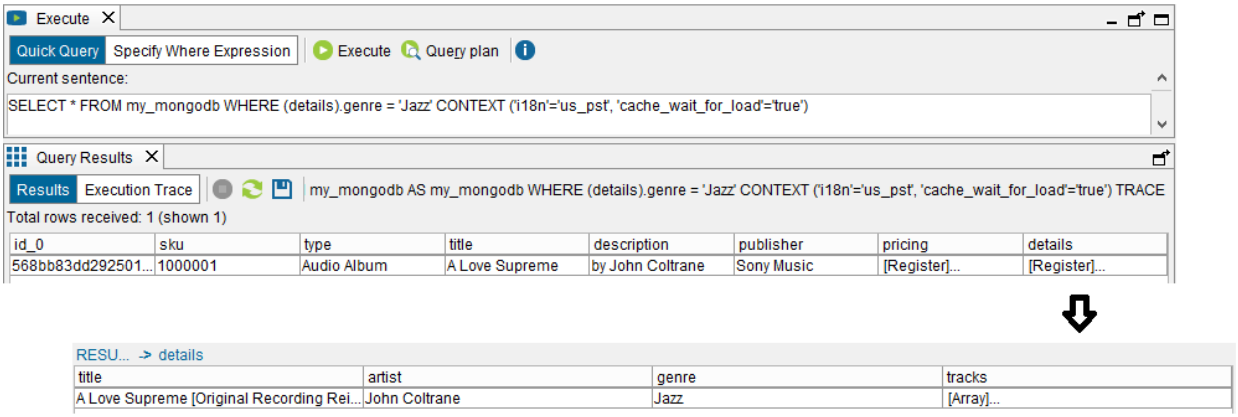
A example of the Base view with SSL:



MongoDB Base View using SSL

5.8 QUERYING MONGODB COLLECTIONS

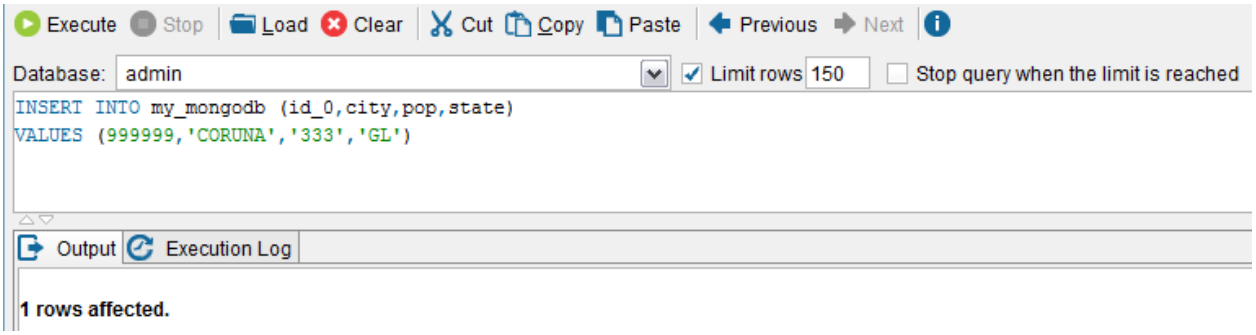
From now on we can query using the VQL Shell tool of the Admin Tool, create derived views, etc.



Querying a MongoDB View

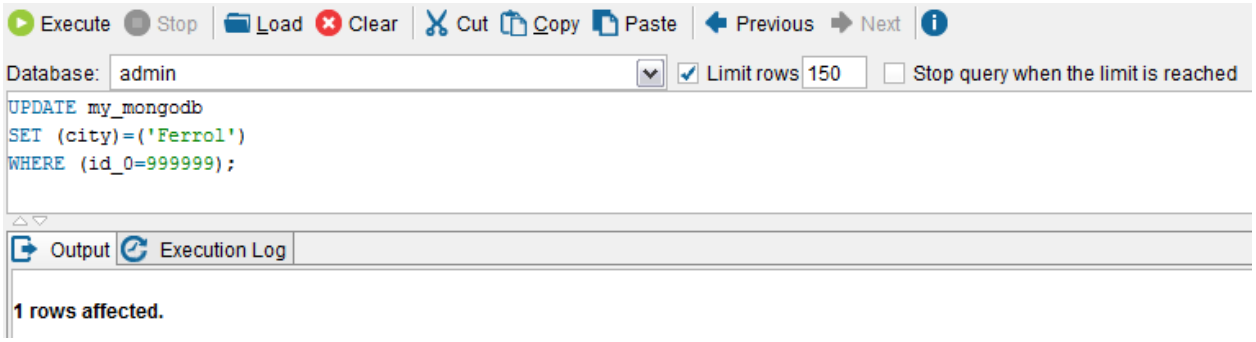
5.9 INSERT

This wrapper allows to make insertion with not complex fields



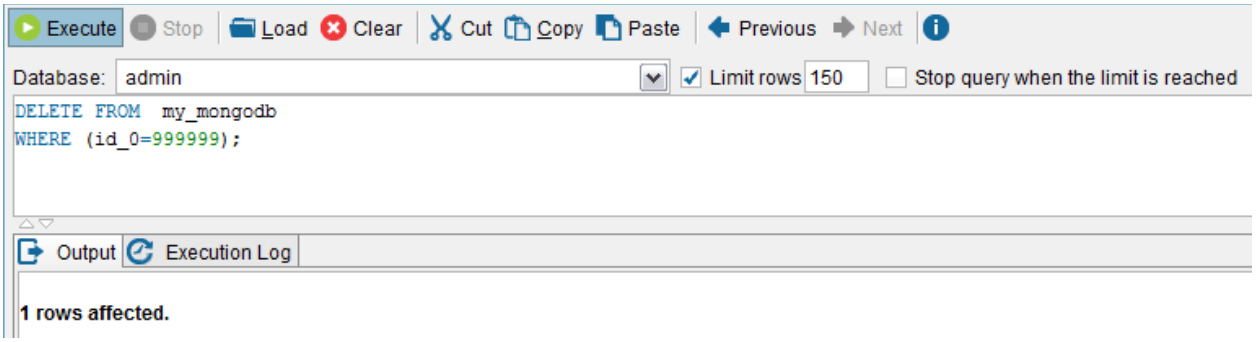
5.10 UPDATE

This wrapper allows to make updates in a document with not complex fields



5.11 DELETE

It is allowed to delete documents of a collection



6 LIMITATIONS

Write operations

Create/update operations with complex fields, arrays and registers, on the MongoDB server are not supported through this custom wrapper at its current version. If you try this operation, you can see the following exception, in the output of the VQL shell:
MongoDB wrapper error. Can't find a codec for class com.denodo.vdb.engine.customwrapper.value.CustomWrapperStruct.

MongoDB BSON Timestamp data type

The MongoDB “BSON Timestamp” data type is considered by MongoDB to be for internal use only (see https://docs.mongodb.com/manual/reference/bson-types/#timestamps), but is nevertheless supported by this MongoDB custom wrapper by considering values of this type equivalent to MongoDB’s recommended “BSON Date” type.

Note however that “BSON Timestamp” data types are not supported in WHERE clauses in queries/updates/deletes, nor they will be used in INSERT or UPDATE sentences to set new values into MongoDB (new values will be set as BSON Date, not BSON Timestamp).