Denodo MongoDB Custom Wrapper

Revision 20140718

Denodo Technologies North America & APAC: 530 Lytton Avenue, Suite 301. Palo Alto, CA 94301. USA
Denodo Technologies Iberia & LatinoAmérica:C/ Montalbán, 5. 28014 Madrid. SPAIN
Denodo Technologies EMEA: 17th Floor, Portland House, Bressenden Place, London SW1E 5RS. UK
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 ARCHITECTURE AND FEATURES

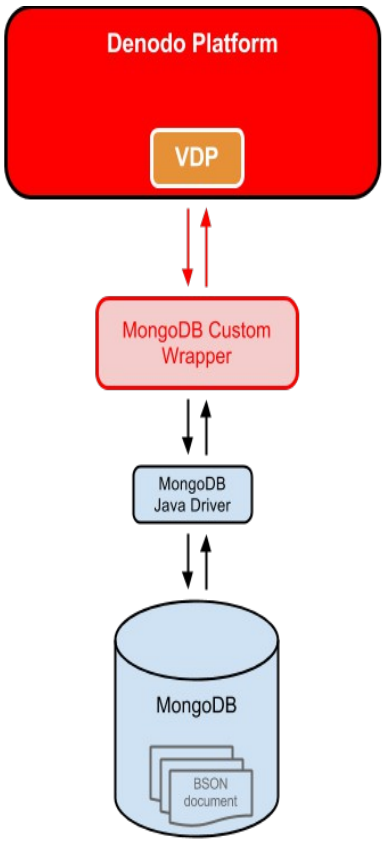
`mongodb-customwrapper` allows us to create base views on MongoDB collections and execute SQL queries on those collections.

The wrapper uses the official MongoDB API for Java Driver that is provided by 10gen.

The 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 `java.sql.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.
- 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.
    - An introspection query should be used when treating with complex fields that holds `arrays` and `subdocuments`.
    - 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.

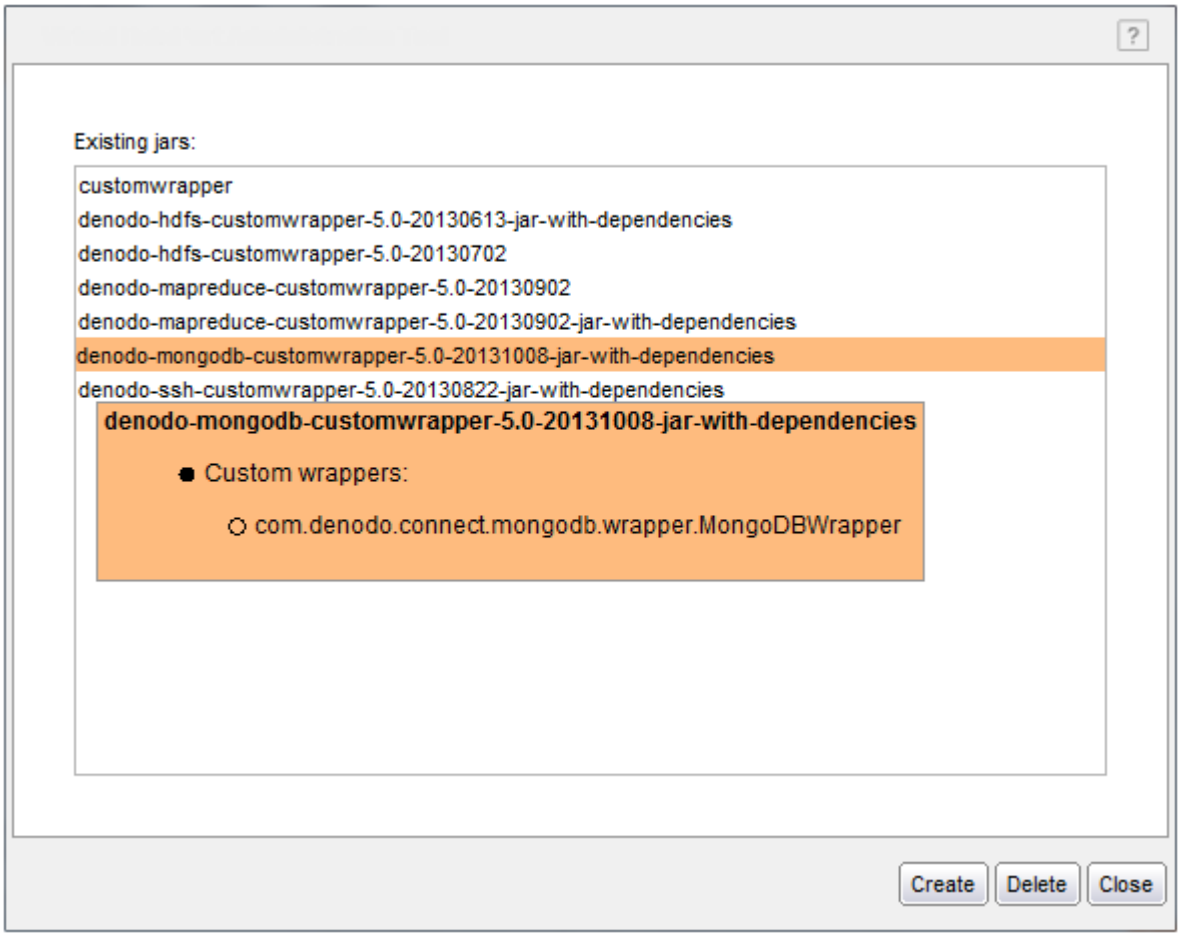
4 USAGE

4.1 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.

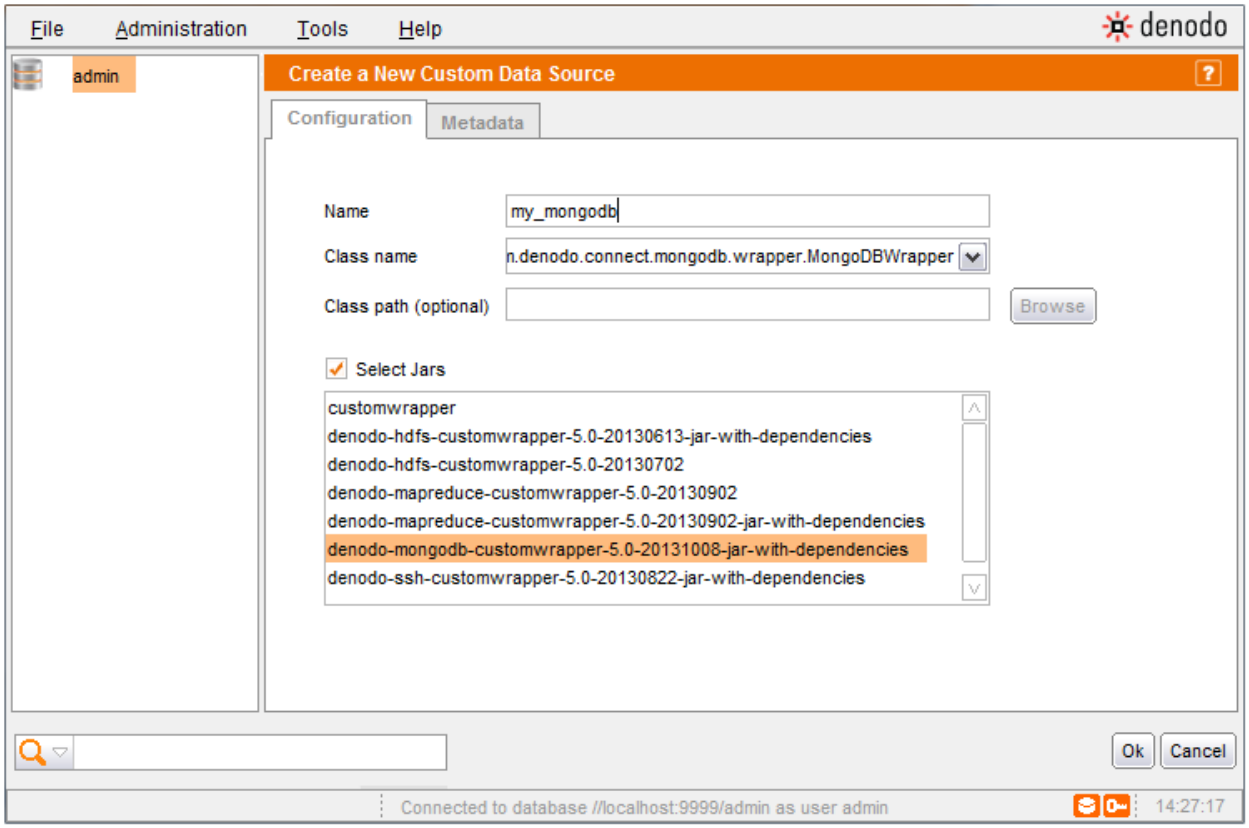


MongoDB Extension in VDP

4.2 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

4.3 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 the following **mandatory** parameters:

- `Database`: Database name.
- `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.
- `Port`: Port number to connect to MongoDB, default is 27017.
- `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 `java.sql.Types`.
- `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 collection. This query requires **MongoDB syntax**.

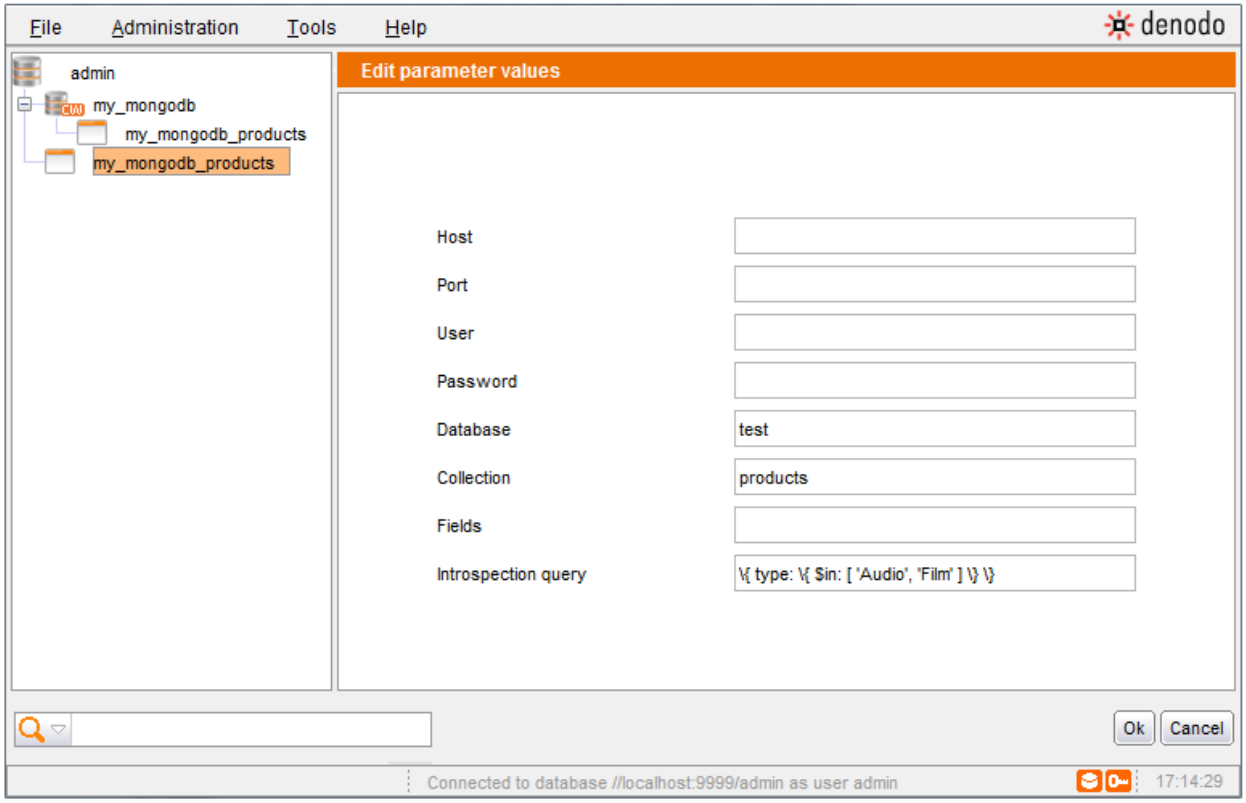
4.4 EXAMPLE

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.

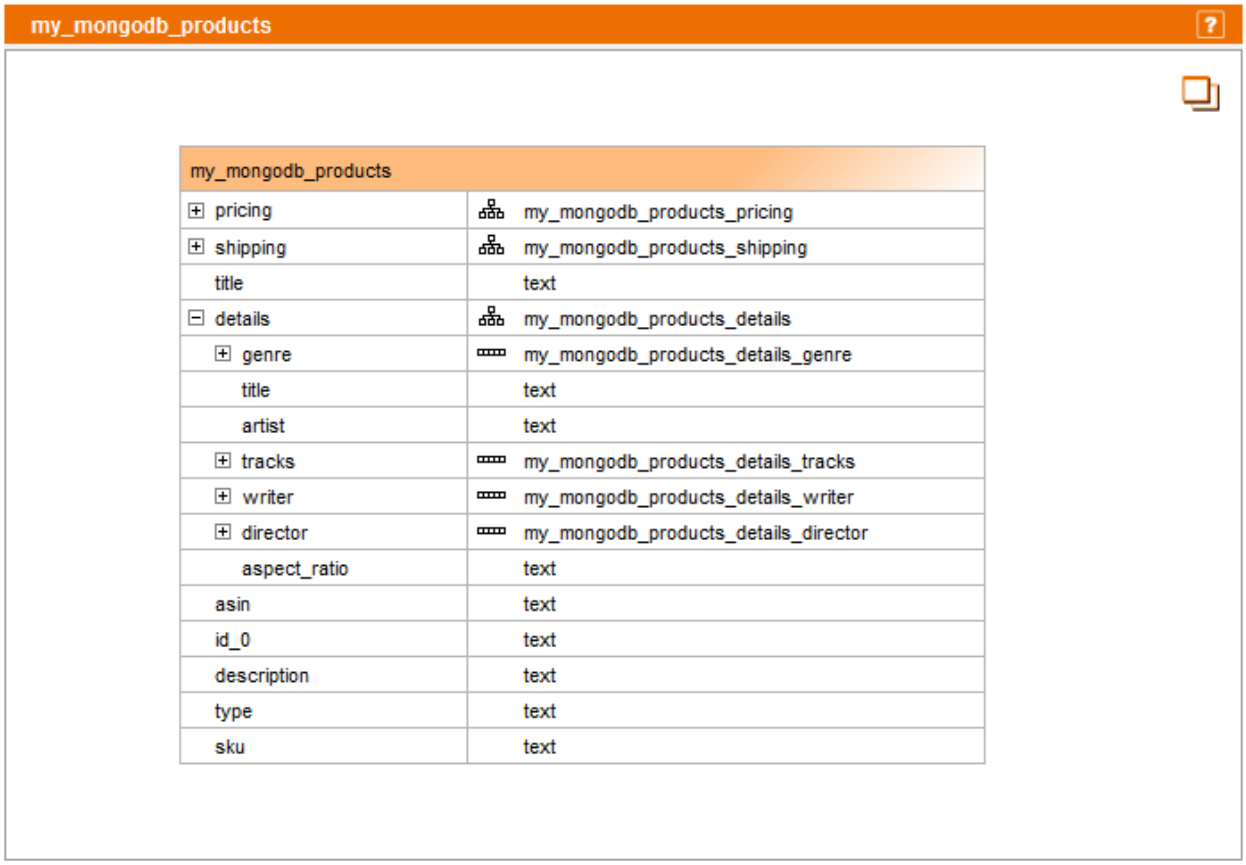
As we are only interested in albums and films products we use the following introspection query:

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

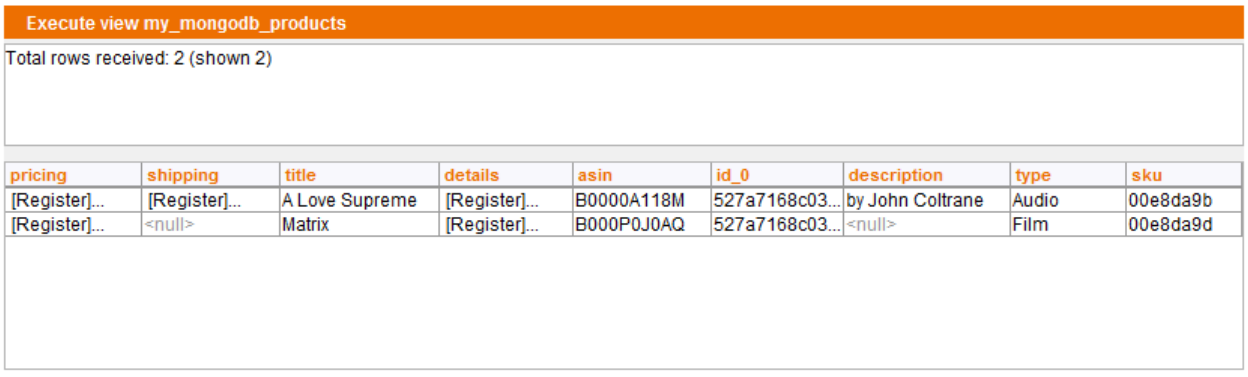


MongoDB Base View edtion

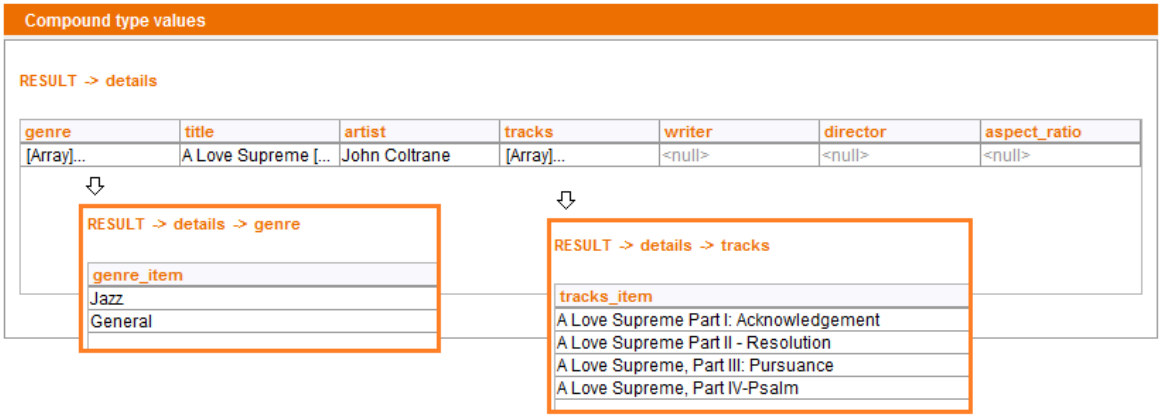
The resulting schema for this product catalog collection can be seen in the image below. It contains common product information like `title`, `type`, `pricing` and the `details` subdocument --an VDP record-- that varies between products: `writer`, `director` and `aspect_ratio` for film products and `artist`, `genre` and `tracks` for audio products.



MongoDB Base View



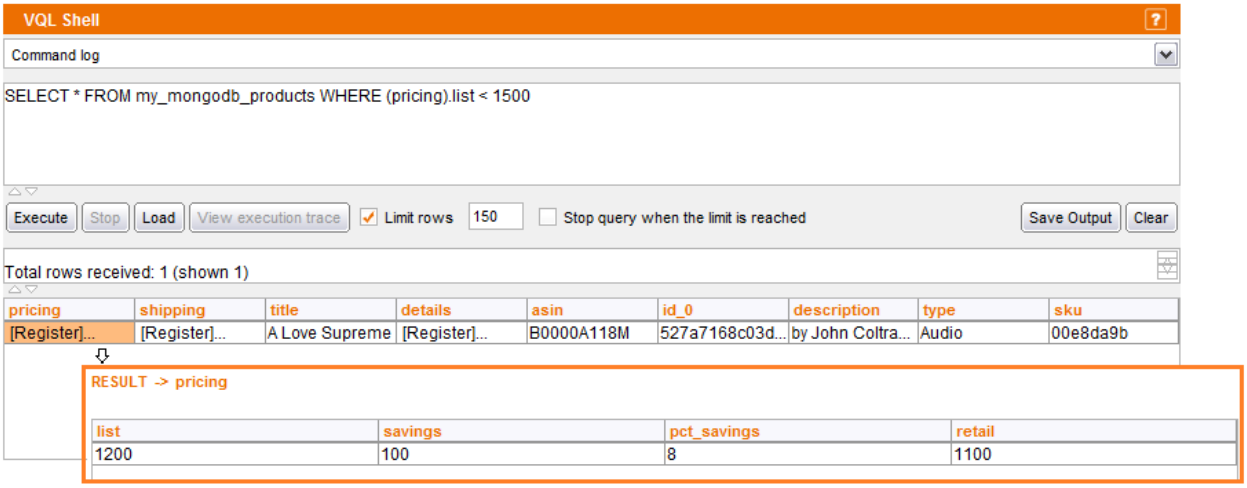
MongoDB Base View execution



Array and record details of MongoDB Base View execution

4.5 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 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.