

DbSchema Tutorial with Introduction in MongoDB

Contents

- MySql vs MongoDB..... 2
- Connect to MongoDB 4
- Insert and Query Data 5
- A Schema for MongoDB ?..... 7
- Relational Data Browse 8
- Virtual Relations 9
- The Query Builder..... 13
- Load JSON files into the database 15
- The End..... 16

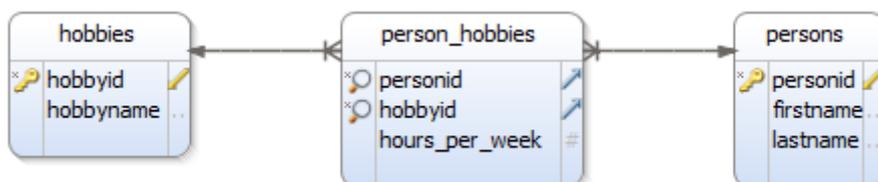
Reading this tutorial to get the basics of MongoDB, understand the JSON documents, the fundamentals of inserting and querying the database, data explorer and diagrams. The tutorial is based on DbSchema tool which you can install and try 15 days for free.

MySQL vs MongoDB

I will explain the difference between SQL databases and NoSQL with a practical example. We will store in MySQL and MongoDB a list of persons with their hobbies. In MySQL we will execute

```
CREATE TABLE PERSONS (  
  personid    integer primary key,  
  firstname   varchar(100),  
  lastname    varchar(200)  
);  
  
CREATE TABLE HOBBIES (  
  hobbyid     integer primary key,  
  hobbyname   varchar(100)  
);  
  
CREATE TABLE PERSON_HOBBIES (  
  personid    integer not null,  
  hobbyid     integer not null,  
  hours_per_week integer,  
  constraint fk1 foreign key( personid ) references persons(personid),  
  constraint fk2 foreign key( hobbyid ) references hobbies(hobbyid)  
);  
  
Insert into persons (personid, firstname, lastname) values (1,'John',  
'Steven');  
Insert into hobbies (hobbyid, hobbyname) values (1, 'Tennis');  
Insert into hobbies (hobbyid, hobbyname) values (2, 'Swimming');  
  
Insert into person_hobbies (personid, hobbyid, hours_per_week) values (1,1,5  
);  
Insert into person_hobbies (personid, hobbyid, hours_per_week) values (1,2,3  
);
```

SQL databases are table-oriented. Each table has a predefined structure as part of the schema. In our case we have created three tables: one for persons, one for hobbies and one which stores each person hobby. You can execute the script above in DbSchema SQL Editor, refresh the schema and get the diagram bellow. For detailed instructions please read the DbSchema SQL tutorial from www.dbschema.com.



To list the hobbies for each person we have to execute:

```
SELECT p.firstname, p.lastname, ph.hours_per_week, h.hobbyname
FROM sample2.persons p
  INNER JOIN sample2.person_hobbies ph ON (p.personid = ph.personid)
  INNER JOIN sample2.hobbies h ON (ph.hobbyid = h.hobbyid)
```

And the result:

firstname	lastname	hobbyname	hours_per...
John	Steven	Tennis	5
John	Steven	Swimming	3

In MongoDB the data can have a hierarchical structure, called JSON. Here is a JSON document:

```
{
  Firstname: 'John',
  Lastname: 'Steven',
  Hobbies:
  {
    {
      Hobbyname: 'Tennis',
      HoursPerDate : 5
    },
    {
      Hobbyname: 'Swimming',
      HoursPerDate : 3
    }
  }
}
```

This document is in fact a text which will be saved in MongoDB. In the next chapter we will connect to MongoDB and implement this inside the database.

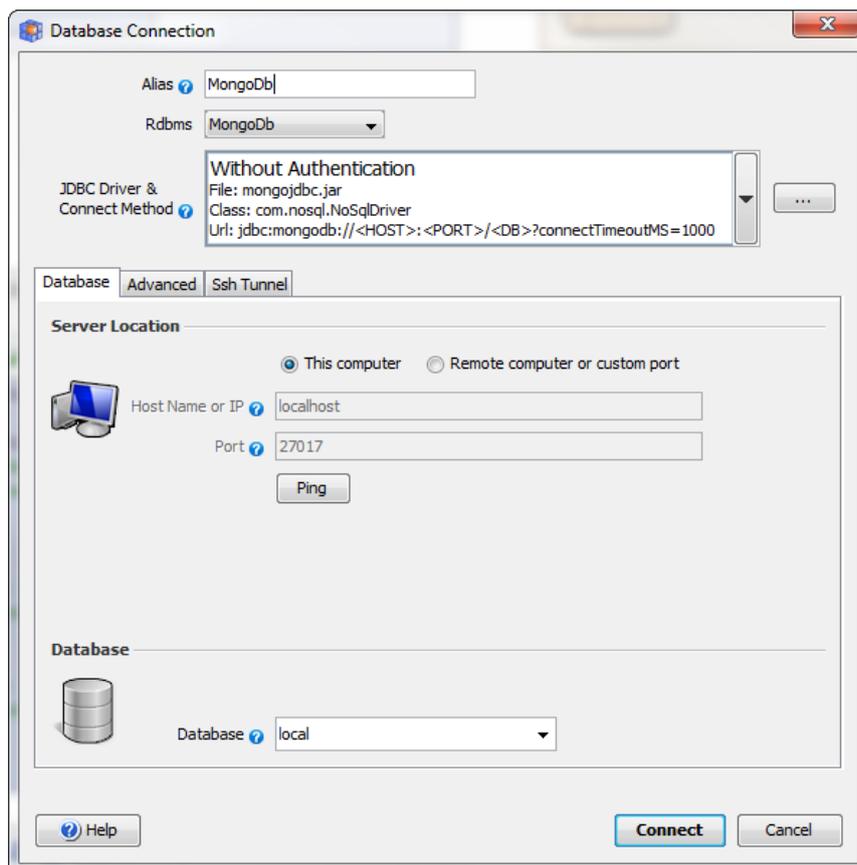
Connect to MongoDB

First download and install MongoDB from <https://www.mongodb.org>. Download also DbSchema from <http://www.dbschema.com>. You may use DbSchema trial for 2 weeks for free.

On windows start the Mongo daemon from the command prompt. The mongo daemon may require to create a data directory (will complain if it does not exists) or you may specify a different folder using **mongod.exe -dbpath <path>**

```
Administrator: C:\windows\system32\cmd.exe - mongod.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Alle Rechte vorbehalten.
C:\Users\DP186021>cd "c:\Program Files\MongoDB\Server\3.0\bin"
c:\Program Files\MongoDB\Server\3.0\bin>mongod.exe
```

Start DbSchema and choose 'New Project Connected to Database'.

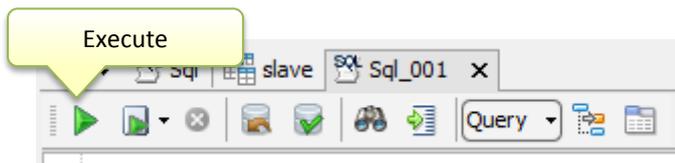


Here we will choose the connection method without authentication (use this unless the server default changes has changed). The host is the machine where the database resides (localhost if is the same machine where DbSchema is started).

Insert and Query Data

Open a new SQL editor inside DbSchema and copy inside the text bellow:

```
local.widgets.insertOne( {
  "debug": "on",
  "window": {
    "title": "Sample Konfabulator Widget",
    "name": "main_window",
    "width": 500,
    "height": 500
  },
  "image": {
    "src": "Images/Sun.png",
    "name": "sun1",
    "hOffset": 250,
    "vOffset": 250,
    "alignment": "center"
  }
});
```



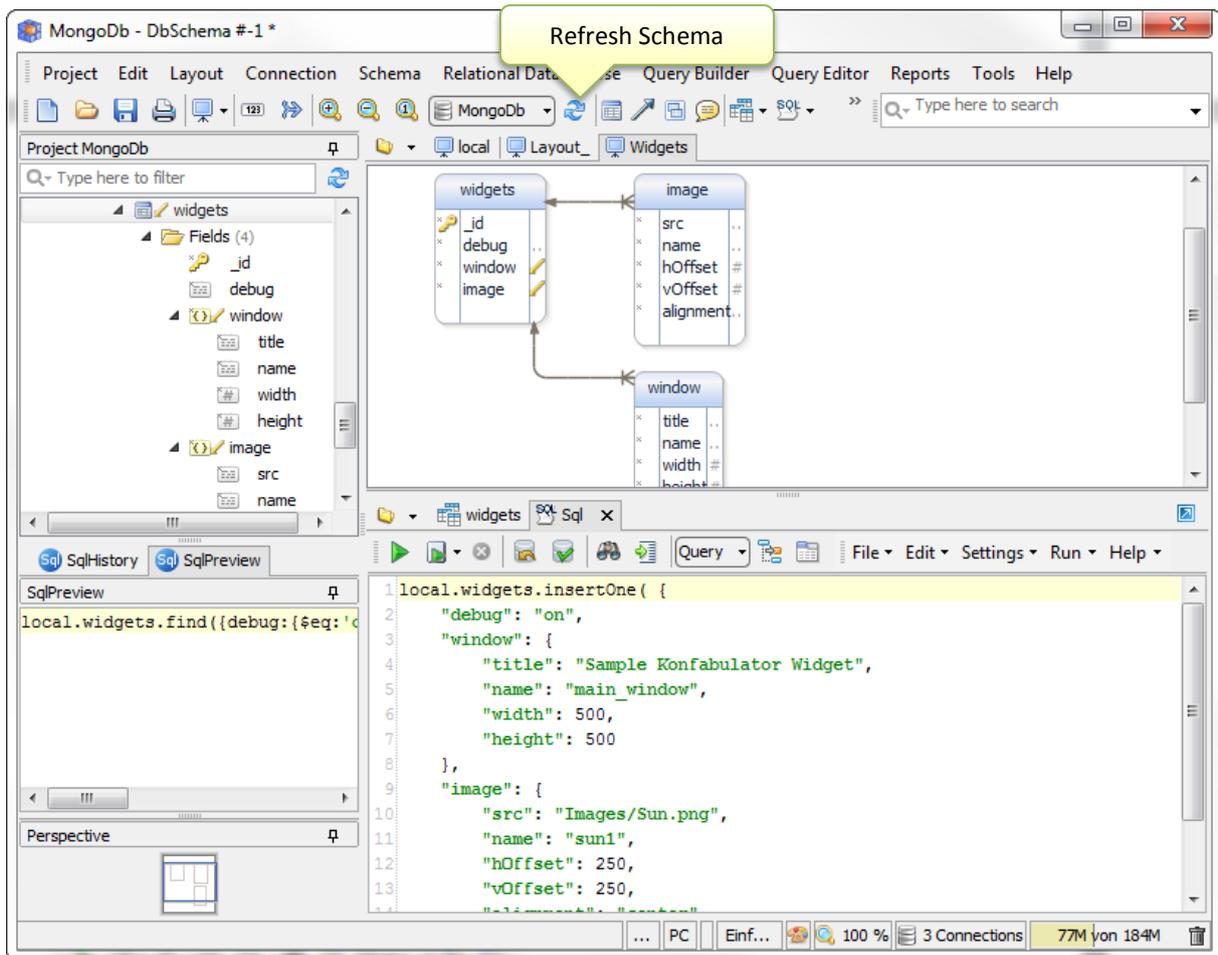
Place the caret inside the text and press the execute button in the SQL editor. The editor will execute the selected text or the text where the caret is, delimited by empty lines. This creates the collection if not existing and store the document inside. Press the 'refresh schema' inside DbSchema to get the entities in the diagram.



Copy and execute in SQL editor:

```
db.widgets.find()
```

The result shows as a hierarchical structure.



The query language used by DbSchema is the same as in MongoDB. You can use `db.<collection>.find()` as well as `<database>.<collection>.find()`, example

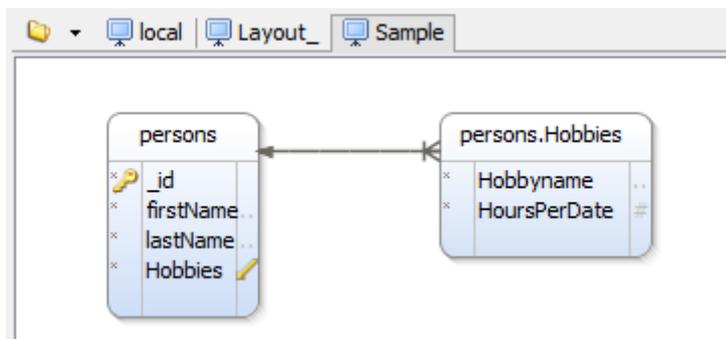
```
local.sample.insertOne({name: 'Sam'})  
local.sample.find({name: 'Sam'})  
use local  
db.sample.find({name: 'Sam'})
```

For MongoDB DbSchema implements its own JDBC driver by using the JavaScript engine embedded in Java 1.8 and the original MongoDB 3.1 java driver. All classes exposed in the Mongo Java driver can be used from DbSchema. Check <http://api.mongodb.org/java/current> for documentation.

A Schema for MongoDB ?

In relational databases the schema must be created before data can be stored in the database (this is done using CREATE TABLE...). In MongoDB no schema is required. We simply push the data in some collections. Nobody will tell us what to save inside. It is possible to save the companies and employees in the same collection. Most of the programmers won't do this because is hard to read this later.

DbSchema does a 'schema discovery' by scanning the database data. The schema is presented in the structure tree and diagrammed layouts. Bellow two entities were created for the 'persons' collection, one for the main document and one for the sub-document.



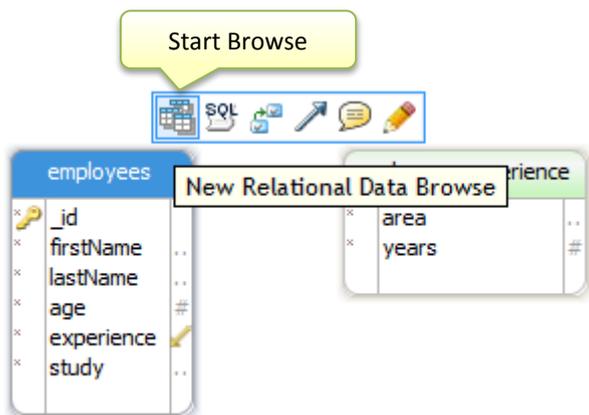
It is possible to create multiple layouts with the same or different tables. The layouts will be saved to DbSchema project file.

DbSchema use its own image of the schema, so when the database is modified you should 'refresh the schema from the database'. Using an internal image is possible to compare two different databases, open the project file without being connected to the database, etc.

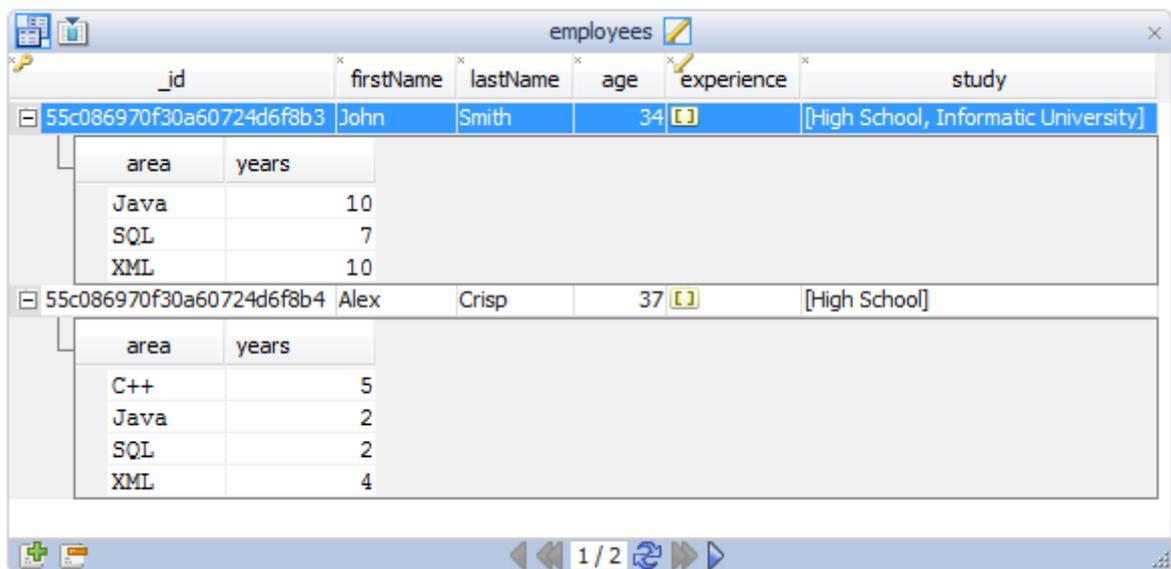
Relational Data Browse

Use relational data browse to view or **edit** the data from different collections. Follow this chapter to understand how to browse data from different collections using **virtual relations**.

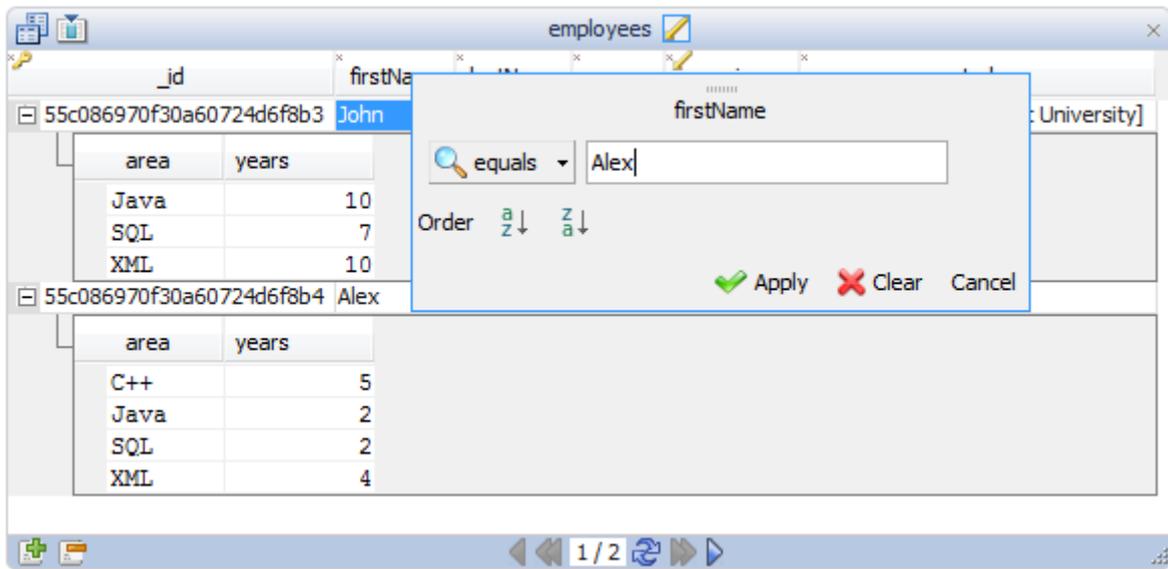
Start the browse by clicking the table header and choose 'browse'. This will open in the browse editor the first table.



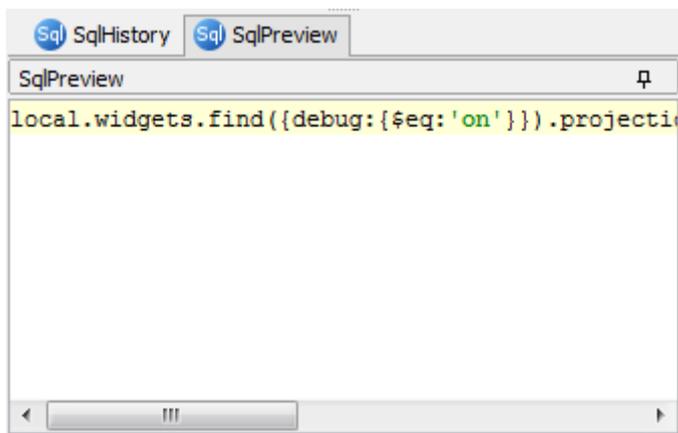
A browse frame will be opened in the Browse Editor. If a document has children sub-documents, then the record can be expanded.



Click a browse table header column to start filtering the data. This will create a special query to filter data over the entire collection.



You can see the used query on the left side, in the history panel.



Virtual Relations

In MongoDB you can refer one document from another document via ObjectIds. As example consider a collection 'airports' with name, location, etc., and a collection 'flights'. The flights collection may refer the 'airport' collection and not repeat each time the entire data which is stored for the airport.

Collections have assigned automatically an **_id** field (done automatically by Mongo DB when you store some data). This value can be used in the referencing collection to point to the referred collection.

Copy the example bellow in DbSchema SQL Editor. Select with the mouse one block of text (like one for statement with all following lines) and execute them one by one by pressing the 'run single query' button.

```

local.master.drop()
local.slave.drop()

for ( i = 0; i < 100; i++){
    local.master.insertOne({name: 'Master__' + i, position: i })
}

local.master.find()

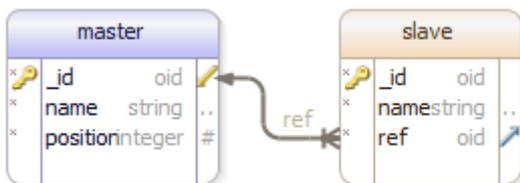
for ( i = 0; i < 100; i++){
    rnd = Math.floor( Math.random() * 100 )
    masterId = local.master.find({position :{ $eq : rnd }}).first()._id
    local.slave.insertOne({ name: "Slave__"+i, ref : masterId })
}

local.slave.find()

```

This code is creating a collection 'master' with name and position. The next collection slave has a field ref as the `_id` of one of the master documents. You can copy-paste this in DbSchema and execute it. Refresh the schema as in the chapters before to get the collection into the diagram.

The line between collections is a virtual relation, meaning the 'ref' field is pointing to the 'master' collection. The virtual foreign keys are saved in the DbSchema project file.



Virtual foreign key will help then in the Relational Data Browse to explore the data from two collections keeping track of the matching between them:

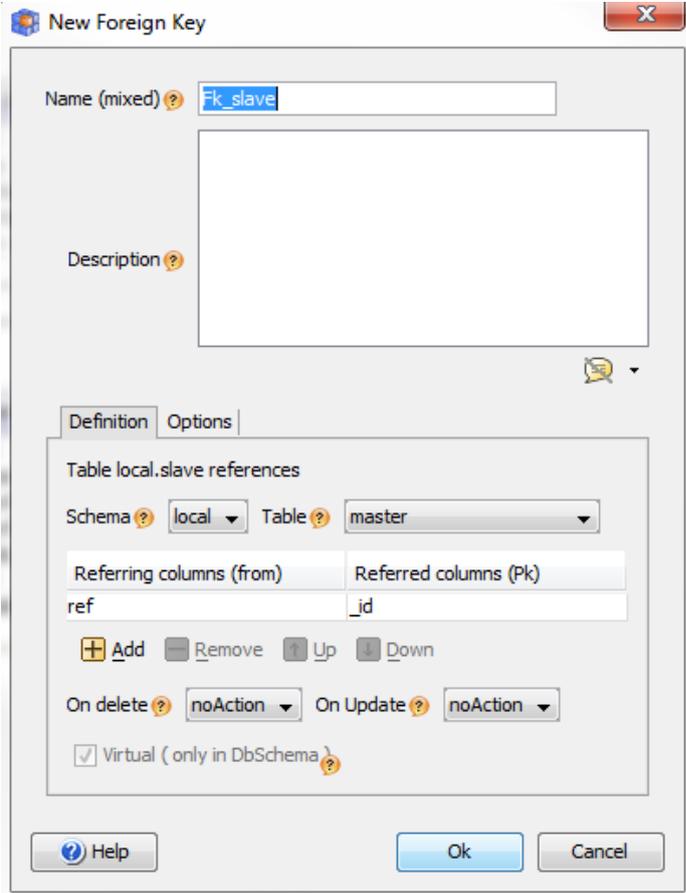
Press this arrow to descend into the collection 'slave'

_id	name	position
566a89a54e20e41b6c15e54e	Master_0	0
566a89a54e20e41b6c15e54f	Master_1	1,00
566a89a54e20e41b6c15e550	Master_2	2,00
566a89a54e20e41b6c15e551	Master_3	3,00
566a89a54e20e41b6c15e552	Master_4	4,00
566a89a54e20e41b6c15e553	Master_5	5,00
566a89a54e20e41b6c15e554	Master_6	6,00
566a89a54e20e41b6c15e555	Master_7	7,00
566a89a54e20e41b6c15e556	Master_8	8,00
566a89a54e20e41b6c15e557	Master_9	9,00

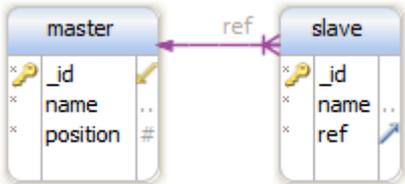
_id	name	ref
566a8a1f4e20e41b6c15e5ea	Slave_53	566a89a54e20e41b6c15e551

Now let's create a virtual relation. For this drag and drop the ref column over the _id column with the mouse by keeping the mouse button.

This will open the foreign key dialog.



On the bottom is a checkbox 'Virtual' which is checked and disabled. All foreign keys created in DbSchema for Mongo DB are by default virtual. The virtual relation will be painted with a distinct color.



Now you can browse the data from master or slave and cascade into the other collection. The browse will show only the records corresponding to the selected record in the first browse frame.

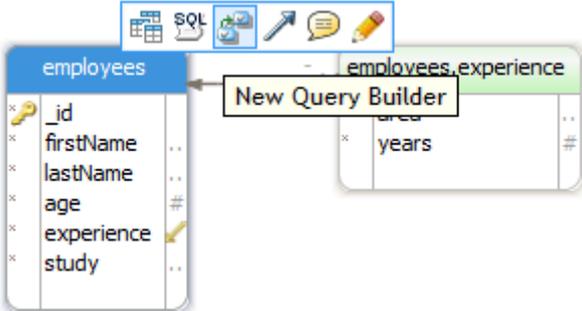
_id	name	position
558cf913bd03ec29bcd69843	Master_0	0
558cf913bd03ec29bcd69844	Master_1	1
558cf913bd03ec29bcd69845	Master_2	2
558cf913bd03ec29bcd69846	Master_3	3
558cf913bd03ec29bcd69847	Master_4	4
558cf913bd03ec29bcd69848	Master_5	5
558cf913bd03ec29bcd69849	Master_6	6
558cf913bd03ec29bcd6984a	Master_7	7
558cf913bd03ec29bcd6984b	Master_8	8
558cf913bd03ec29bcd6984c	Master_9	9
558cf913bd03ec29bcd6984d	Master_10	10
558cf913bd03ec29bcd6984e	Master_11	11

_id	name	ref
558cf916bd03ec29bcd698c2	Slave_27	558cf913bd03ec29bcd69844

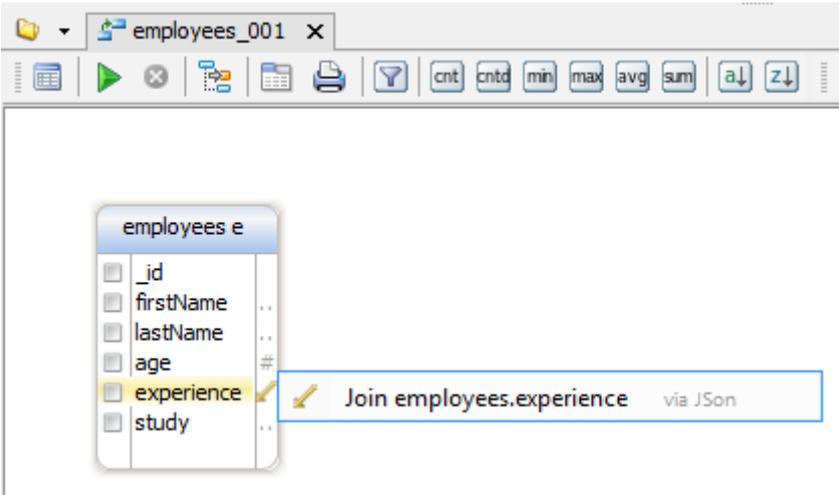
The virtual relations are created only in DbSchema and not in the database. Save the DbSchema project to file and the virtual relations will be saved as well. Next time when you open the application the diagrams and the virtual foreign keys are available.

The Query Builder

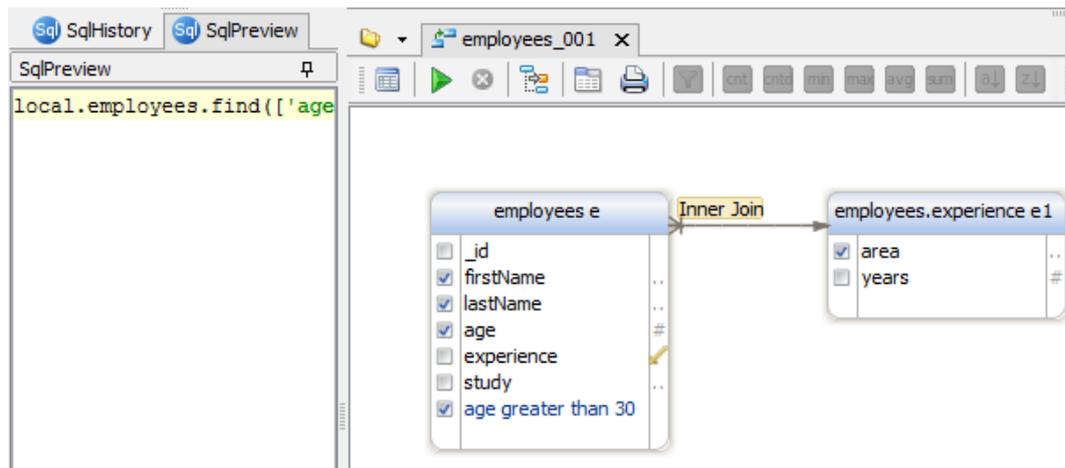
Use the query builder for creating more complicated queries using the graphical interface. This works similar with the browse: click a table header and choose the query builder.



In the created query editor press the foreign key icon near 'experience' to go for the experience sub-document.



Tick the checkboxes for the columns you want to select. Right-click any column to set a 'where' filter like **age > 30**. The generated query is visible on the left, in the Preview panel.



You can execute the query directly in the editor or copy the generated query from the preview panel.

In MongoDB the queries will work only over one single collection. Therefore the virtual relations are useless in the query builder. The application itself has to bind the information from two different collections if needed. Alternative is to write a map-reduce job.

You can write map-reduce jobs in DbSchema as well. This and further groovy operations are documented on <http://www.dbschema.com/mongodb-tool.html>. Below is a sample map-reduce job.

```

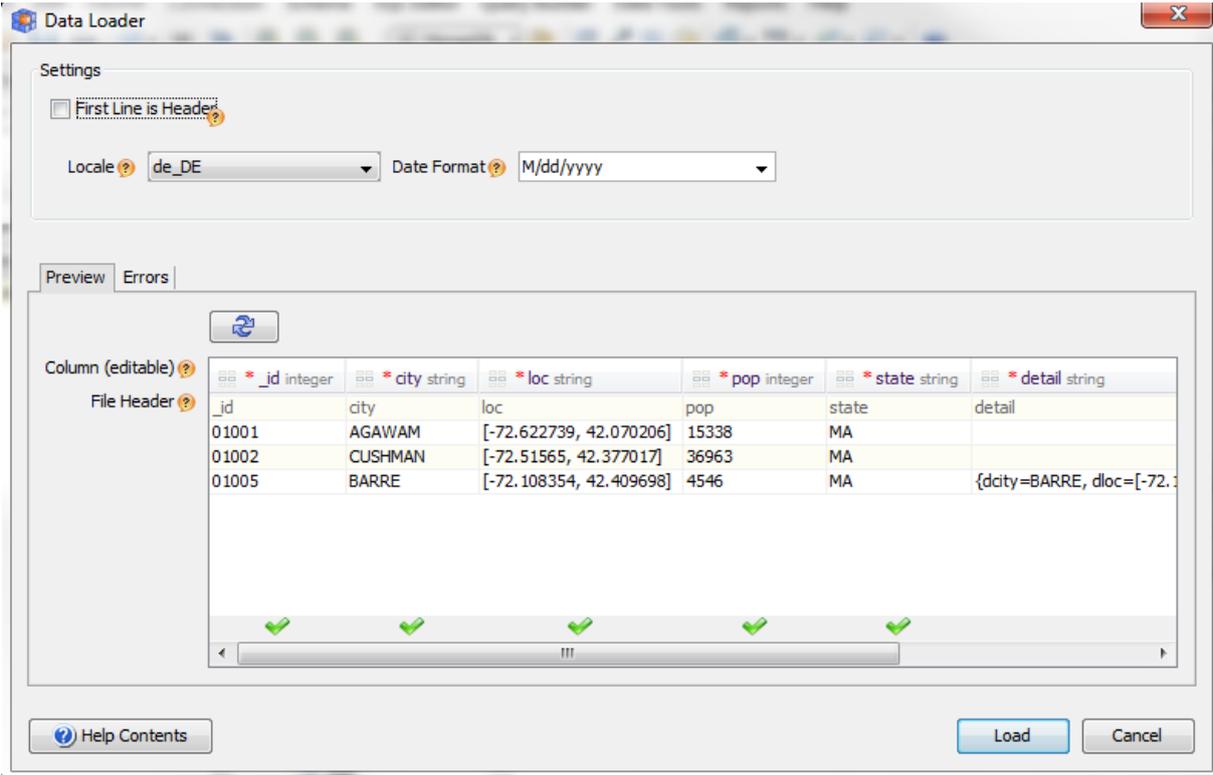
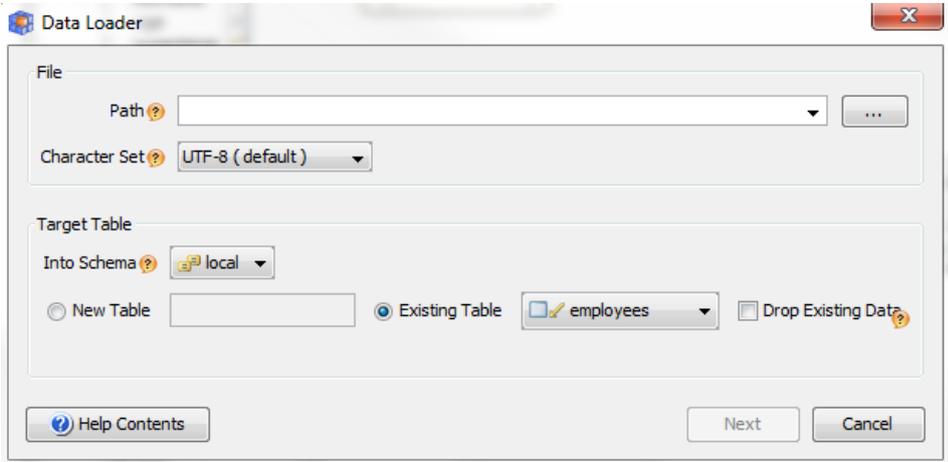
local.words.insertOne({word: 'bla'});
local.words.insertOne({word: 'cla'});
local.words.insertOne({word: 'zla'});

var m =function map() {
    emit(this.word, {count: 5})
}
var r=function reduce(key, values) {
var count = 5
    for (var i = 0; i < values.length; i++)
        count += values[i].count
    return {count: count}
}
local.words.mapReduce(m, r );

```

Load JSON files into the database

From the 'Data Tools' application menu choose the data loader to load JSON files into the database.



The End

DbSchema for Mongo DB is a beta feature. Write us about the bugs you encounter and DbSchema features. This will help us to improve the application. From DbSchema help menu choose 'Report a bug' to write to technical.

We wish you having fun using DbSchema!