Connecting LuaTEX to MongoDB

Roberto Giacomelli

Abstract

This paper describes in details a way to connect LuaTEX to a MongoDB database server. As a consequence, the Lua-powered typesetting engine becomes a candidate, along with other tools, to generate beautiful reports.

From local-wide project to large web applications, MongoDB—the popular NoSQL database—creates a new point of view on datasets also adopted by LuaTEX. This could extend the development perspectives.

Sommario

Questo articolo descrive in dettaglio un modo per connettere LuaTEX a un database server MongoDB, dimostrando come il motore di composizione dotato dell'interprete Lua possa candidarsi, insieme ad altri strumenti, a generare bellissimi report.

Dal piccolo progetto alle grandi applicazioni web, MongoDB — il noto database di tipo NoSQL — crea un nuovo punto di vista sui dati che si ritrova anche in LuaTEX. Ciò potrebbe incrementare ulteriormente le possibilità di sviluppo.

1 Reasons

I'm currently working on a project based on LuajitLATEX, and SQLite3[1] as a set of databases handler, to typeset a lot of high quality reports. I exposed about that in my article (Giacomelli, 2017).

LuajitTEX includes the Luajit FFI Foreign Function Interface, a powerful technology to bind C executable modules such as the dynamic linking SQLite3 library. Not only LuajitTEX becomes capable of collecting data by running SQL queries, but it also makes life easier in configuring the system.

Why did I discard simpler data management software such as spreadsheets or plain text files, and why didn't I ensure a full separation between TEX and data layer, for example through an agnostic file format?

First of all, the main goal of this architecture is to ensure *data safety*: that is why I chose a RDBMS system[2]. Every record saved in the archive is checked by the database engine itself that controls the integrity of the relational model, in consideration of a handful of concepts such as the primary and foreign key constraint, the `not null` clause and so on.

Secondly, the valuable advantage is that avoiding intermediate layer between data and TEX means speeding up modifications.

I have frequently adapted project entity-relationship diagrams whenever improvements were (e.g., the addition of a complex set of economic information). So, I learned a lot about how to translate real world entities into conceptual SQL models, trying to keep minimalism in mind as much as possible.

Changes were easy to apply but, after six years of development, things are no longer so easy to keep up-to-date despite the success of the project. After all, the world is constantly changing. And I consider more important finding out database systems to allow a better data representation than incrementing data bandwidth as fast as hardware potentially could do. In other words, I'm looking for more natural criteria to model a reality in rapid and unpredictable evolution.

When I talk about *project* I mean a production business where data are shared over the IT network infrastructure and automatically processed to get reports. Just think of a freelance engineer or a little design team, dealing with invoices of technical documents as part of their customer service, or a company delivering financial statements to the clients.

1.1 What I'll talk about

This paper is covering a brief description of the NoSQL unconventional way of thinking in the section 2, where I also introduce the MongoDB database system presenting fundamental concepts like the *document*.

Section 3 and the following two are dedicated to a step-by-step detailed tutorial valid for Ubuntu and Windows operating systems—section 4 and section 5 respectively. They provide instructions to set up a MongoDB local server for experimenting and a suitable MongoDB connector for LuaTEX.

Finally, from section 7 on I will discuss two demo projects simple *and* meaningful, useful to practice the projects development.

1.2 Verbatim conventions

When a shell command doesn't fit in the column width, an ending backslash `\` is added to split text up into more lines.

While experimenting, you can improve readability too, splitting the lines up in your command

1. `https://www.sqlite.org`.
2. The acronym RDBMS stands for Relational Database Management System.

window according to your operating system syntax. For instance, Windows PowerShell supports Shift+Enter keys combination for multiline editing or even a backtick[3] ` at the end of breaking lines. Beware: PowerShell inserts a space for each newline you enter, so you won't be able to even split up a file path, while Linux or Mac OSX user can safely use backslash \ to break up to command names.

1.3 Requirements

You need a desktop computer running a 64 bit Operating System in order to execute demo projects programs illustrated from section 7 on: running locally database operations and typesetting documents for reports. A basic Lua understanding is also recommended to knowingly run such source files with LuaTEX.

2 NoSQL, Not only SQL

NoSQL philosophy denies the SQL model: no fixed schemas and no relational constraints are imposed on data. Leaving out that essential checks, web applications can take advantage of NoSQL in two main areas: a great capability to scale out—that is partitioning data across several servers, optimizing costs and performances—and a flexible design of data models. Shortly, a powerful and easy to use storage engine.

The heart of the SQL system is the *table*: a tabular dataset organized in rows and columns that describes and represents entities. Improvements require to reformulate all the dataset archived in the table. This is what we call a fixed-schema: it's just not possible to write data that don't suit the current schema.

Quite the opposite for the NoSQL databases: the information models can be eventually changed, whatever and wherever.

Google, Amazon and other Internet big companies are developing their own NoSQL data storage system. In this article, I'm going to consider the open source project MongoDB located at www.mongodb.com.

2.1 The MongoDB document

A MongoDB *document* is a key/value ordered list. As a matter of fact, not only the database can archive data in binary format, but can also understand the structure of the key/value data type for querying.

A document looks like this:

```
{"name":"Missouri River", "length km":3768}
```

In absence of a fixed and predefined SQL-like schema, MongoDB developers use to visualize how information is structured, simply printing documents. Thus documents literally represent themselves.

In the document above—delimited by braces—there are two fields separated by a comma. Each field has a key and a value separated by a colon. Keys are strings while values are, respectively, a string and an integer. Values are not limited to atomic types like those just considered: they can also represent compound types such as arrays—a list of comma separated values, enclosed in square brackets—or even other documents. For example, we can add geospatial information as an array of coordinates localizing the river source:

```
{
    "name" : "Missouri River",
    "length km" : 3768,
    "source coords" : [45.9275, -111.508056]
}
```

That notation—pretty much the same of the table constructor in Lua—corresponds exactly to the JavaScript object type and it highlights the strong inclination of MongoDB for web development.

2.2 Collection and database

A MongoDB *collection* is a set of documents and, in turn, a *database* is a set of collections.

Collections are referenced by name as a single UTF-8 string or as a sequence of strings concatenated with the dot character. It's the recommended way to organize document groups with namespaces.

For instance, the previous document regarding essential data about rivers could be part of the `info.basic` collection. Please pay attention to the fact that in this case there is no collection belonging to another one but it's only an optional naming rule to identify a *single* collection.

There are no constraints defined on the document structure except on the field `_id`. If the document does not have an `_id` field, MongoDB will attach one to it with a default value of type `ObjectId`, otherwise it will check if the `_id` value of any type is unique within the collection.

2.3 The JSON and BSON formats

JSON (JavaScript Object Notation) is a text-based data-interchange open standard format. It is built upon a key/value structure called *object* and a list called *array*. Values types belong to a list of seven of them like string or number, but even object and array.

The JSON specification takes only a single web page, see https://www.json.org/, to be completely defined. As the website says, JSON is easy for humans to read and write, and it is also easy for machines to parse and generate.

JSON is not alone in the arena of structured text formats; competitors are TOML[4], YAML[5],

3. Backtick is issued typing ALT+0096 on Windows machines.

4. Tom's Obvious, Minimal Language https://github.com/toml-lang/toml.

5. YAML Ain't Markup Language http://yaml.org/.

XML[6], RON[7] and many others, but JSON is very popular on the modern web.

The data transfer over the network is based on the serialization/deserialization process where text is encoded into an efficient binary format. BSON (Binary JSON) is the network transfer format used by MongoDB.

2.4 Working with `mongo`

Once MongoDB is installed, a JavaScript-based CLI client called `mongo` is available alongside the other executables.

`mongo` is a way for accomplishing administrative tasks on databases and it is a useful tool for getting started with CRUD operations and querying. The acronym CRUD stands for the four basic operations on databases: Create, Read, Update and Delete.

2.5 MongoDB learning resource

For more information on MongoDB visit the `https://docs.mongodb.com` website. It's a clear, complete and well organized documentation site. A good printed reference is the book (Chodorow, 2013).

Furthermore, online courses can be attended at `https://university.mongodb.com/` website: basic, intermediate and advanced levels can suit everybody's needs.

3 Setting up a stand-alone system

The objective is to get started with a locally-running MongoDB instance. With few adjustments you should be able to run the database server within a Local Area Network. Of course, a production-ready installation of a MongoDB server requires special treatments for security issues, an effort that goes beyond the objectives of the present work.

Anyway, a 64 bit operating system is required to run recent versions of MongoDB. No such limitation involves the client running from within a LuaTeX process.

A Lua-MongoDB connector can be build in different ways:

- a low level binding to the MongoDB C driver `http://mongoc.org/`;
- a pure Lua connector that understands the MongoDB server binary protocol over TCP network protocol;
- an intermediate node connected to LuaTeX via TCP using the library `luasocket`, that is statically linked in the Lua-powered typesetting engine. This proxy server is connected to MongoDB by means of every suitable programming languages;
- an independent execution of CLI tools like `mongo` shell itself, using files as communication channel with LuaTeX (see section 10).

Depending on the state of the art of open source projects, I chose the low level bindings. Building them requires two steps regardless of the target OS: compiling the MongoDB C driver and binding Lua to it via the LuaRocks package manager. LuaRocks must use the same Lua version of the LuaTeX interpreter, currently 5.2 for a standard TeX Live 2018 distribution.

To determine your Lua version you can read the LuaTeX manual shipped with TeX Live[8] or compile the following file with LuaTeX and check the resulting PDF:

```
% !TeX program = LuaTeX
\directlua{tex.print(lua.version)}
\bye
```

3.1 Installing a MongoDB driver for LuaTeX

The first step is to compile the MongoDB C driver and its companion. The project home is at `http://mongoc.org/`, where we can find both source code and documentation for the library `libmongoc` and the companion `libbson`. The latest library deals with the BSON format, that is the binary format used by MongoDB to store documents in the database and to perform network communication, as mentioned in the section 2.3.

The second step is to compile a Lua binding to MongoDB C driver. The project I take into account is `lua-mongo` hosted on GitHub at `https://github.com/neoxic/lua-mongo`.

In this paper I will only give detailed installation instructions for Ubuntu Linux and Windows.

4 Installing on Ubuntu

First of all it's important to notice that the MongoDB binary for Ubuntu has been compiled with SSL enabled and dynamically linked. This requires the SSL libraries to be seperately installed on your system with the following shell command:

```
$ sudo apt-get install libcurl4 openssl
```

Adding to the repository list the official MongoDB software archive, you can also install the binary via the standard `apt` package manager. Doing so the package will be updated automatically. However, the manual procedure allows you to keep things under a complete control.

4.1 Installing MongoDB

MongoDB installation is very simple because you only have to download and unzip a file: select

6. Extensible Markup Language `https://www.w3.org/XML/`.
7. Rusty Object Notation `https://github.com/ron-rs/ron`.

8. To access your TeX-related documentation just type `texdoc` ⟨*package*⟩ in a terminal session. For instance, `texdoc luatex`.

Table 1: Direct links to the downloadable files of the MongoDB Community Server package available at the time of writing for the main 64 bit desktop OS.

OS identifier/Direct download link
Ubuntu 18.04 LTS 64 bit `https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-ubuntu1804-4.0.1.tgz`
Windows 64 bit installer `https://fastdl.mongodb.org/win32/mongodb-win32-x86_64-2008plus-ssl-4.0.1-signed.msi`
OSX 64 bit `https://fastdl.mongodb.org/osx/mongodb-osx-ssl-x86_64-4.0.1.tgz`

the Community Server tab from the official MongoDB download web page `https://www.mongodb.com/download-center` and pick your choice selecting your preferred Operating System. You can also grab your installation package from the direct link shown in table 1.

Placing executables in a system directory is safer due to the superuser rights protection, but for testing operations I prefer to temporarily keep the binary as well as the database files in my home directory:

```
$ tar -zxvf \
  mongodb-linux-x86_64-ubuntu1804-4.0.1.tgz
$ cp -r \
  ./mongodb-linux-x86_64-ubuntu1804-4.0.1 \
  ~/mongodb
$ mkdir ~/mongodb/data
```

The following commands make the server start on the localhost network address with no user authentication—in production environments a safe access to the server must be configured:

```
$ cd ~/mongodb/bin
$ ./mongod --smallfiles --dbpath ~/mongodb/data
```

Carefully look at the output produced by `mongod`: you can check if the system is up and correctly running. To safely stop the server press CTRL+C.

It's also recommended to install a copy of the Compass Community Edition GUI client, looking for related tab form in the download MongoDB web site. In figure 1 is shown a screenshot of Compass.

4.2 Compiling MongoDB C Driver

As a preliminary step, we need to install some packages from the Ubuntu repository. These shell commands do the job:

```
$ sudo apt-get install build-essential cmake
$ sudo apt-get install liblua5.2-dev
$ sudo apt-get install libssl-dev libsasl2-dev
```

As reported in the installation guide of the MongoDB C driver, download, unzip and prepare the project with:

```
$ wget \
  https://github.com/mongodb/mongo-c-driver\
  /releases/download/1.12.0\
  /mongo-c-driver-1.12.0.tar.gz
$ tar xzf mongo-c-driver-1.12.0.tar.gz
$ cd mongo-c-driver-1.12.0
$ mkdir cmake-build
$ cd cmake-build
$ cmake \
  -DENABLE_AUTOMATIC_INIT_AND_CLEANUP=OFF ..
```

If everything is OK, you may go on and install the driver—make sure that the working current directory is `cmake-build`—:

```
$ make
$ sudo make install
```

4.3 Installing LuaRocks

LuaRocks instruction from its website `https://luarocks.org/` are quite clear:

```
$ wget https://luarocks.org/releases\
  /luarocks-3.0.0.tar.gz
$ tar zxpf luarocks-3.0.0.tar.gz
$ cd luarocks-3.0.0
$ ./configure; sudo make bootstrap
```

4.4 Installing lua-mongo bindings

The very last shell command is:

```
$ sudo luarocks install lua-mongo
```

4.5 Post-install adjustment

To provide *visibility* to the Lua MongoDB binding from within LuaTEX, the library file could be copied in the `bin` directory of the main TeX Live tree, accordingly to the system variable `$CLUAINPUTS`. In fact, it contains paths where the typesetting engines looks for the C module.

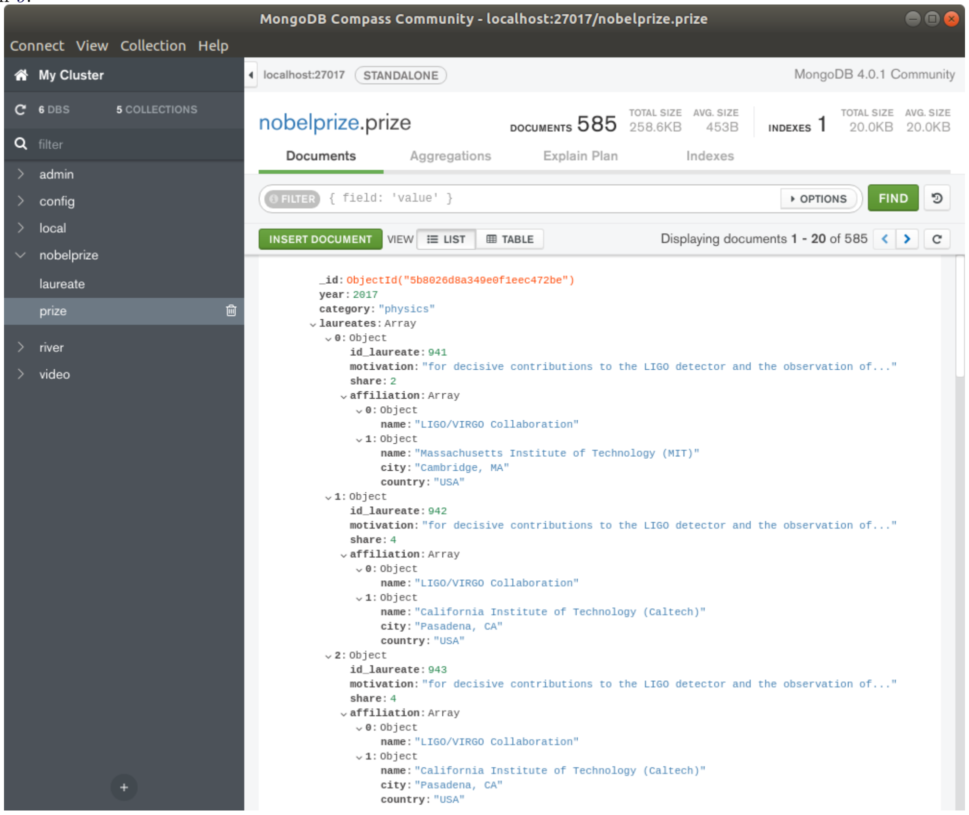
Indeed the LuaTEX manual reports the variable value as the following pattern:

```
CLUAINPUTS= .:\
  $SELFAUTOLOC/lib/{$progname,$engine,}/lua//
```

The first directory in the pattern is simply the folder containing the `.tex` source file itself, identified by the dot char '.', but what about the next `$SELFAUTOLOC` variable? To find out the answer we can address LuaTEX itself, compiling the source file below:[9]

9. We obtain the same answer if we invoke the command `kpsewhich -var-value= $SELFAUTOLOC` from the terminal.

Figure 1: A screenshot of Compass Community Edition GUI client running on Ubuntu 18.04 while connected to the local MongoDB server. It shows a document of the `prize` collection belonging to the `nobelprize` database, part of the project defined in section 9.



```
_id: ObjectId("5b8026d8a349e0f1eec472be")
year: 2017
category: "physics"
laureates: Array
  0: Object
    id_laureate: 941
    motivation: "for decisive contributions to the LIGO detector and the observation of..."
    share: 2
    affiliation: Array
      0: Object
        name: "LIGO/VIRGO Collaboration"
      1: Object
        name: "Massachusetts Institute of Technology (MIT)"
        city: "Cambridge, MA"
        country: "USA"
  1: Object
    id_laureate: 942
    motivation: "for decisive contributions to the LIGO detector and the observation of..."
    share: 4
    affiliation: Array
      0: Object
        name: "LIGO/VIRGO Collaboration"
      1: Object
        name: "California Institute of Technology (Caltech)"
        city: "Pasadena, CA"
        country: "USA"
  2: Object
    id_laureate: 943
    motivation: "for decisive contributions to the LIGO detector and the observation of..."
    share: 4
    affiliation: Array
      0: Object
        name: "LIGO/VIRGO Collaboration"
      1: Object
        name: "California Institute of Technology (Caltech)"
        city: "Pasadena, CA"
        country: "USA"
```

```
% !TeX program = LuaTeX
\directlua{
  print(kpse.expand_var [[$SELFAUTOLOC]])
}
\bye
```

that prints on the standard output the variable expansion as:

```
/usr/local/texlive/2018/bin/x86_64-linux
```

Essentially, C modules home can be only a platform-specific directory in the main TDS tree[10], and this in far from the ideal solution because the main tree is something that users should leave untouched, giving out local or personal tree of distribution to settle what is not officially part of TEX Live.

A workaround may be directly using the loader internally called by the Lua function `require()`, that is the standard way to load a module. `require()` has a default list of search paths included those derived from the pattern in the `$CLUAINPUTS` variable.

While we will discuss this in the section 6, let's now complete the installation: we only have to copy the C module in the local tree of TEX Live with the command:

10. For more information on TDS tree structure run the shell command `texdoc tds`.

```
$ cd /usr/local/texlive/texmf-local
$ sudo mkdir -p ./scripts/mongo
$ sudo cp /usr/local/lib/lua/5.2/mongo.so \
  ./scripts/mongo/
```

5 Installing on Windows

Since this task is slightly long, you may find comfortable to help yourself some coffee or tea, cookies and so on, and relax. Let's start installing Visual Studio Community latest release, currently 2017, from Microsoft website `https://visualstudio.microsoft.com/en/downloads/`. If necessary, register your Visual Studio copy submitting an account from the dedicated command you find in the Help menu.

Later, it will be useful Microsoft Build Tools 2015 too, so install it from Microsoft Download Center but only *after* installing Visual Studio.

You also need to install `cmake` from `https://cmake.org/download/`. You should have a 64 bit OS so choose win64-x64 architecture and not win32-x86 one.

5.1 Installing MongoDB

The Windows installer—the `.msi` file reported in the table 1—gives you two different kinds of configuration: running MongoDB as a service or not. The second option is recommended for testing and local project according to our experimental purpose. In any case, you may wish to install the Compass utility too, a GUI client for MongoDB databases. If so, check on the related flag in the Setup Wizard and press the Next button to enrich the final step.

To locally start the server, run in a console window the following commands:

```
> mkdir C:\mongodb-data
> cd 'C:\Program Files\MongoDB\Server\4.0\bin\'
> .\mongod --smallfiles --dbpath C:\mongodb-data
```

Check the output to assess if all is OK, then stop the server pressing CTRL+C to send a shutdown signal to the process.

5.2 Compiling MongoDB C Driver

Download the source code of the latest version of the C driver from `http://mongoc.org/#download`, current release is 1.12.0, unzip the file `mongo-c-driver-1.12.0.tar.gz`, then open a console on the uncompressed folder.

According to the documentation, the commands to create a Visual Studio 2017 project are the following (the double dot in the last line is important):

```
> cd mongo-c-driver-1.12.0
> mkdir cmake-build
> cd cmake-build
> cmake -G "Visual Studio 15 2017" \
  "-DCMAKE_INSTALL_PREFIX=C:\mongo-c-driver" \
  "-DCMAKE_PREFIX_PATH=C:\mongo-c-driver" \
  ..
```

If you don't know how to split the last command into several lines inside the shell, simply type it in a single line.

Since the `luatex` executable is a 32 bit x86 binary, make sure Visual Studio is set for `Win32` platforms before compiling. To inspect which architecture `luatex` is compiled for, digit the shell command:

```
> luatex --version
```

If the answer is (or looks like)

```
This is LuaTeX, Version 1.07.0 \
  (TeX Live 2018/W32TeX)
```

it's evident that we have a `Win32` compliant program.

Back to the source folder, open the generated file `mongo-c-driver.sln` in Visual Studio, select into the explorer the solution named `ALL_BUILD`, right click on it and run the command *Compile*. When the process ends, repeat the same steps with the solution named `INSTALL`. At this point you should see in `C:\mongo-c-driver` directory the files you were waiting for.

Add to the system PATH variable the folder `C:\mongo-c-driver\bin`. This allows Windows to find the binary. In order to easily open the dialog, type in the search bar "system environment variable" and select the proper icon.

5.3 Installing Lua Libraries

Pursuing a Lua driver installation for MongoDB, make sure you have Lua itself installed with the same version of LuaTEX, as explained before. If not, it is very simple, thanks to the project Lua Binaries, getting that files.

Browse the web page `http://luabinaries.sourceforge.net/` and pick up Lua 5.2.4 release pages hosted on SourceForge. Click in sequence on the folder `Windows`, hence `Dynamic`, and download the file `lua-5.2.4_Win32_dll15_lib.zip`. The file name tells whether the platform we have chosen is correct: `Win32` shows that library is for 32 bit binary and `dll15` is about the latest Visual Studio version.

Please make sure you have clicked on the `Dynamic` and not on the `Static` link or you will suffer soon or later the error *Multiple Lua VM's detected*. As a comparison, in the file name you must read the string `dll`, standing for dynamic linking library in the Microsoft ecosystem terminology.

Unpack the files and copy all of them in the folder, say `C:\Lua`, as a name easy to trace and remember. That folder will be the home of LuaRocks too, the next main character of our "play".

From the same repository, download a Lua interpreter clicking on the link folder `Tools Executables`. The file should be `lua-5.2.4_Win32_bin.zip`. Unzip it in the same directory where you are going to place the libraries file.

Add to the system `PATH` variable the folder `C:\Lua`, then run the interpreter.

5.4 Installing LuaRocks

The starting point for the Lua most famous package manager is the official website `https://luarocks.org/`. The latest release is 3.0.1 but at the moment it is unable to complete the installation process due to a dynamic linking failure. Fall back on 2.4.4 version downloading the file `luarocks-2.4.4-win32.zip` from `http://luarocks.github.io/luarocks/releases/`.

Unzip it and run this command:

```
> .\install.bat /P 'C:\Lua\Luarocks244' \
  /CONFIG 'C:\Lua\Luarocks244'
```

The option `/P` gives the location where LuaRocks will be installed, while `/CONFIG` gives the location of the configuration file.

As it should be clear now, add to the user variable the entries `LUA_PATH` and `LUA_CPATH` as follow (type the first, that has two paths, in one line):

```
LUA_PATH = C:\Lua\systree\share\lua\5.2\?.lua;
C:\Lua\systree\share\lua\5.2\?\init.lua

LUA_CPATH = C:\Lua\systree\lib\lua\5.2\?.dll
```

We have to append the path of the MongoDB C driver binaries

```
external_deps_dirs = {
    ”c:/mongo-c-driver”,
}
```

in the LuaRocks configuration file config-5.2.lua.

5.5 Installing the lua-mongo binding

Navigate to the folder C:\Program Files (x86) Microsoft Visual C++ Build Tools. Right click on “Visual C++ 2015 x86 Native Build Tools Command Prompt”, and select “Run as administrator”, then in the opened console window, run the following command:

```
”C:\Program Files (x86)\LuaRocks\luarocks.bat” \
  install lua-mongo
```

In order to allow LuaTEX to load the driver using the strategy explained in section 4.5, as the very last commands run the following sequence:

```
> cd C:\texlive/texmf-local
> mkdir scripts
> cd scripts
> mkdir mongo
> cd mongo
> copy ”C:\Lua\systree\lib\lua\5.2\mongo.dll”
```

6 Testing

In the end, the connector binary file is in the local tree, more specifically, in the sub-directory $TEXMFLOCAL/scripts/mongo. It can’t be loaded by the usual Lua function due to the restricted list of search path set for require(), as explained in the section 4.5.

The Lua function package.loadlib fits our case. It returns the initialization function of a C module written according to the Lua C API, that returns the actual library. The expected arguments are the path to the file and the name of the initialization function beginning with luaopen_.

Instead of hard-coding the library path, a more general option is to call the LuaTEX specific function expand_var() from the kpathsea utility module; in this way we get the value of the $TEXMFLOCAL variable and take into account the file extension depending on the Operating System—.dll for Windows, .so for other OS—calling the function os.type().

The code that implements such solution is the following:

```
% !TeX program = LuaTeX
\directlua{
  local path = kpse.expand_var [[$TEXMFLOCAL]]
  assert(path)
  if os.type == [[windows]] then
      path = path..[[/scripts/mongo/mongo.dll]]
  else
      path = path..[[/scripts/mongo/mongo.so]]
  end
  local _mongo = package.loadlib(
      path,
      ”luaopen_mongo”
    )
  assert(_mongo)
  local mongo = assert(_mongo())
  % print the available functions
  local par = string.char(92)..[[par]]
  for k, v in pairs(mongo) do
    if type(v) == [[function]] then
      tex.print(k..[[()]]..par)
    end
  end
}
\bye
```

As a general test, that .tex file prints also in the PDF file the list of the functions available in the connector lua-mongo. If the compilation process ends correctly, the minimal test can be considered passed and the functions list appearing regardless of the order should be the following:

```
Double()
Int64()
Decimal128()
Regex()
BSON()
Timestamp()
Binary()
DateTime()
ObjectID()
Int32()
ReadPrefs()
type()
Client()
Javascript()
```

That list can be compared with the API documentation of the project at the URL https://github.com/neoxic/lua-mongo/blob/master/doc/main.md.

7 LuaTEX application

In the next two sections I present example projects involving operations on a MongoDB database and queries execution from within LuaTEX in order to typeset reports.

Be sure you have a mongod server instance locally up and running while you compile with LuaTEX. So, along with a modern TEX distribution, you have installed several components in order to compile the source code examples. In case you didn’t do that yet, please head to section 3 for details.

All of the project files are downloadable from ArsTEXnica home page at https://www.guitex.org/home/it/arstexnica. Browse issues online page to find out the link pointing to the related compressed archive named mongodb-project.zip.

8 Rivers

The first example models a very simple dataset regarding the longest rivers in the United States[11], including a one-to-many relationship: alongside basic information like river name and length, there is the list of the crossed states too.

The dataset would be represented in a SQL schema with three different tables:

- table `river` with the basic information;
- table `regions` defining states names;
- a pure relational table `crossing` connecting rivers with states through a pair of foreign key identifiers.

On the contrary, thanks to the recursive nature and compound data types of MongoDB documents, we can represent rivers and crossed regions by them with only one document, within a single collection.

A document may contain the list of the rivers regions as an array assigned to the field `regions` as the following:

```
{
    ”name” : ”Missouri River”,
    ”length_km” : 3768,
    ”regions” : [
        ”Montana”,
        ”North Dakota”,
        ”South Dakota”,
        ”Nebraska”,
        ”Iowa”,
        ”Kansas”,
        ”Missouri”
    ]
}
```

We can even enrich the previous model because the regions may be represented as an array of embedded documents as showed in the listing below:

```
{ // embedded document version
    ”name” : ”Missouri River”,
    ”length_km” : 3768,
    ”regions” : [
        {
            ”state”:”Montana”,
            ”code”:”MT”,
            ”shortname”:”Mont.”
        },
        {
            ”state”:”North Dakota”,
            ”code” :”ND”,
            ”shortname”:”N. Dak.”
        },
        {
            ”state”:”South Dakota”,
            ”code” :”SD”,
            ”shortname”:”S. Dak.”
        },
        {
            ”state”:”Nebraska”,
            ”code” :”NE”,
            ”shortname”:”Nebr.”
        },
        {
            ”state”:”Iowa”,
            ”code” :”IA”,
            ”shortname”:”Iowa”
        },
        {
            ”state”:”Kansas”,
            ”code” :”KS”,
            ”shortname”:”Kans.”
        },
        {
            ”state”:”Missouri,
            ”code” :”MO”,
            ”shortname”:”Mo.”
        }
    ]
}
```

11. Data source `https://en.wikipedia.org/wiki/List_of_longest_rivers_of_the_United_States_(by_main_stem)`.

In MongoDB we may decide to *embed* or to *reference* data. While the first technique is represented by the latest river model, the second consists in just including the reference to the region documents instead of the documents themselves. Those references have the form of `_id` values of the corresponding documents in a different and unspecified collection.

Deciding which is the best solution depends on the performance of the application context and not on the normalization criteria that leads the design of SQL schemas.

8.1 Making the database

In order to create the database, the simplest method is to write a JavaScript file `river.js` to be executed in the `mongo` shell with the command:

```
$ mongo --host localhost river.js
```

The code must define names, collection, and rivers documents, structured as formerly seen in the first model for the longest river in the USA, where regions are a simple list of strings. The script `river.js` can be illustrated by the listing below where a three dots sequence shortens lines but not the essence:

```
// get/define database object
db = db.getSiblingDB(”river”);

// data insertion
db.generalinfo.insertMany([
    {...}, {...}, {...}, ...
])
```

The pre-instantiated `db` variable refers to the current database. The method `getSiblingDB()` of the db type returns a pointer the requested database which will be created if it does not exist. The next expression is a chain where in the first stage the collection `generalinfo` of the database pointed by db is returned—once again, if the collection doesn't exist it will be created—then its method `insertMany()` will write an array of documents.

Too many words to explain such a few lines of smart and practical code. `mongo` shell is inspired by SQL CLI clients like `psql` in PostgreSQL, but its JavaScript library is designed with the following motto in mind: *learn and try* then *configure*.

8.2 Querying with mongo shell

It is a very common way in MongoDB that methods performing operations on a database take as an argument a document object. For instance, in `mongo` shell you can retrieve rivers that are exactly 2000 km long by the following query:

```
> use river
switched to db river
> db.generalinfo.find({"length_km": 2000})
```

Querying that on our dataset we obtain the following document:

```
{
   "_id" : ObjectId("5b7af5a589923a94e7ce4cb0"),
   "name" : "Columbia River",
   "length_km" : 2000,
   "regions" : [
       "British Columbia",
       "Washington",
       "Oregon"
   ]
}
```

As expected, MongoDB has automatically added the mandatory `_id` field to each document inserted by the `river.js` script.

8.3 This looks like a job for LuaLATEX

Our goal is to typeset the tabular shown in table 2. This is what we have to do:

1. create a client that has established a connection with the MongoDB server;
2. get the collection object `generalinfo` of the `river` database;
3. perform the query to retrieve all the rivers information;
4. typeset the table with a `tabular` environment.

All these steps are straightforward. To keep the `.tex` source code simple and a little bit more abstract, I'm going to write an auxiliary Lua library in the external file `libmongo.lua` to be located in the same directory of the main source file.

The constructor `Connect()` takes as an argument the database name, then establishes a client connection to the MongoDB server on `localhost`.

The real job is demanded to the Lua binding of MongoDB C Driver, compiled as described in sections 4 (for Ubuntu)–5 (for Windows). The loading mechanism of the binding library is explained in section 6.

No further functions but the method `FindIn()` are required. In the previous task list, it performs the 2nd and 3rd tasks and the Lua array with all the documents selected by the query is returned.

The `libmongo.lua` code is the following:

```
local lib = {}
lib.__index = lib
-- constructor
function lib:Connect(dbname)
    assert(type(dbname)=="string")
    local path = kpse.expand_var [[$TEXMFLOCAL]]
    assert(path)
    if os.type == [[windows]] then
       path = path..[[/scripts/mongo/mongo.dll]]
    else
       path = path..[[/scripts/mongo/mongo.so]]
    end
    local _mongo = package.loadlib(
        path,"luaopen_mongo"
    )
    assert(_mongo)
    local mongo = assert(_mongo())
    local client = assert(
        mongo.Client("mongodb://127.0.0.1")
    )
    local o = {
        _mongo  = mongo,
        _client = client,
        _dbname = dbname,
    }
    setmetatable(o, self)
    return o
end
-- query method
function lib:FindIn(
    collection, query, option, prefs
    )
    assert(type(collection) == "string")
    local client = self._client
    local coll = client:getCollection(
        self._dbname, collection
    )
    local data = {}
    query = query or {}
    for doc in coll:find(query, option, prefs)
                   :iterator() do
        data[#data + 1] = doc
    end
    return data
end

return lib
```

According to the object oriented paradigm in Lua, we have to use the *colon notation* and not the dot operator to call methods. If these sentence sounds weird to you, please refer to the book (Ierusalimschy, 2016) and read the chapter dedicated to metamethods and metatables and the chapter dedicated to Object Oriented Programming.

Table 2: The longest United States of America's rivers as reported by Wikipedia. The table is typeset by LuaTEX directly connecting to a MongoDB database as explained step by step in the text.

River name	Length (km)	Regions
Missouri River	3768	Montana, North Dakota, South Dakota, Nebraska, Iowa, Kansas, Missouri
Mississippi River	3544	Minnesota, Wisconsin, Iowa, Illinois, Missouri, Kentucky, Tennessee, Arkansas, Mississippi, Louisiana
Yukon River	3185	British Columbia, Yukon Territory, Alaska
Rio Grande	2830	Colorado, New Mexico, Texas, Chihuahua, Coahuila, Nuevo León, Tamaulipas
Colorado River	2330	Colorado, Utah, Arizona, Nevada, California, Sonora, Baja California
Arkansas River	2322	Colorado, Kansas, Oklahoma, Arkansas
Columbia River	2000	British Columbia, Washington, Oregon
Red River	1811	Oklahoma, Texas, Arkansas, Louisiana
Snake River	1674	Wyoming, Idaho, Oregon, Washington
Ohio River	1575	Pennsylvania, Ohio, West Virginia, Indiana, Illinois, Kentucky

The last task—number 4 in the list—is reserved to the LuaLATEX source file itself:

```
% !TeX program = LuaLaTeX
\documentclass{standalone}
\usepackage{fontspec}
\defaultfontfeatures{Ligatures=TeX}
\setmainfont{CMU Serif}
\usepackage{booktabs}

\directlua{
    local libmongo = require ”libmongo”
    local db = libmongo:Connect(”river”)
    river = db:FindIn(”generalinfo”)
}
\begin{document}
\begin{tabular}{lcp{120mm}}
\toprule
River name & Length (km) & Regions\\
\midrule
    \directlua{
    local stop = string.char(92)
    stop = stop..stop
    for _, r in ipairs(river) do
        tex.print(r.name)
        tex.print(”&”)
        tex.print(r.length_km)
        tex.print(”&”)
        tex.print(table.concat(r.regions, ”, ”))
        tex.print(stop)
    end
    }
\bottomrule
\end{tabular}
\end{document}
```

Reading this code, two elements worth a comment: first of all, the field `regions` are an array of strings as in the documents, so the Lua binding uses the Lua table type as the direct incarnation of the MongoDB document: in fact, the variable `r` in every cycle has the keys `name` and `length_km` as well as the key `regions` that refers to a further Lua table in the array form. The Lua standard function `table.concat()` provides the string corresponding to the comma separated values of States and provinces crossed by river.

As the second interesting point, the query returns documents in the same order of the original insertion, which is the order of the Wikipedia table at the time I wrote the `river.js` script.

An optional parameter allows us to explicitly set the order, sorting documents by value. Try to edit the previous code as in:

```
\directlua{
    local libmongo = require ”libmongo”
    local db = libmongo:Connect(”river”)
    river = db:FindIn(
        ”generalinfo”,
        {},
        {sort = {length_km = 1}}
    )
}
```

The integer 1 specifies ascending order while -1 specifies descending order, in our case applied to the rivers length.

Furthermore, special query document fields named *operators* act as conditional parameters being part of the MongoDB query language. They have a dollar sign as their first character, like in `$gte`—that stands for *greater or equal* or $\geq$. For instance, in the previous code we can select only those rivers that have a length greater or equal to 2000 km, adding a specific query document as the second argument of the `FindIn()` method:

```
river = db:FindIn(”generalinfo”,
    { length_km = { [”$gte”] = 2000 }}
)
```

or even by a length range like in:

```
river = db:FindIn(”generalinfo”,
   length_km = {[”$gte”] = 2000, [”$lte”]=3000}}
)
```

We can execute the same query in the `mongo` shell as in the following session:

```
> use river
switched to db river
> db.generalinfo.find({”length_km”:
... {”$gte”:2000, ”$lte”:3000}
... })
```

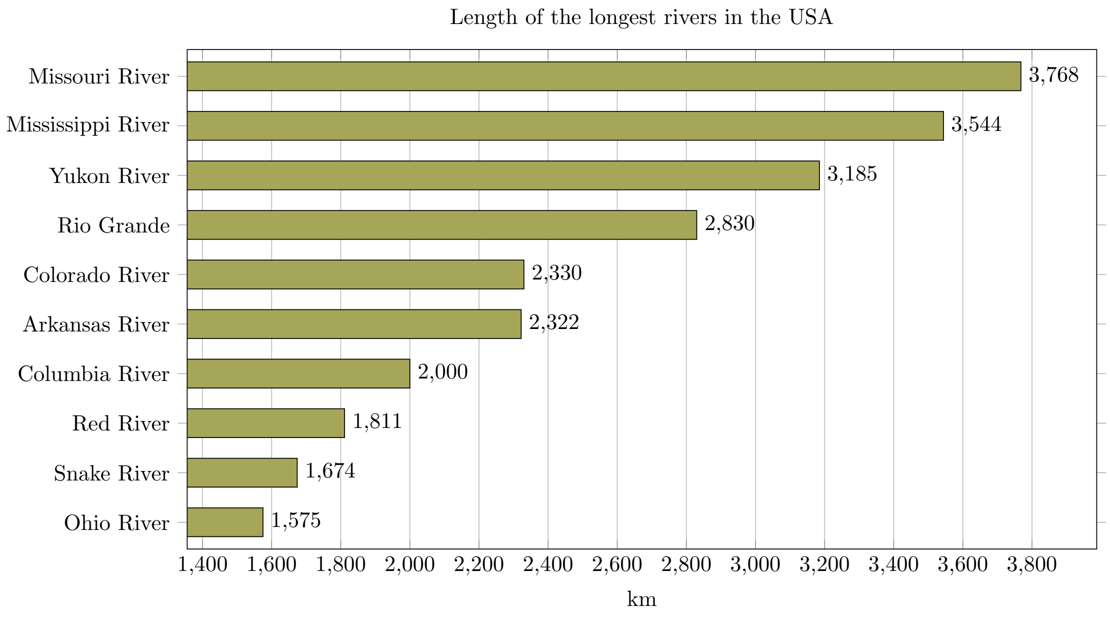


FIGURE 2: The longest United States of America's rivers as reported by Wikipedia. The histogram plot is typeset by LuaTEX performing a direct connection to a MongoDB database as explained step by step in the text.

8.4 Histogram

Once data is retrieved from MongoDB and stored in Lua tables, LuaTEX can typesets information in whatever form the user requires, e.g., a histogram plot. The figure 2 is the PDF output of the following source file where pgfplots package plots rivers length:

```
% !TeX program = LuaLaTeX

\documentclass[margin=2pt]{standalone}
\usepackage{fontspec}
\defaultfontfeatures{Ligatures=TeX}
\setmainfont{CMU Serif}

\usepackage{pgfplots}
\pgfplotsset{compat=1.16}

\directlua{
    local libmongo = require "libmongo"
    local db = libmongo:Connect("river")
    river = db:FindIn("generalinfo", {},
        {sort={length_km = 1}}
    )
}
\newcommand\coords{\directlua{
    for _, riv in ipairs(river) do
    tex.print(
        "("..riv.length_km..","..riv.name..")"
    )
    end
}}
\newcommand\riverlist{\directlua{
    local t = {}
    for i, riv in ipairs(river) do
        t[i] = riv.name
    end
    tex.print(table.concat(t, ","))
}}

\begin{document}
\begin{tikzpicture}
\begin{axis}[
    xbar,
    bar width=13pt,
    width=16cm,
    height=9.5cm,
    enlarge y limits=0.060,
    xmajorgrids,
    title={%
    Length of the longest rivers in the USA},
    xlabel={km},
    symbolic y coords/.expanded={\riverlist},
    ytick=data,
    nodes near coords,
]
\addplot[draw=black,fill=blue!35!yellow]
    coordinates {\coords};

\end{axis}
\end{tikzpicture}
\end{document}
```

9 Nobel Prize

Appending to the URL `http://api.nobelprize.org/v1/` one of the strings `laureate.json`, `prize.json` or `country.json`, everyone can download data regarding Nobel Prizes in JSON format.

For more information about Nobel Prize API please visit the website `https://www.nobelprize.org/about/developer-zone-2/`.

This dataset, freely available from the Nobel Foundation, contains many-to-many relationships, such as people who won more than one Nobel Prize and single Nobel Prizes won by more than one person.

That problem is complex enough to deserve an interesting project to be developped in MongoDB: how should we define data models for such an admirable dataset?

9.1 Modelling documents

Instead of embedding data in a single structured document, we will consider a pair of distinct models representing *laureate* and *prize*.

A listing is worth a thousand words, so let start with a possible laureate document:

```
{ // basic model for laureate
    ”_id”: 1,
    ”firstname”: ”Wilhelm Conrad”,
    ”surname”: ”Röntgen”,
    ”born”: new Date(”1845-03-27”),
    ”died”: new Date(”1923-02-10”),
    ”bornCountry”: ”Prussia (now Germany)”,
    ”bornCountryCode”: ”DE”,
    ”bornCity”: ”Lennep (now Remscheid)”,
    ”diedCountry”: ”Germany”,
    ”diedCountryCode”: ”DE”,
    ”diedCity”: ”Munich”,
    ”gender”: ”male”
}
```

The fields `born` and `died` have type `Date`—they are instantiated objects—and the remaining fields, but the document identifier `_id` which is an integer, have type `string`.

A very simple model after all, and also flexible: for instance, if the laureate is alive, the field `died` does not exist as well as `diedCountry`. Furthermore, laureate can be an organization without any first name.

A more structured model could represent the prize:

```
{ // basic model for prize
    ”year”: 1901,
    ”category”: ”physics”,
    ”laureates”: [
      {
        ”id_laureate”: 1,
        ”motivation”: ”in recognition of ...”,
        ”share”: 1,
        ”affiliation”: [
          {
            ”name”: ”Munich University”,
            ”city”: ”Munich”,
            ”country”: ”Germany”
          }
        ]
      }
    ]
}
```

The field `laureates` is an array of documents, each one representing a winner referenced through the field `id_laureate`. The field `affiliation`—one more time—is an array because scientists can have more than one affiliation, for instance when they are part of world-wide research program in addition to their Academic Institution.

But why affiliations fields aren't in the laureate model? After all Nobel Prize is unrelated to the institution where laureates carry out their studies.

This is right but what about the case of a laureate who won more than one prize, each one being afferent to different institutes or organizations? It is a very uncommon event, but we can't forget the case of Marie Curie.

In fact, affiliation is something that may change during life, so we have to trace such a piece of information while the Nobel Prize is awarded.

The introduced *basic model* fulfills that requirementin a practical way in spite of coherence. A different possible way is to reference a third referenced document `affiliation` that references `laureate` such as:

```
{ // alternative prize model
    ”year”: 1901,
    ”category”: ”physics”,
    ”laureates”: [
      {
        ”id_affiliation”: 100, // reference
        ”motivation”: ”in recognition of ...”,
        ”share”: 1,
      }
    ]
}
```

and

```
{ // alternative affiliation model
    ”_id”: 100,
    ”id_laureate”: 1,
    ”affiliation”: [
        {
          ”name”: ”Munich University”,
          ”city”: ”Munich”,
          ”country”: ”Germany”
        }
    ]
}
```

while the laureate document remains the same.

These alternative models require at least three queries to know who won a specific prize: the first one to retrieve the affiliation identifier, another one to query the affiliation document to get the laureate identifier, and finally to query the laureate document. Vice versa the same task with the basic model takes only one query for the prize document and one for laureate document.

Talking about the alternative three-parted model, it is rather interesting the document reference scheme for people who won the prize twice: there will be two prizes referencing two different affiliations, each one referencing the same laureate: a link figure similar to an upper case V.

From now on I will adopt the basic model counting the collections `laureate` and `prize` within the `nobelprize` database. It is out of the scope of this work an in-deep exploration of design pattern for MongoDB applications. Nevertheless, it is important to show an example of document referencing as counterpart of document embedding. Further information on design pattern application in MongoDB can be found in the book (Copeland, 2013) from O'Reilly Media.

MongoDB doesn't offer fast join functions between tables as in SQL. From the server point of view referencing documents means a larger number of queries when data are required, while embedding document means a larger number of updating when data are changed. Pros and cons assessment is focused on server workload rather than data coherence, as in the case of high volume website or big data archive.

9.2 Data validation

What I didn't know about the Nobel Prize was the meaning of the field `share` in the `prize` document. If the Nobel Prize is won by more than one laureate, they share the prize in proportion and not always in equal parts. As an example, if three laureates share the prize respectively with 2, 4, 4 quote, the first one deserves one half of the prize while the others deserve a quarter each.

I'm not digressing. It should be true that the sum of the inverse of each prize `share` values is exactly equal to 1. Prior MongoDB 3.2 version, client-side validation was the only option to find out errors that MongoDB wouldn't have discovered in absence of any server-side constraints usually active in a SQL schema.

Recent versions of MongoDB are able to validate documents before update or insert operations concerning a single collection. For instance, we can validate every values types, the existence of a mandatory field and the binding of a value to an enumeration.

As an example on such server-side validation, we can explicitly create the laureate collection adding a validator as an object with the `$jsonSchema` operator. The following JavaScript code shows an example:

```
db.createCollection(”laureate”, {
   validator: {
      $jsonSchema: {
         bsonType: ”object”,
         required: [ ”surname”, ”gender” ],
         properties: {
            firstname: {
               bsonType: ”string”,
               description: ”must be a string”
            },
            surname: {
               bsonType: ”string”,
               description: ”must be a string”
            },
            gender: {
               enum: [”female”, ”male”, ”org”],
               description: ”enum values only”
            },
         }
      }
   }
})
```

Validator may be also a query filter expression, generally less expressive than a `$jsonSchema` operator.

Server-side validation is only a condition, that can be true or false, about acceptability of documents whereas client can even edit a non-compliant document according to specific rules, instead of rejecting such objects.

In my opinion, in most cases it is desirable to add a server-side validator to every collection. Anyway in the next section there will be an example of client-side validation, mainly to provide a very helpful method on how to execute JavaScript code.

9.3 A Method for client-side data validation

The basic model defined in the previous section requires to apply several adjustments to the original JSON file spread by Nobel Prize Foundation. With a mix of regular expressions and JavaScript code, I have translated the original files into the structure required by the basic model. The final result consists of two JavaScript scripts called `laureates.js` and `prizes.js`.

Each script contains only one assignment: a variable takes a literal array of objects. I'm going to show how to load these files very soon. In a compact syntax the laureates script may be written as:

```
// laureates.js
var laureate = [{...}, {...}, ...];
```

Now suppose such variables—`laureate` and `prize`—are available in a JavaScript environment: we could check everything about dataset integrity and coherence by means of the full featured JavaScript programming language. For instance, to ensure that gender fields values are only limited to the strings `male`, `female` or `org`, we can iterate over laureate array and check if every gender value is in a hash map as in the code below:

```
var gender = Object.create({
    ”male”  : true, // boolean values are unused
    ”female”: true,
    ”org”   : true
});

for (l of laureate) {
    if (!(l.gender in gender)) {
        print(”’”+l.gender+”’ not allowed”)
    }
}
```

While it will be MongoDB to check uniqueness of the field `_id` within the laureate collection, we are allowed to check dataset references: prize document must have valid laureates identifier:

```
// checking laureate reference
var idx = Object.create({});
for (l of laureate) {
    idx[l._id.toString()] = true;
}
for (p of prize) {
    for (l of p.laureates) {
        var id = l.id_laureate.toString();
        if (!(id in idx)) {
            print("wrong laureate id")
        }
    }
}
```

and it is also not hard to check the sharing quotes of prizes:

```
function okShare(share) {
    var num = 0; // numerator
    var den = 1; // denominator
    var i;
    for (i = 0; i < share.length; i++) {
        den *= share[i];
        var k;
        var part = 1;
        for (k = 0; k < share.length; k++) {
            if ( i != k ) {
                part *= share[k];
            }
        }
        num += part;
    }
    return num == den
}

// checking share fields
for (p of prize) {
    var vshare = [];
    for (l of p.laureates) {
        vshare.push(l.share)
    }
    if (!okShare(vshare)) {
        print("wrong share group")
    }
}
```

9.4 Populating the database

Functions similar to those presented in the previous section helps to keep documents reliable. Assuming that all of that checking code is saved in the file `check.js`, `mongo` shell is able to execute such JavaScript files with the following command:

```
$ mongo --nodb laureates.js prizes.js check.js
MongoDB shell version v4.0.1
loading file: laureates.js
loading file: prizes.js
loading file: check.js
'laureate' document to control: 916
'prize' document to control: 585
Passed 'laureate' gender fields: 916/916
Passed 'prize' laureate reference: 585/585
Passed 'prize' share fields: 585/585
```

and finally load the dataset into database with the command:

```
$ mongo laureates.js prizes.js populate.js
MongoDB shell version v4.0.1
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 4.0.1
loading file: laureates.js
loading file: prizes.js
loading file: populate.js
```

where the code of the `populate.js` JavaScript file is the following:

```
// run as
// mongo laureates.js prizes.js populate.js

// get database object
db = db.getSiblingDB("nobelprize");

// insert laureates
db.laureate.insertMany(laureate)
// insert Nobel Prize
db.prize.insertMany(prize)
```

9.5 Asking LuaTEX about laureates

In our `libmongo.lua` auxiliary library introduced in section 8 we can add a new function named `FindOne()` especially for treating the case of a query returning only one document:

```
function
lib:FindOne(collection, query, option, prefs)
    assert(type(collection) == "string")
    local client = self._client
    local coll = client:getCollection(
        self._dbname, collection
    )
    query = query or {}
    return coll:findOne(query, option, prefs)
                :value()
end
```

As a proof of concept, we can try LuaLATEX to typeset basic information about laureates such as the birth date and place. The Lua table that corresponds to the query-generated document defines criteria with both `firstname` and `surname` fields:

```
local query = {
    firstname = "Enrico",
    surname   = "Fermi"
}
```

The complete listing below is pretty straightforward: once defined the Lua table `fermi` as a global variable, we can index it with the field key without any restriction in order to print the corresponding value everywhere in the report:

```
% !TeX program = LuaLaTeX
\documentclass{article}
```

```
\usepackage{fontspec}
\defaultfontfeatures{Ligatures=TeX}
\setmainfont{CMU Serif}

\directlua{
local libmongo = require ”libmongo”
local db = libmongo:Connect(”nobelprize”)
local query = {
    firstname = ”Enrico”,
    surname   = ”Fermi”
}
fermi = db:FindOne(”laureate”, query)
}

\newcommand\print[1]{\directlua{
    local expr = tostring(#1);
    if expr then
        tex.print(expr)
    end
}}

\newcommand\printdate[1]{\directlua{
    local d0 = #1;
    local d1 = d0:unpack()/1000
    local d2 = os.date([[*t]], d1)
    local m = {
        ”January”,
        ”February”,
        ”March”,
        ”April”,
        ”May”,
        ”June”,
        ”July”,
        ”August”,
        ”September”,
        ”October”,
        ”November”,
        ”December ”,
    }
    tex.print(
        tostring(d2.day), m[d2.month], d2.year
    )
}}

\begin{document}
\print{fermi.firstname} \print{fermi.surname}
was born in \print{fermi.bornCity} on
\printdate{fermi.born}.
\end{document}
```

The result is:

> Enrico Fermi was born in Rome, Italy, on 29 September 1901.

Dates are saved as objects and not as strings. It was our initial choice. After all objects are more clever than strings because they have specific methods. When dealing with dates, we can print only the year or even the days between two dates.

In spite of that, why the source code of our working minimal example compiles under Linux but not under Windows?

It's because dates are not yet a `Date` object. In fact, the `lua-mongo` driver returns dates in BSON format. Subsequently, in the macro `\printdate` defined in the code, we call the method `unpack()` to get the date as the number of milliseconds since the Epoch[12], the moment in time fixed to the midnight of 1 January 1970.

The Lua function of the standard library `os.date()` is able to determine date values starting from the number of seconds since the Epoch, and this is the reason why in the code a factor 1/1000 multiply the unpacked value of the BSON date object.

Unfortunately, the Windows system library that deals with dates returns a null pointer if the value in time units is negative, and this is what happens in the case of Enrico Fermi, who was born before 1 January 1970. As a consequence, `os.date()` returns `nil`.

The best solution is to transform BSON date object in Lua date object. In recent time I have aimed at the LuaDate project located at `http://tieske.github.io/date/`, but I'm aware, time counting is not so easy, and this project could miss the goal depending on your need.

9.6 Asking LuaTEX about Nobel Prizes

The table 3 represents the next target. It shows the list of Nobel Prizes awarded in 2017, with Category, Laureates and Motivation of the single Prize.

Our basic model is divided into two collections: `prize` and `laureate`. In the first collection a prize document defines, along with other pieces of information, the category and, for each laureate, an identifier that refers to the corresponding document in the second collection, a share value and a motivation.

To summarise, the query process consists in three different steps:

- query prizes awarded in 2017;
- for each selected prize, query each laureate for their complete names and share values;
- then determine the main motivation as the motivation of the laureate that has the most relevant share value or, two or more laureates have the same share, that comes first in the ordered array.

MongoDB has a powerful query language with aggregate operations, projections and so on. It's probably capable to return the desired result with only one query actually composed by a chain of operations. However, I will implement the job in Lua: this paper focus isn't on learning advanced MongoDB. It is also important to notice that the ideal query takes advantage of what the server-side has to offer.

12. For more information please visit the webpage `https://en.wikipedia.org/wiki/Unix_time`.

Table 3: The Nobel Prizes awarded in 2017. The table is typeset by LuaTEX performing a direct connection to a MongoDB database as explained in the paper.

2017	
Category	Laureates (share)/Main motivation
physics	Rainer Weiss (2), Barry C. Barish (4), Kip S. Thorne (4) for decisive contributions to the LIGO detector and the observation of gravitational waves
chemistry	Jacques Dubochet (3), Joachim Frank (3), Richard Henderson (3) for developing cryo-electron microscopy for the high-resolution structure determination of biomolecules in solution
medicine	Jeffrey C. Hall (3), Michael Rosbash (3), Michael W. Young (3) for their discoveries of molecular mechanisms controlling the circadian rhythm
literature	Kazuo Ishiguro (1) who, in novels of great emotional force, has uncovered the abyss beneath our illusory sense of connection with the world
peace	International Campaign to Abolish Nuclear Weapons (ICAN) (1) for its work to draw attention to the catastrophic humanitarian consequences of any use of nuclear weapons and for its ground-breaking efforts to achieve a treaty-based prohibition of such weapons
economics	Richard H. Thaler (1) for his contributions to behavioural economics

Finally, the complete code generating the table 3 is showed in the listing below. Improvements are left to the reader, such as to eliminate the redundant `\midrule` in the last row, to capitalize the category name, or to eliminate the share value when unnecessary, that is when there is no sharing at all:

```
% !TeX program = LuaLaTeX
\documentclass{standalone}
\usepackage{fontspec}
\defaultfontfeatures{Ligatures=TeX}
\setmainfont{CMU Serif}
\usepackage{booktabs}

\newcommand\YEAR{2017}

\directlua{
local libmongo = require ”libmongo”
local db = libmongo:Connect(”nobelprize”)
local query = {year = \YEAR}
prize = db:FindIn(”prize”, query)

for _, p in ipairs(prize) do
    local tl = {}
    local share, motivation
    for i, l in ipairs(p.laureates) do
        if share then
            if share < l.share then
                share = l.share
                motivation = l.motivation
            end
        else
            share = l.share
            motivation = l.motivation
        end
        local ql = {_id = l.id_laureate}
        linfo = db:FindOne(”laureate”, ql)
        tl[i] = linfo.firstname .. ” ” ..
                (linfo.surname or ””) ..
                ” (”..share..”)”
    end
    p.laureatelist = table.concat(tl, ”, ”)
    p.mainmotivation = motivation or ””
end
}

\newcommand\print[1]{\directlua{
    local expr = tostring(#1);
    if expr then
        tex.print(expr)
    end
}}

\begin{document}
\begin{tabular}{lp{140mm}}
\toprule
\textbf{\YEAR}\\
\midrule
Category & Laureates (share)/Main motivation\\
\midrule
\directlua{
local bs = string.char(92)
stop = bs..bs
for _, p in ipairs(prize) do
    tex.print(p.category)
    tex.print(”&”)
    tex.print(p.laureatelist)
    tex.print(stop)
    tex.print(”&”)
    tex.print(bs..”small ”)
    tex.print(p.mainmotivation)
    tex.print(stop)
    tex.print(bs..”midrule”)
end
}
\bottomrule
\end{tabular}
\end{document}
```

10 Alternative ways

Some alternative ways are suitable for experimenting or for self-contained project, and can be part of a step-by-step development strategy toward an HTTP or TCP MongoDB proxy server.

Different ways to query MongoDB databases from LuaTEX are alternative in the sense that they don't use neither a Lua low-level binding nor an intermediate network service. They rely on independent tools and an elementary form of communication between LuaTEX and data, that is essentially a file.

I'm referring to a file as a communication layer in two different ways:

static: the file is saved in the file system and contains data that are a query result, encoded in a declared format like JSON;

dynamic: textual result of a query is printed to the standard output and caught by Lua via the standard function `io.popen()`.

If the channel is static, then data files are file system objects, and the unique limitation is the precise knowledge of the exchange data format. If the channel is dynamic, it is even required that tools generating data are a CLI program.

For instance, the external tool can be the `mongo` shell or Node.js. Both execute a JavaScript file or even a Rust program taking advantage of drivers like the open source project `mongo-rust-driver-prototype` written in pure Rust and providing a native interface to MongoDB.

11 Conclusion

The first part of this paper (sections 3–5) has shown how to set up a project infrastructure relying on MongoDB as database storage and LuaTEX as reporting tool. The guide is detailed both for Ubuntu and Windows, and leads you to a step-by-step source code compilation in order to build a database connector suitable for the typesetting engine.

Low-level Lua binding to the C module isn't the only communication way with MongoDB. Intermediate network services via TCP connection protocol can be implemented for LuaTEX in order to simplify the components and to ensure reliability.

The second part of this paper (sections 8–9) has shown two demo projects from the very first step in designing data models and creating database via JavaScript to the working example in LuaTEX for generating reports.

MongoDB documents are expressive, flexible and make it easy to improve data models. Thanks to the similarity to the Lua tables, high-quality reports can be typeset by LuaTEX with less code than that required when the storage engine is a SQL relational database.

Further work is required to fully use MongoDB query language capabilities, especially for building project in real context, and to make a final decision about how to safely connect LuaTEX to MongoDB.

12 Acknowledgments

I would like to thank all of the ArsTEXnica team members who have supported this work with great effort. My gratitude goes also to my family.

References

Chodorow, K. (2013). *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage.* O'Reilly Media, 2ª edizione. URL http://shop.oreilly.com/product/0636920028031.do.

Copeland, R. (2013). *MongoDB Applied Design Patterns.* O'Reilly Media, 1ª edizione. URL http://shop.oreilly.com/product/0636920027041.do.

Giacomelli, R. (2017). «A database experiment with LuajitLATEX». *ArsTEXnica*, (23), pp. 12–34. URL http://www.guitex.org/home/numero-23.

Ierusalimschy, R. (2016). *Programming in Lua.* Lua.org, 4ª edizione.

The LuaTEX development team (2018). *LuaTEX Reference Manual.* Version 1.0.7.

▷ Roberto Giacomelli
Carrara
`giaconet dot mailbox at gmail dot com`