# THE BIG-R BOOK

## FROM DATA SCIENCE TO LEARNING MACHINES AND BIG DATA

## — PART 03—

*Dr. Philippe J.S. De Brouwer*

last compiled: September 1, 2021
Version 0.1.1

# *THE BIG R-BOOK:*

# *From Data Science to Big Data and Learning Machines*

## ♡– PART 03: Data Import –♡

These slides are to be used in with the book – for best experience, teachers will read the book *before* using the slides and students have access to the book and the code.

part 03: Data Import

↓

chapter 12:

# A Short History of Modern Database Systems

# Short History of Databases

1. tally sticks (since the Upper Palaeolithic)
2. paper files
3. 1950s: punch cards
4. 1960s: tape storage and later disks with random access and the appearance of navigational database systems
5. 1970s: RDBMS (relational database systems)
6. 1980s: the personal computers, object oriented programming
7. 2000s: big data and revisiting the NoSQL databases from the 1950s, NewSQL, MapReduce, Hadoop, etc.

part 03: Data Import

↓

chapter 13:

# RDBMS

# An example of a database: a library

Consider a simple example that will demonstrate the basics of a relational database system. Imagine that we want to create a system that governs a library of books. There are multiple ways to do this, but the following tables are a good choice to get started:

- Authors, with their name, first name, eventually year of birth and death (if applicable) – A table of authors: Table 1 on slide 7;
- Books, with title, author, editor, ISIN, year, number of pages, subject code, etc. – A table of books: Table 2 on slide 8;
- Subject codes, with a description – A table of genres: Table 3 on slide 9.

# The table to store author names

| tbl_authors | | | | |
|---|---|---|---|---|
| id | pen_name | full_name | birth | death |
| PK | | | | |
| 1 | Marcel Proust | Valentin Louis G. E. Marcel Proust | 1871-07-10 | 1922-11-18 |
| 2 | Miguel de Cervantes | Miguel de Cervantes Saavedra | 1547-09-29 | 1616-04-22 |
| 3 | James Joyce | James Augustine Aloysius Joyce | 1882-02-02 | 1941-01-13 |
| 4 | E.L. James | Erika Leonard | 1963-03-07 | |
| 5 | Isaac Newton | Isaac Newton | 1642-12-25 | 1726-03-20 |
| 7 | Euclid | Euclid of Alexandria | Mid-4th C BC | Mid-3rd C BC |
| 11 | Bernard Marr | Bernard Marr | | |
| 13 | Bart Baesens | Bart Baesens | 1975-02-27 | |
| 17 | Philippe De Brouwer | Philippe J.S. De Brouwer | 1969-02-21 | |

**Table 1:** The table of authors for our simple database system.

# The table with books

**tbl_books**

| **id** | **author** | **year** | **title** | **genre** |
|---|---|---|---|---|
| PK | FK | | | FK |
| 1 | 1 | 1896 | Les plaisirs et les jour | LITmod |
| 2 | 1 | 1927 | Albertine disparue | LITmod |
| 4 | 1 | 1954 | Contre Sainte-Beuve | LITmod |
| 5 | 1 | 1871–1922 | À la recherche du temps perdu | LITmod |
| 7 | 2 | 1605 and 1615 | El Ingenioso Hidalgo Don Quijote de la Mancha | LITmod |
| 9 | 2 | 1613 | Novelas ejemplares | LITmod |
| 10 | 4 | 2011 | Fifty Shades of Grey | LITero |
| 15 | 5 | 1687 | PhilosophiæNaturalis Principia Mathematica | SCIphy |
| 16 | 7 | 300 BCE | Elements | SCImat |
| 18 | 13 | 2014 | Big Data World | SCIdat |
| 19 | 11 | 2016 | Key Business Analytics | SCIdat |
| 20 | 17 | 2011 | Malowian Portfolio Theory | SCIfin |

**Table 2:** The table that contains information related to books.

# The table with genres

| tbl_genres | | | |
|---|---|---|---|
| **id (PK)** | **type** | **sub_type** | **location** |
| PK | | | FK |
| LITmod | literature | modernism | 001.45 |
| LITero | literature | erotica | 001.67 |
| SCIphy | science | physics | 200.43 |
| SCImat | science | mathematics | 100.53 |
| SCIbio | science | biology | 300.10 |
| SCIdat | science | data science | 205.13 |
| FINinv | financial | investments | 405.08 |

**Table 3:** A simple example of a relational database system or RDBMS for a simple system for a library. It shows that each piece of information is only stored once and that tables are rectangular data.

part 03: Data Import

↓

chapter 14:

# SQL

PART 03: DATA IMPORT

↓

CHAPTER 14: SQL

↓

SECTION 1:

# Designing the Database

# The Entity Relationship (ER) diagram

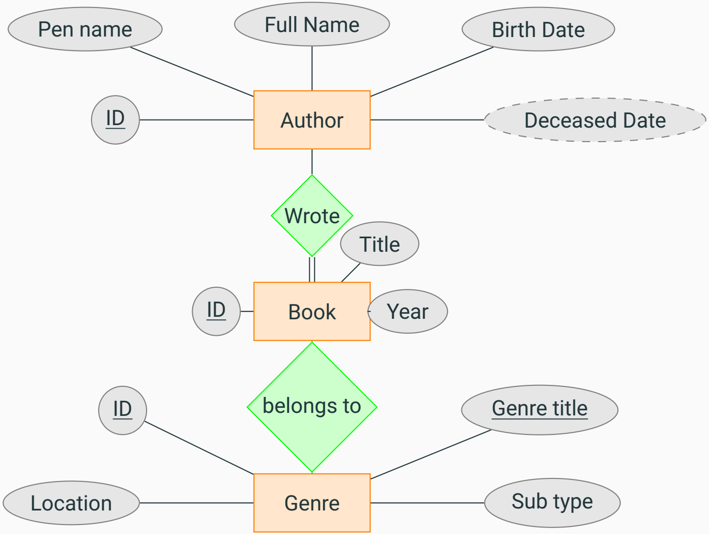


**Figure 1:** The entity relationship (ER) diagram for our example, the library of books.

# The tables and the relations

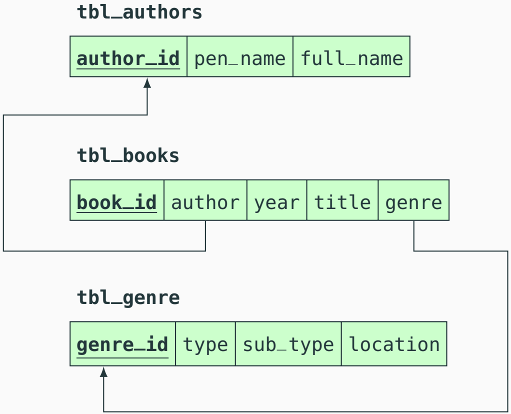


**Figure 2:** The database scheme for the library.

PART 03: DATA IMPORT

↓

CHAPTER 14: SQL

↓

SECTION 2:

# Building the Database Structure

# SQL code to create the database

```
-- First create a database:
CREATE DATABASE library;

-- Create a superuser for that database:
GRANT ALL PRIVILEGES ON library.* To 'libroot'@'localhost' IDENTIFIED BY 'librootPWD';

-- Create also a user who can only update data:
CREATE USER librarian@localhost IDENTIFIED BY 'librarianPWD';
GRANT SELECT, INSERT, UPDATE, DELETE ON library.* TO librarian@localhost;

-- Display a list of tables:
show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| library            |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
-- Note that we did not create the other databases, they are used by MySQL
-- to manage everything.

-- Leave the MySQL terminal:
```

# Start MySQL with the newly created user

At this point, the database administrator for our library, “libroot,” exits. Using this user is safer, because you will not be able to accidentally delete any other database, table or record in a table. We will now log in once more, but with that user-id:

```
mysql -u libroot -p
```

***Listing 2:*** *Starting MySQL, as user “libroot.” Note that this is done from the Linux CLI.*

# Create the tables for the authors in SQL

Now, that we are logged in as `libroot`, we will start to create the tables in which later all data will reside:

```sql
-- First, we need to tell MySQL which database to use.
USE library;

-- Check if the table exists and if so delete it:
DROP TABLE IF EXISTS `tbl_authors`;

-- Then we can start to create tables
CREATE TABLE `tbl_authors`
  (
  author_id         INT UNSIGNED  auto_increment PRIMARY KEY not null,
  pen_name          VARCHAR(100) NOT NULL,
  full_name         VARCHAR(100),
  birth_date        DATE,
  death_date        DATE
  )
  ENGINE INNODB COLLATE 'utf8_unicode_ci';
```

***Listing 3:** Create the table tbl_authors.*

# Important aspects when creating a table i

- `INT` stands for integer and will hold integers. INT will be encoded in 4 bytes. Since each byte is 4 bits, this will hold 32 bits. We need one bit for the sign, so the range is from $-2^{(32-1)}$ to $+2^{(32-1)}$. INT can be signed or unsigned (ie. allowing negative numbers or not). The default is signed, so that does not have to be mentioned. If you will only have positive numbers, one can choose for `UNSIGNED`. MysQL supports the following:
  1. TINYINT = 1 byte (8 bit): from $-128$ to 127 signed and from 0 to 255 unsigned
  2. SMALLINT = 2 bytes (16 bit): from $-32\,768$ to 32 767 signed and from 0 to 65 535 unsigned
  3. MEDIUMINT = 3 bytes (24 bit): from $-8\,388\,608$ to 8 388 607 signed and from 0 to 16 777 215 unsigned
  4. INT = 4 bytes (32 bit): from $-2\,147\,483\,648$ to 2 147 483 647 signed and from 0 to 4 294 967 295 unsigned
  5. BIGINT = 8 bytes (64 bit): from $-2^{63}$ to $-2^{63} - 1$ signed and from 0 to $-2^{64} - 1$ unsigned;
- `PRIMARY KEY` indicates – not surprisingly – that this field is the primary key (PK), hence, it will have to be unique and that is the field that will uniquely define the author;
- `auto_increment` tells MySQL to manage the value of this field by itself: if the user does not provide a unique number, then MySQL will automatically allocate a number that is free when a record is created;
- `VARCHAR(100)` will hold a string up to 100 characters;
- `DATE` will hold dates between 1000-01-01 and 9999-12-31;

# Important aspects when creating a table ii

- We also provide a “collation.” A collation in MySQL is a set of rules that defines how to compare and sort character strings, and is somehow comparable to the typical regional settings. For example, the `utf8_unicode_ci`” implements the standard Unicode Collation Algorithm, it supports expansions and ligatures, for example: German letter ß (U+00DF LETTER SHARP S) is sorted near “ss” Letter œ (U+0152 LATIN CAPITAL LIGATURE OE) is sorted near “OE,” etc. We opt for this collation, because we expect to see many international names in this table.

# Create the tables for the authors in SQL

```
-- If the table already exists, delete it first:
DROP TABLE IF EXISTS `tbl_books`;

-- Create the table tbl_books:
CREATE TABLE `tbl_books`
  (
  book_id             INT unsigned auto_increment
                      PRIMARY KEY not null,
  author              INT unsigned NOT NULL REFERENCES
                      tbl_authors(author_id)
                      ON DELETE RESTRICT,
  year                SMALLINT,     -- provides 30~000 years
  title               VARCHAR(50),  -- maximum 50 characters
  genre               CHAR(6) NOT NULL REFERENCES -- always 6
                      tbl_genres(genre_id) ON DELETE RESTRICT
  )
  ENGINE INNODB COLLATE 'utf8_unicode_ci';

-- Create an index for speedy lookup on those fields:
CREATE INDEX idx_book_author ON tbl_books (author);
CREATE INDEX idx_book_genre ON tbl_books (genre);
```

***Listing 4:*** *This SQL code block creates the table* `tbl_books` *and then define an index on two of its fields.*

# Manage the indexes of the tables

```sql
--  Create an index where the values must be unique
-- (except for the NULL values, which may appear multiple times):
ALTER TABLE tbl_name ADD UNIQUE index_name (column_list);

-- Create an index in which any value may appear more than once:
ALTER TABLE tbl_name ADD INDEX index_name (column_list);

-- A special index that helps searching in full text:
ALTER TABLE tbl_name ADD FULLTEXT index_name (column_list);

-- We can also add a primary key:
ALTER TABLE tbl_name ADD PRIMARY KEY (column_list);

-- Drop an index:
ALTER TABLE table_name DROP INDEX index_name;

-- Drop a primary key:
ALTER TABLE tbl_name DROP PRIMARY KEY;
```

***Listing 5:*** *Manage indexes in MySQL*

# Create the tables for the genres in SQL

```
-- In case it would already exist, delete it first:
DROP TABLE IF EXISTS `tbl_genres`;

-- Create the table:
CREATE TABLE `tbl_genres`
  (
  genre_id          CHAR(6) PRIMARY KEY not null,
  type              VARCHAR(20),
  sub_type          VARCHAR(20),
  location          CHAR(7)
  )
  ENGINE INNODB COLLATE 'utf8_unicode_ci';

-- Show the tables in our database:
show tables;
+-------------------+
| Tables_in_library |
+-------------------+
| tbl_authors       |
| tbl_books         |
| tbl_genres        |
+-------------------+
3 rows in set (0,01 sec)
```

***Listing 6:*** *This SQL code creates the table tbl_genres and then checks if it is really there.*

# Inspect the results

It is possible to check our work with `DESCRIBE TABLE`. That command will show relevant details about the table and its field:

```
mysql> DESCRIBE tbl_genres;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| genre_id | char(6)     | NO   | PRI | NULL    |       |
| type     | varchar(20) | YES  |     | NULL    |       |
| sub_type | varchar(20) | YES  |     | NULL    |       |
| location | char(7)     | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0,00 sec)
```

***Listing 7:*** *Checking the structure of the table* `tbl_books`*.*

PART 03: DATA IMPORT

↓

CHAPTER 14: SQL

↓

SECTION 3:

# Adding Data to the Database

# Log into the database as librarian

Since we did the effort to create a username for updating the data, let us use it. In MySQL, type `exit` or `\q` followed by enter and then login again from the command prompt.

```
mysql -u librarian -p
```

***Listing 8:*** *Logging in as user "librarian."*

# Add one record

Let us start by adding one record:

```sql
INSERT INTO tbl_authors
  VALUES (1, "Philppe J.S. De Brouwer", "Philppe J.S. De Brouwer", "1969-02-21", NULL);
```

***Listing 9:*** *Adding our first author to the database.*

**Note – Providing values for automatically incremented fields**

While MySQL was supposed to manage the `author_id`, it did not complain when it was coerced in using a certain value (given that the value is in range, of good type and of course still free). Without this lenience, uploading data or restoring and SQL-dump would not always be possible.

# Add multiple records i

Of course, it is also possible to add multiple records in one statement:

```
-- First, remove all our testing (1 is equivalent with TRUE):
DELETE FROM tbl_authors WHERE 1;

-- Since we provide a value for each row, we can omit the
-- fields, though it is better to make it explicit when
-- inserting the data:
INSERT INTO tbl_authors (author_id, pen_name, full_name, birth_date, death_date)
  VALUES
  (1 , "Marcel Proust",
       "Valentin Louis G. E. Marcel Proust",
       "1871-07-10", "1922-11-18"),
  (2 , "Miguel de Cervantes",
       "Miguel de Cervantes Saavedra",
       "1547-09-29", "1616-04-22"),
  (3 , "James Joyce",
       "James Augustine Aloysius Joyce",
       "1882-02-02", "1941-01-13"),
  (4 , "E. L. James",         "Erika Leonard",
       "1963-03-07",  NULL),
  (5 , "Isaac Newton",        "Isaac Newton",
       "1642-12-25", "1726-03-20"),
  (7 , "Euclid",              "Euclid of Alexandria",
        NULL,         NULL),
  (11, "Bernard Marr",        "Bernard Marr",
        NULL,         NULL),
  (13, "Bart Baesens",        "Bart Baesens",
       "1975-02-27",  NULL),
  (14, "Philippe J.S. De Brouwer",
       "Philippe J.S. De Brouwer",
       "1969-02-21",  NULL)
  ;
```

***Listing 10:*** *This SQL code adds all books in one statement.*

# Inserting data in the genres table

```
INSERT INTO tbl_genres (genre_id, type, sub_type, location)
  VALUES
  ("LITmod", "literature", "modernism",    "001.45"),
  ("LITero", "literature", "erotica",      "001.67"),
  ("SCIphy", "science",    "physics",      "200.43"),
  ("SCImat", "science",    "mathematics",  "100.53"),
  ("SCIbio", "science",    "biology",      "300.10"),
  ("SCIdat", "science",    "data science", "205.13"),
  ("FINinv", "financial",  "investments",  "405.08")
  ;
```

***Listing 11:*** *Add the data to the table tbl_genres.*

# Inserting data in the table for books

```
INSERT INTO tbl_books (author, year, title, genre)
  VALUES
  (1,  1896, "Les plaisirs et les jour",      "LITmod"),
  (1,  1927, "Albertine disparue",            "LITmod"),
  (1,  1954, "Contre Sainte-Beuve",           "LITmod"),
  (1,  1922, "AÌĂ la recherche du temps perdu", "LITmod"),
  (2,  1615, "El Ingenioso Hidalgo Don Quijote de la Mancha", "LITmod"),
  (2,  1613, "Novelas ejemplares",            "LITmod"),
  (4,  2011, "Fifty Shades of Grey",          "LITero"),
  (5,  1687, "PhilosophiÃę Naturalis Principia Mathematica", "SCIphy"),
  (7,  -300, "Elements (translated )",        "SCImat"),
  (13, 2014, "Big Data World",                "SCIdat"),
  (11, 2016, "Key Business Analytics",        "SCIdat"),
  (14, 2011, "Maslowian Portfolio Theory",    "FINinv")
  ;
```

***Listing 12:*** *Add the data to the table tbl_books.*

PART 03: DATA IMPORT

↓

CHAPTER 14: SQL

↓

SECTION 4:

# Querying the Database

# SELECT queries are intuitive

```
-- Show all info about all authors:
SELECT * from tbl_authors;

-- Show all pen_names and birth_dates from tbl_authors:
SELECT pen_name, birth_date FROM tbl_authors;

-- Show all authors from the last two centuries
SELECT pen_name FROM tbl_authors WHERE birth_date > DATE("1900-01-01");

-- Include also the ones that have no birth data in the system"
SELECT pen_name FROM tbl_authors
    WHERE (
           (birth_date > DATE("1900-01-01"))
           OR
           (ISNULL(birth_date))
           ) ;
```

***Listing 13:*** *Some example of SELECT-queries. Note that the output is not shown here, simply because it would be too long.*

part 03: Data Import

↓

chapter 14: SQL

↓

section 5:

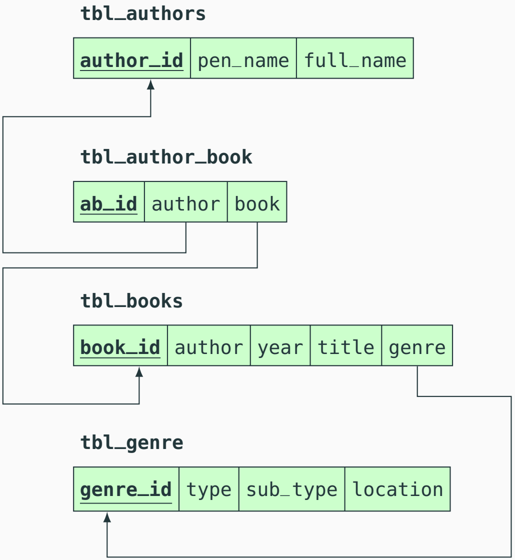
# Modifying the Database Structure

# How to add a book with multiple authors

This is a great day today. We received finally our copy of Hadley Wickham and Garret Gerolemund's book "R for Data Science" and we want to add it to our library. However, we can enter only one reference to one author in our library. After a brainstorm meeting, we come up with the following solutions:

1. Pretend that this book did not arrive, send it back or make it disappear: adapt reality to the limitations of our computer system.
2. Put one of the two authors in the system and hope this specific issue does not occur to often.[1]
3. Add a second `author`-field to the table `tbl_books`. That would solve the case for two authors, but not for three or more.
4. Add 10 additional fields as described above. This would indeed solve most cases, but we still would have to re-write all queries in a non-obvious way. Worse, most queries will just run and we will only find out later that something was not as expected. Also, we feel that this solution is not elegant at all.
5. Add a table that links the authors and the books. This will solution would allow us to record between zero and a huge amount of authors. This would be a fundamentally different database design and if the library software would already be written[2] this solution might not pass the Pareto rule.

# The updated database design



**tbl_author_book**

| **ab_id** | author | book |
|---|---|---|

**tbl_authors**

| **author_id** | pen_name | full_name |
|---|---|---|

**tbl_books**

| **book_id** | author | year | title | genre |
|---|---|---|---|---|

**tbl_genre**

| **genre_id** | type | sub_type | location |
|---|---|---|---|

**Figure 3:** The improved database scheme that allows multiple authors to co-author one book by adding a table `tbl_author_book`

# Implementing this design

```
-- Note that this has to be done as libroot or root,
-- the user librarian cannot do this!
-- Note also the different cascading rules. Why did we do so?
DROP TABLE IF EXISTS `tbl_author_book`;
CREATE TABLE `tbl_author_book`
  (
  ab_id  INT unsigned auto_increment PRIMARY KEY not null,
  author INT unsigned NOT NULL
         REFERENCES tbl_authors(author_id) ON DELETE RESTRICT,
  book   INT unsigned NOT NULL
         REFERENCES tbl_books(book_id) ON DELETE CASCADE
  );

-- Ensure the combination of author/book appears only once:
ALTER TABLE `tbl_author_book`
                   ADD UNIQUE `unique_index`(`author`, `book`);

-- Insert all pairs of authors and books that we already know:
INSERT INTO tbl_author_book (author, book)
  (SELECT author_id, book_id FROM tbl_authors, tbl_books
  WHERE tbl_books.author = author_id
  );

-- We can just drop the field author from the table tbl_books
-- and the link automatically disappears.
```

# Using the new database

Also retrieving information will be different. For example, finding all the books of a given author can work as follows:

```
SELECT pen_name, title FROM tbl_authors, tbl_author_book, tbl_books
  WHERE
    (author_id = tbl_author_book.author) AND
    (book_id   = tbl_author_book.book)   AND
    (pen_name LIKE '%oust%')
    ;

-- MySQL will then reply this:
+---------------+---------------------------------+
| pen_name      | title                           |
+---------------+---------------------------------+
| Marcel Proust | Les plaisirs et les jour        |
| Marcel Proust | Albertine disparue              |
| Marcel Proust | Contre Sainte-Beuve             |
| Marcel Proust | A la recherche du temps perdu   |
+---------------+---------------------------------+
```

PART 03: DATA IMPORT

↓

CHAPTER 14: SQL

↓

SECTION 6:

# Selected Features of SQL

# The keyword SET

The command SET also allows to update all selected variables in a table. For example, we can capitalize all author names as follows:

```sql
UPDATE tbl_authors
  SET full_name = CONCAT(UPPER(SUBSTRING(full_name,1,1)),
                         SUBSTRING(full_name,2,LENGTH(full_name)))
  WHERE 1;
```

***Listing 15:*** *Capitalize all first letter of all full names of authors.*

# A function in SQL to calculate the average

```
delimiter //
CREATE PROCEDURE CalcAvgBooks (OUT avgBooks INT)
  BEGIN
  SELECT AVG(nbrBooks) INTO avgBooks FROM (SELECT COUNT(*) AS nbrBooks  FROM tbl_author_book GROUP BY
     author);
  END//
delimiter ;

-- Now use the function:
CALL CalcAvgBooks(@myAVG);

-- Use now the parameter myAVG
SELECT CONCAT('The average number of books per author in our libaray is: ',@myAvg);
```

***Listing 16:*** *Creating a function and using it in SQL.*

part 03: Data Import

↓

chapter 15:

# Connecting R to an SQL Database

# RMySQL: setting it up

With the package RMySQL, it is possible to both connect to MariaDB and MySQL in a convenient way and copy the data to R for further analysis. The basics of the package are to create a connection variable first and then use that connection to retrieve data.

```
# install.packages('RMySQL')
library(RMySQL)
# connect to the library
con <- dbConnect(MySQL(),
                 user     = "librarian",
                 password = "librarianPWD",
                 dbname   = "library",
                 host     = "localhost"
                 )

# in case we would forget to disconnect:
on.exit(dbDisconnect(con))
```

# RMySQL: using the open connection

Now, we have the connection stored in the object con and can use this to display data about the connection, run queries, and retrieve data.

```
# Show some information:

show(con)
summary(con, verbose = TRUE)
# dbGetInfo(con)  # similar as above but in list format
dbListResults(con)
dbListTables(con) # check: this might generate too much output

# Get data:
df_books <- dbGetQuery(con, "SELECT COUNT(*) AS nbrBooks
                     FROM tbl_author_book GROUP BY author;")

# Now, df_books is a data frame that can be used as usual.

# close the connection:
dbDisconnect(con)
```

# Define functions to connect to the database i

The code below does the same as the aforementioned code, but with our own custom functions that are a wrapper for opening the connection, running the query, returning the data and closing the connection. We strongly recommend to use this version.

# Define functions to connect to the database ii

```r
# Load the package:
library(RMySQL)

# db_get_data
# Get data from a MySQL database
# Arguments:
#    con_info -- MySQLConnection object -- the connection info to
#                                          the MySQL database
#    sSQL     -- character string       -- the SQL statement that
#                                          selects the records
# Returns
#    data.frame, containing the selected records
db_get_data <- function(con_info, sSQL){
  con <- dbConnect(MySQL(),
                   user     = con_info$user,
                   password = con_info$password,
                   dbname   = con_info$dbname,
                   host     = con_info$host
                   )
  df <- dbGetQuery(con, sSQL)
  dbDisconnect(con)
  df
}
```

# Define functions to connect to the database iii

```
# db_run_sql
# Run a query that returns no data in an MySQL database
# Arguments:
#    con_info -- MySQLConnection object -- open connection
#    sSQL     -- character string       -- the SQL statement to run
db_run_sql <-function(con_info, sSQL)
{
  con <- dbConnect(MySQL(),
                 user     = con_info$user,
                 password = con_info$password,
                 dbname   = con_info$dbname,
                 host     = con_info$host
                 )
  rs <- dbSendQuery(con,sSQL)
  dbDisconnect(con)
}
```

# Creating a historgram of books per author i

```r
# use the wrapper functions to get data.

# step 1: define the connection info
my_con_info <- list()
my_con_info$user     <- "librarian"
my_con_info$password <- "librarianPWD"
my_con_info$dbname   <- "library"
my_con_info$host     <- "localhost"


# step 2: get the data
my_query <- "SELECT COUNT(*) AS nbrBooks
                     FROM tbl_author_book GROUP BY author;"
df <- db_get_data(my_con_info, my_query)

# step 3: use this data to produce the histogram:
hist(df$nbrBooks, col='khaki3')
```

# Creating a historgram of books per author ii

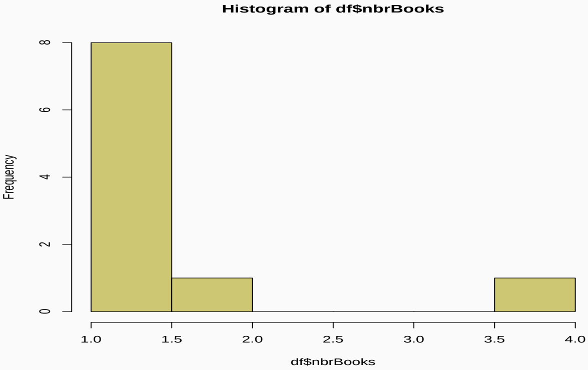


**Figure 4:** Histogram generated with data from the MySQL database.